Revisiting the Kernel Security Enhancements in iOS 10

MOSEC 2017

Liang Chen @ Tencent Keen Lab

Last year in iOS 10

- 2016.6.13 first iOS 10.0 beta was released
  - 2016.7 Pangu team released iOS 9.3.3 jailbreak. Why?
  - 2016.8 First iOS APT, Pegasus, was observed

- 2016.9.13 first release version of iOS 10 was out

- 2016.10.24 iOS 10.1 was released
  - 2016.10.25, Keen Lab pwned iOS 10.1 twice in Pwn2Own(remotely stealing photos, remotely install bogus app

Last year in iOS 10 (cont.)

- 2016.12.12 iOS 10.2 was released
  - Ian Beer of Google Project Zero released mach_portal exploit with a nice write-up
  - Luca Todesco released Yalu + mach_portal, Jailbreak on iOS 10.1.1 (including iPhone 7 AMCC bypass)
  - Yalu102 (by Luca Todesco and Marco Grassi)
- 2017.3.27 iOS 10.3 was released
  - Everything became quite after that
- Question: What happened in this year?

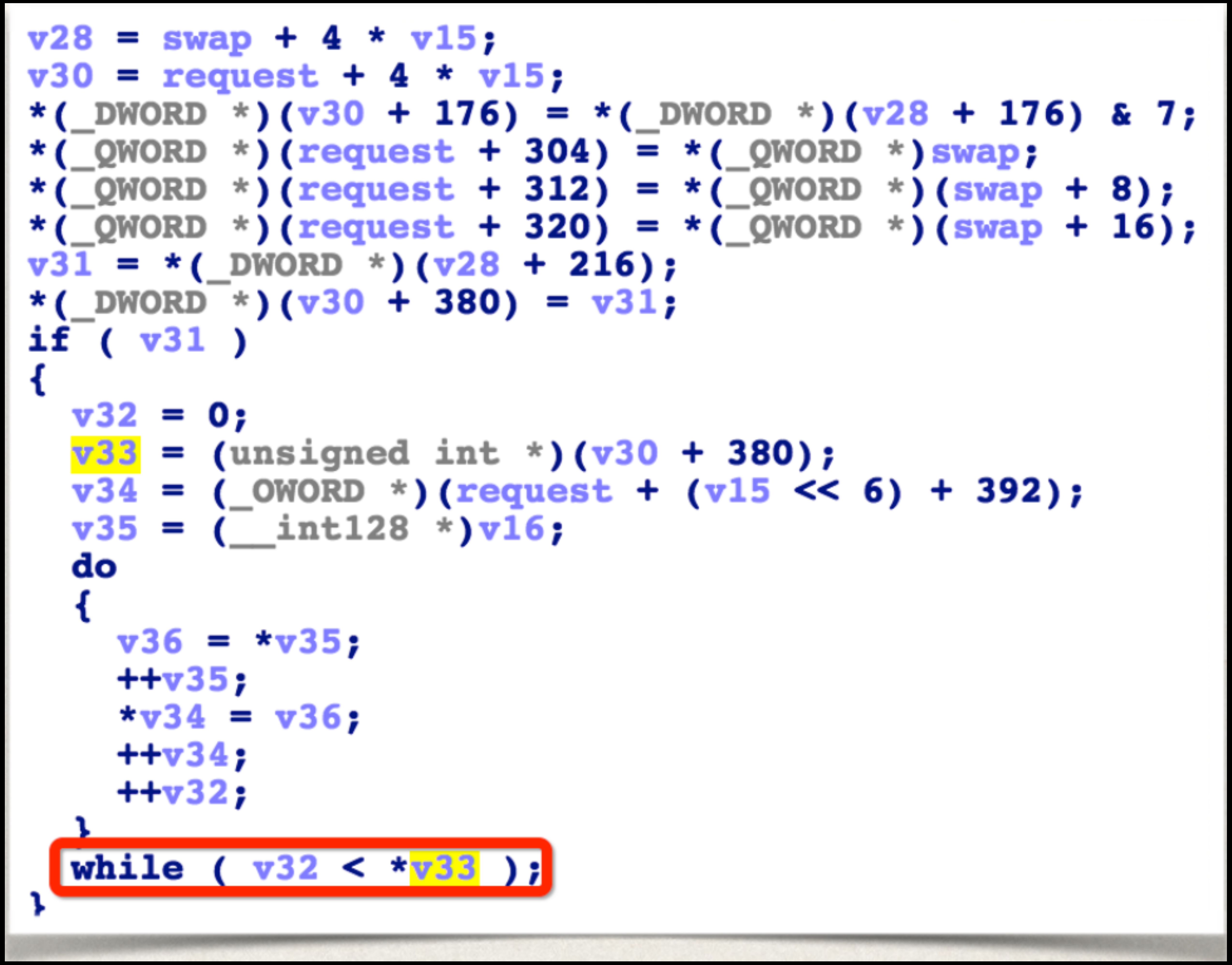
Agenda

- Vulnerability
- Mechanism
- Exploitation
- Summary

Part I: Vulnerability

Begin with Pangu 9.3.3…

- CVE-2016-4654: Heap overflow in IOMobileFramebufferUserClient
- In the 5th methodCall of IOMobileFramebufferUserClient(IOMobileFramebuffer::swap_submit)

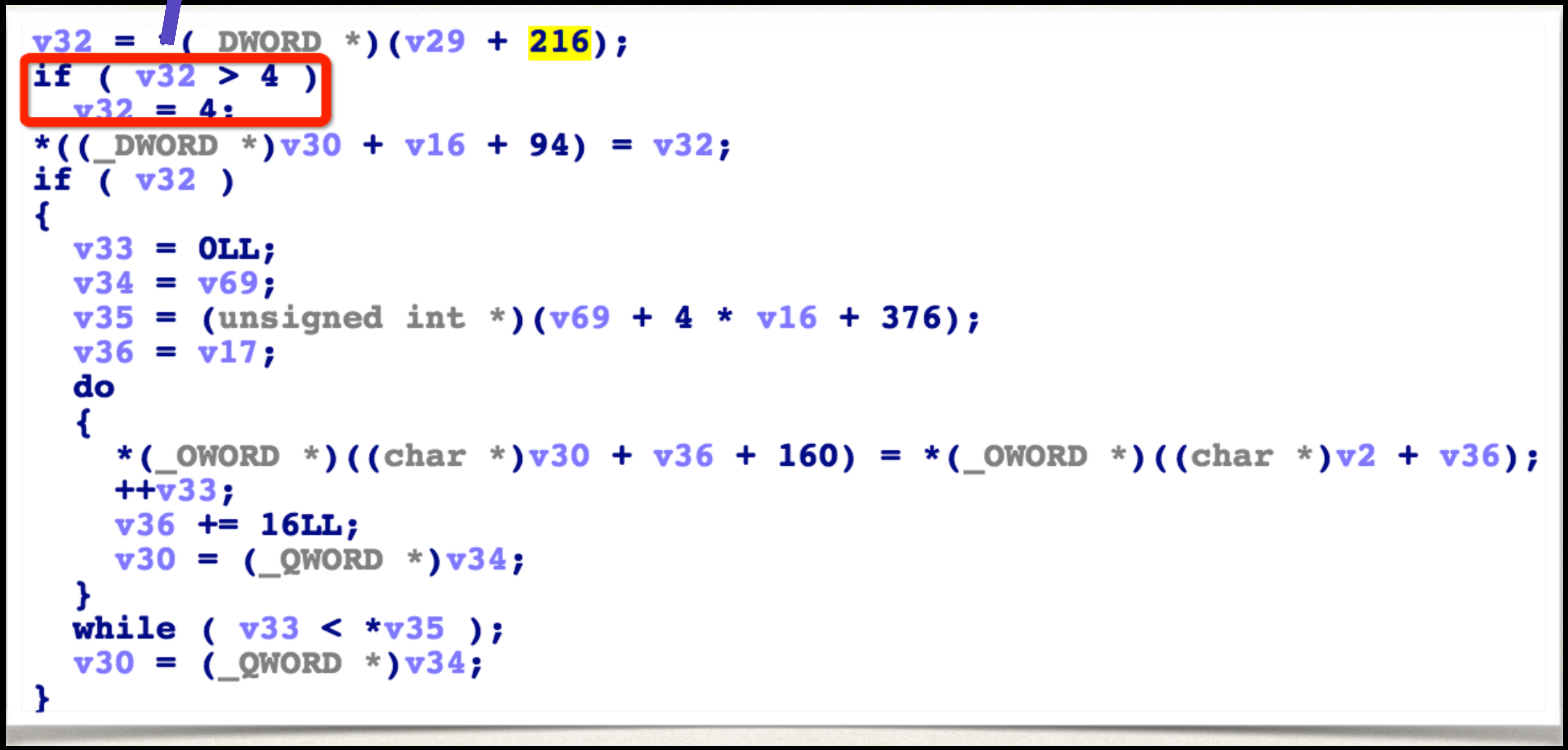


```
v28 = swap + 4 * v15;
v30 = request + 4 * v15;
*(_DWORD *)(v30 + 176) = *(_DWORD *)(v28 + 176) & 7;
*(_QWORD *)(request + 304) = *(_QWORD *)swap;
*(_QWORD *)(request + 312) = *(_QWORD *)(swap + 8);
*(_QWORD *)(request + 320) = *(_QWORD *)(swap + 16);
v31 = *(_DWORD *)(v28 + 216);
*(_DWORD *)(v30 + 380) = v31;
if ( v31 )
{
  v32 = 0;
  v33 = (unsigned int *)(v30 + 380);
  v34 = (_OWORD *)(request + (v15 << 6) + 392);
  v35 = (__int128 *)v16;
  do
  {
    v36 = *v35;
    ++v35;
    *v34 = v36;
    ++v34;
    ++v32;
  }
  while ( v32 < *v33 );
}
```

CVE-2016-4654: the fix

- Fixed in iOS 10.0 beta 2
- someCount cannot exceed 4
- We can get conclusion:
  - Before iOS 10.0.1 release, Apple made strict audit on iOS 9 code
  - Several unfixed bugs were patched in iOS 10 beta

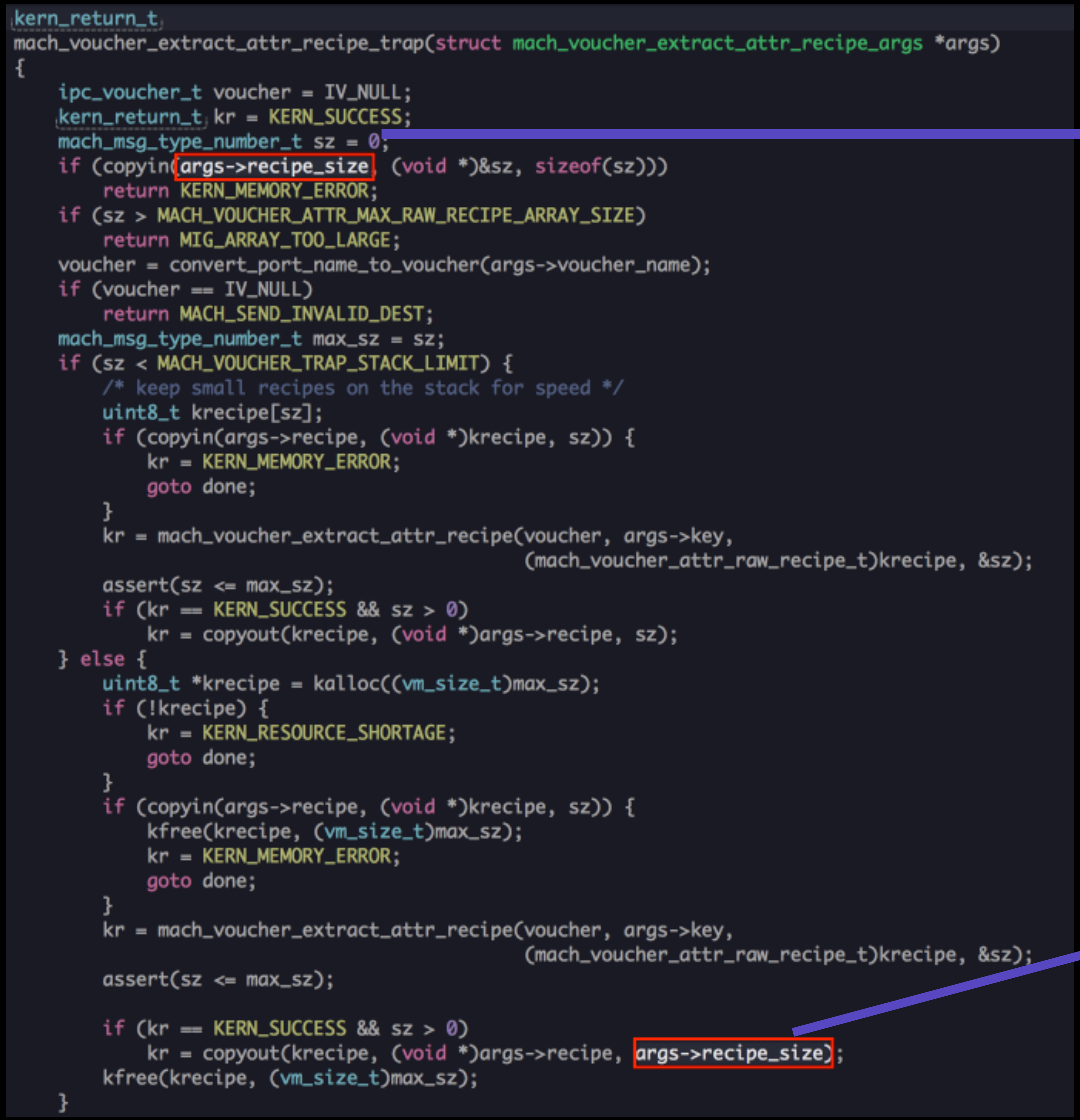
someCount not exceed 4



```
v32 = *(_DWORD *)(v29 + 216);
if ( v32 > 4 )
  v32 = 4;
*((_DWORD *)v30 + v16 + 94) = v32;
if ( v32 )
{
  v33 = 0LL;
  v34 = v69;
  v35 = (unsigned int *)(v69 + 4 * v16 + 376);
  v36 = v17;
  do
  {
    *(_OWORD *)((char *)v30 + v36 + 160) = *(_OWORD *)((char *)v2 + v36);
    ++v33;
    v36 += 16LL;
    v30 = (_QWORD *)v34;
  }
  while ( v33 < *v35 );
  v30 = (_QWORD *)v34;
}
```

XNU case: CVE-2017-2370

- Discovered by Marco Grassi of Keen Lab, bug collision with Ian Beer later on
- Heap overflow in mach_voucher_extract_attr_recipe



```
kern_return_t
mach_voucher_extract_attr_recipe_trap(struct mach_voucher_extract_attr_recipe_args *args)
{
    ipc_voucher_t voucher = IV_NULL;
    kern_return_t kr = KERN_SUCCESS;
    mach_msg_type_number_t sz = 0;
    if (copyin(args->recipe_size, (void *)&sz, sizeof(sz)))
        return KERN_MEMORY_ERROR;
    if (sz > MACH_VOUCHER_ATTR_MAX_RAW_RECIPE_ARRAY_SIZE)
        return MIG_ARRAY_TOO_LARGE;
    voucher = convert_port_name_to_voucher(args->voucher_name);
    if (voucher == IV_NULL)
        return MACH_SEND_INVALID_DEST;
    mach_msg_type_number_t max_sz = sz;
    if (sz < MACH_VOUCHER_TRAP_STACK_LIMIT) {
        /* keep small recipes on the stack for speed */
        uint8_t krecipe[sz];
        if (copyin(args->recipe, (void *)krecipe, sz)) {
            kr = KERN_MEMORY_ERROR;
            goto done;
        }
        kr = mach_voucher_extract_attr_recipe(voucher, args->key,
                                              (mach_voucher_attr_raw_recipe_t)krecipe, &sz);
        assert(sz <= max_sz);
        if (kr == KERN_SUCCESS && sz > 0)
            kr = copyout(krecipe, (void *)args->recipe, sz);
    } else {
        uint8_t *krecipe = kalloc((vm_size_t)max_sz);
        if (!krecipe) {
            kr = KERN_RESOURCE_SHORTAGE;
            goto done;
        }
        if (copyin(args->recipe, (void *)krecipe, sz)) {
            kfree(krecipe, (vm_size_t)max_sz);
            kr = KERN_MEMORY_ERROR;
            goto done;
        }
        kr = mach_voucher_extract_attr_recipe(voucher, args->key,
                                              (mach_voucher_attr_raw_recipe_t)krecipe, &sz);
        assert(sz <= max_sz);

        if (kr == KERN_SUCCESS && sz > 0)
            kr = copyout(krecipe, (void *)args->recipe, args->recipe_size);
        kfree(krecipe, (vm_size_t)max_sz);
    }
```

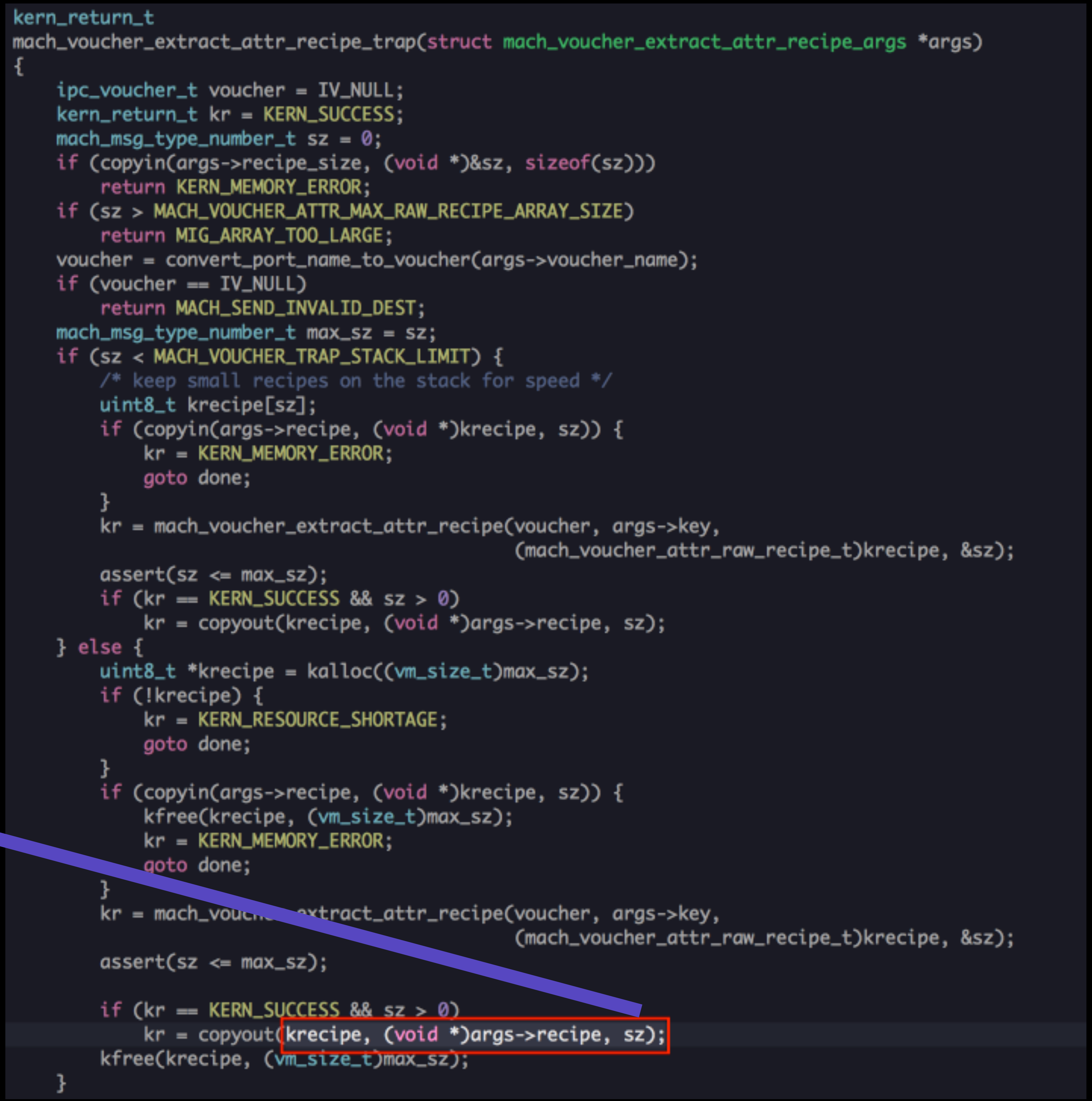
args->recipe_size is a userland pointer pointing to the size value

args->recipe_size is used as size value here

CVE-2017-2370: the fix

- Fixed in iOS 10.2.1
- Lesson learned
  - Newly added interfaces or features are more likely to be vulnerable

Length of copyout is changed to sz (correct value)



```
kern_return_t
mach_voucher_extract_attr_recipe_trap(struct mach_voucher_extract_attr_recipe_args *args)
{
    ipc_voucher_t voucher = IV_NULL;
    kern_return_t kr = KERN_SUCCESS;
    mach_msg_type_number_t sz = 0;
    if (copyin(args->recipe_size, (void *)&sz, sizeof(sz)))
        return KERN_MEMORY_ERROR;
    if (sz > MACH_VOUCHER_ATTR_MAX_RAW_RECIPE_ARRAY_SIZE)
        return MIG_ARRAY_TOO_LARGE;
    voucher = convert_port_name_to_voucher(args->voucher_name);
    if (voucher == IV_NULL)
        return MACH_SEND_INVALID_DEST;
    mach_msg_type_number_t max_sz = sz;
    if (sz < MACH_VOUCHER_TRAP_STACK_LIMIT) {
        /* keep small recipes on the stack for speed */
        uint8_t krecipe[sz];
        if (copyin(args->recipe, (void *)krecipe, sz)) {
            kr = KERN_MEMORY_ERROR;
            goto done;
        }
        kr = mach_voucher_extract_attr_recipe(voucher, args->key,
                                              (mach_voucher_attr_raw_recipe_t)krecipe, &sz);
        assert(sz <= max_sz);
        if (kr == KERN_SUCCESS && sz > 0)
            kr = copyout(krecipe, (void *)args->recipe, sz);
    } else {
        uint8_t *krecipe = kalloc((vm_size_t)max_sz);
        if (!krecipe) {
            kr = KERN_RESOURCE_SHORTAGE;
            goto done;
        }
        if (copyin(args->recipe, (void *)krecipe, sz)) {
            kfree(krecipe, (vm_size_t)max_sz);
            kr = KERN_MEMORY_ERROR;
            goto done;
        }
        kr = mach_voucher_extract_attr_recipe(voucher, args->key,
                                              (mach_voucher_attr_raw_recipe_t)krecipe, &sz);
        assert(sz <= max_sz);

        if (kr == KERN_SUCCESS && sz > 0)
            kr = copyout(krecipe, (void *)args->recipe, sz);
        kfree(krecipe, (vm_size_t)max_sz);
    }
```

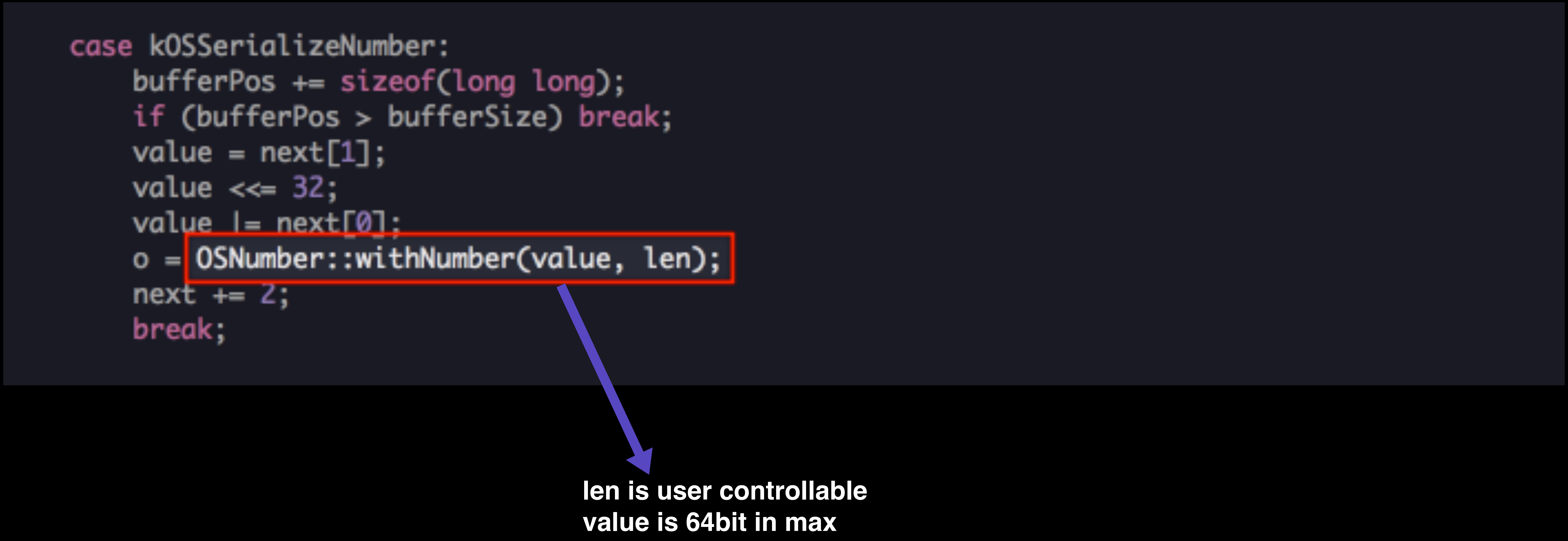
Part II: Mechanism

Story of OSNumber: From Pegasus

- CVE-2016-4655 kernel stack info leak
- OSUnserializeXML receives data from userland, and deserialize into basic data structure in kernelland (E.g OSDictionary, OSArray)
- OSUnserializeXML receives two kind of XML data
  - Binary mode
  - XML mode

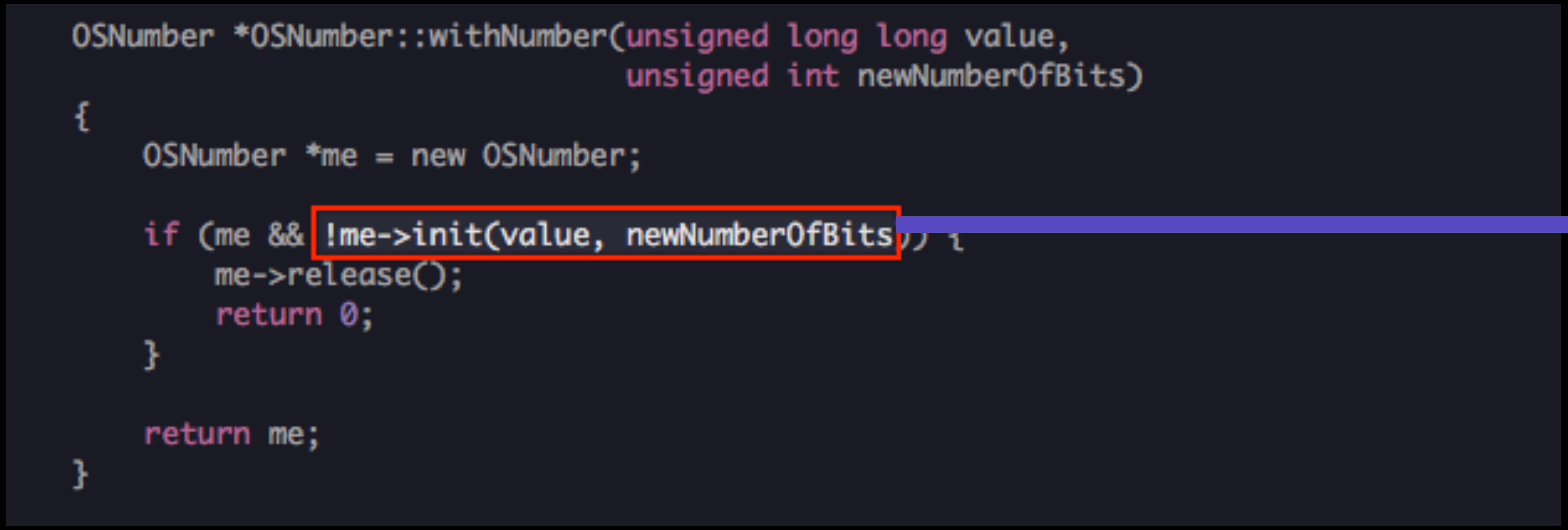
Story of OSNumber: CVE-2016-4655 details

- When binary mode is used, OSUnserializeBinary is called to parse the data



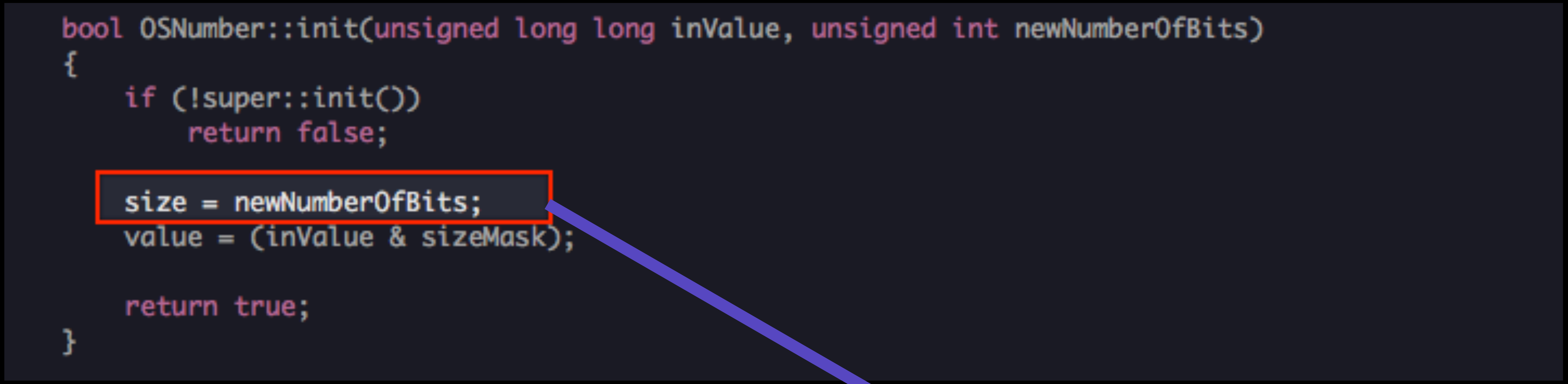
```
case kOSSerializeNumber:
    bufferPos += sizeof(long long);
    if (bufferPos > bufferSize) break;
    value = next[1];
    value <<= 32;
    value |= next[0];
    o = OSNumber::withNumber(value, len);
    next += 2;
    break;
```

Story of OSNumber: CVE-2016-4655 details



```
OSNumber *OSNumber::withNumber(unsigned long long value,
                               unsigned int newNumberOfBits)
{
    OSNumber *me = new OSNumber;

    if (me && !me->init(value, newNumberOfBits)) {
        me->release();
        return 0;
    }

    return me;
}
```

newNumberOfBits is user controllable

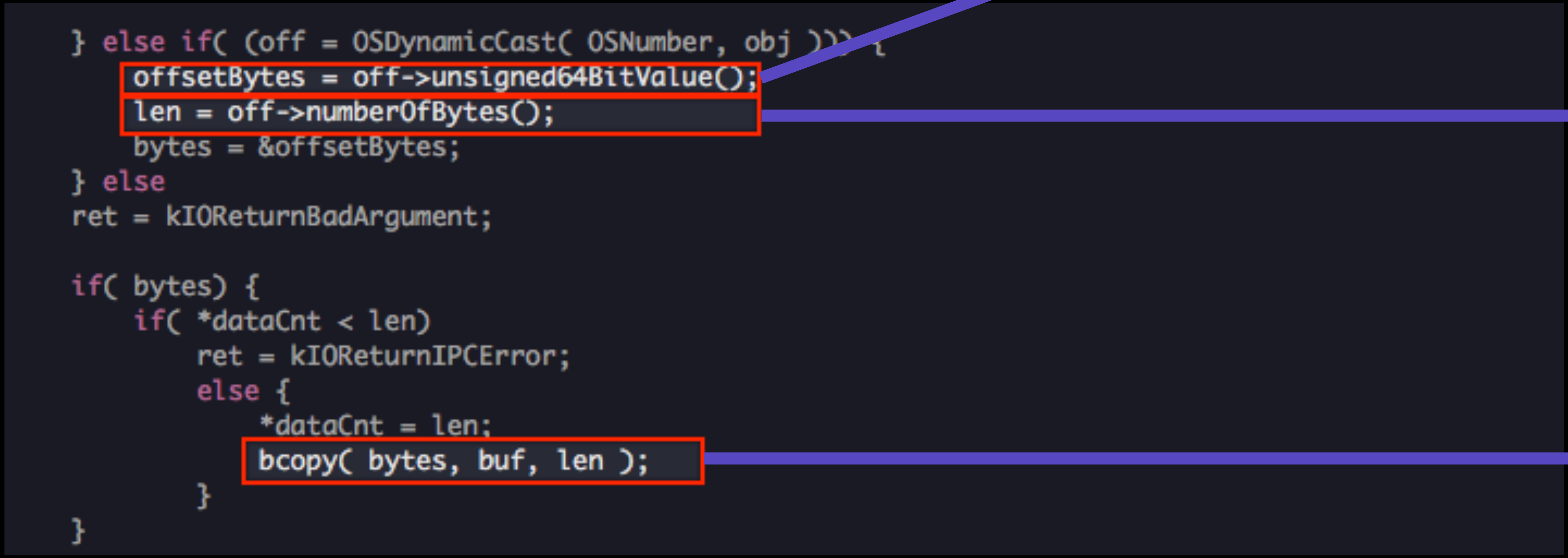


```
bool OSNumber::init(unsigned long long inValue, unsigned int newNumberOfBits)
{
    if (!super::init())
        return false;

    size = newNumberOfBits;
    value = (inValue & sizeMask);

    return true;
}
```

Size can be set to arbitrary value, but value is 64bit in max (8 bytes)

Story of OSNumber: CVE-2016-4655 details

- How to leak?
- is_io_registry_entry_get_property_bytes



```
} else if( (off = OSDynamicCast( OSNumber, obj ))) {
    offsetBytes = off->unsigned64BitValue();
    len = off->numberOfBytes();
    bytes = &offsetBytes;
} else
ret = kIOReturnBadArgument;

if( bytes) {
    if( *dataCnt < len)
        ret = kIOReturnIPCError;
    else {
        *dataCnt = len;
        bcopy( bytes, buf, len );
    }
}
```

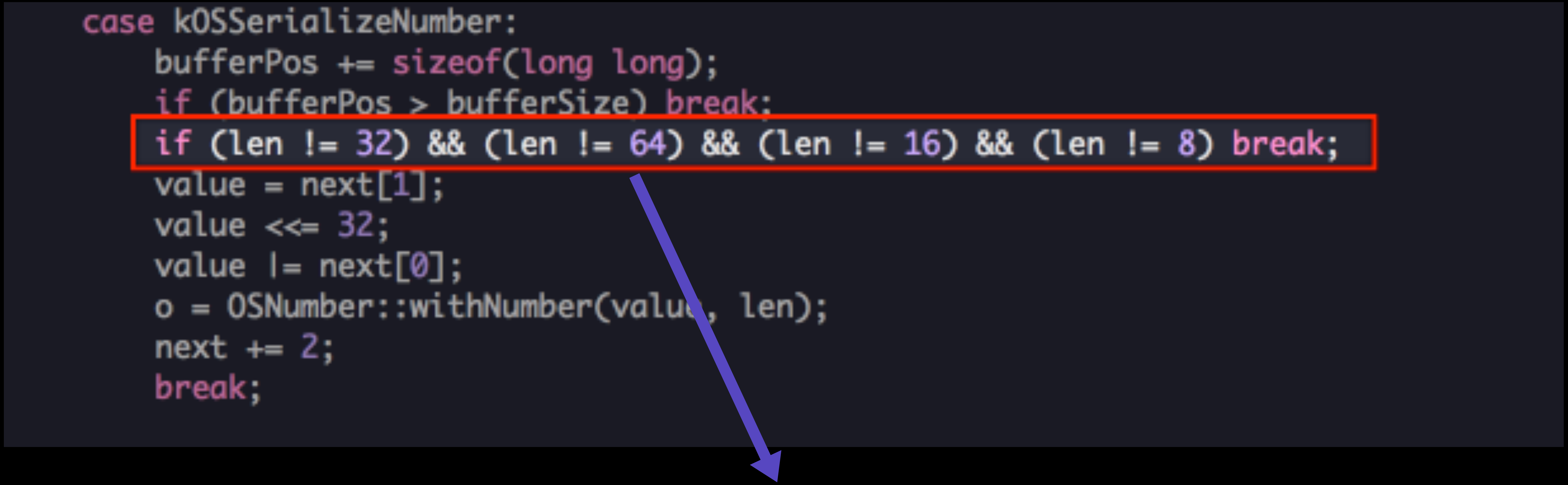
offsetBytes takes 8 byte memory on stack

Len is user controllable

OOB read arbitrary bytes of memory on stack

CVE-2016-4655: the fix

- Fixed in iOS 10.0.1
- In OSUnserializeBinary, only numbers of 8 bits, 16 bits, 32 bits and 64 bits are valid
- Apparently not the standard approach to fix. But for iOS , it might be enough



```
    case kOSSerializeNumber:
        bufferPos += sizeof(long long);
        if (bufferPos > bufferSize) break;
        if (len != 32) && (len != 64) && (len != 16) && (len != 8) break;
        value = next[1];
        value <<= 32;
        value |= next[0];
        o = OSNumber::withNumber(value, len);
        next += 2;
        break;
```

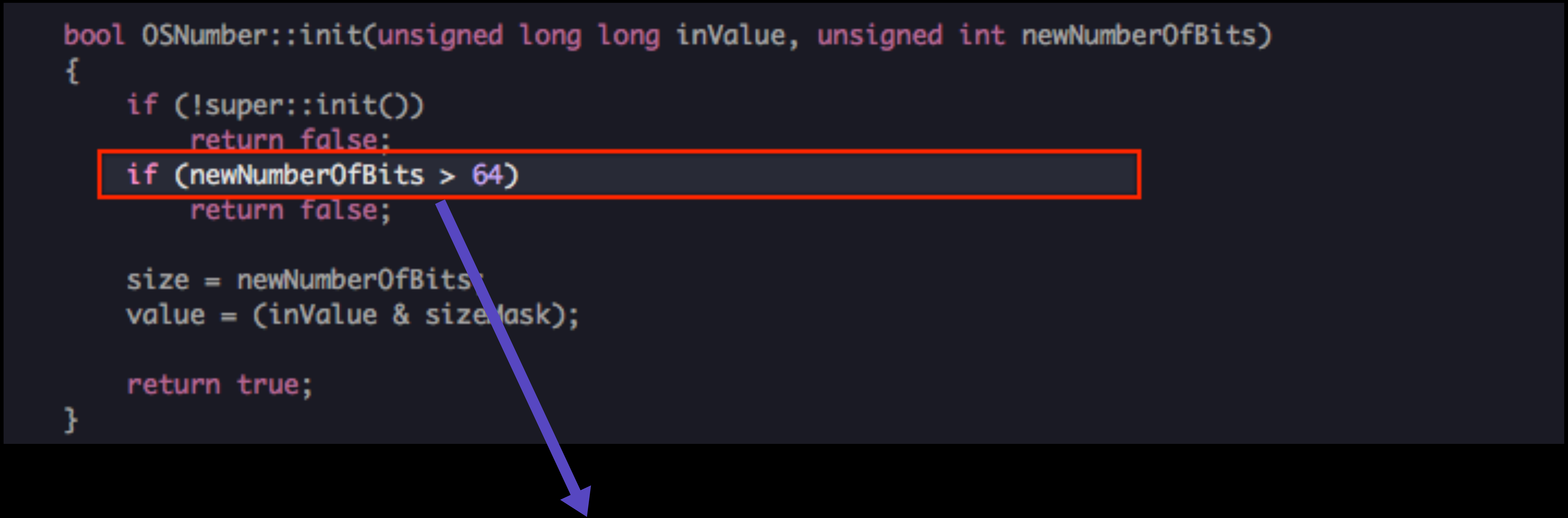
Only allow numbers of 4 modes

OSNumber: any more problems

- OSUnserializeXML receives two kind of XML data…
- Binary mode fixed
- Try XML mode
  - <integer size="100">0X41414141</integer>
- Conclusion: iOS 10.0.1 was once again successfully leaked!

OSNumber bug 2: XML mode of OSUnserializeXML

- Seems apple noticed the issue very soon, and fixed in iOS 10.1
- This time they decided to fix the issue in OSNumber implementation

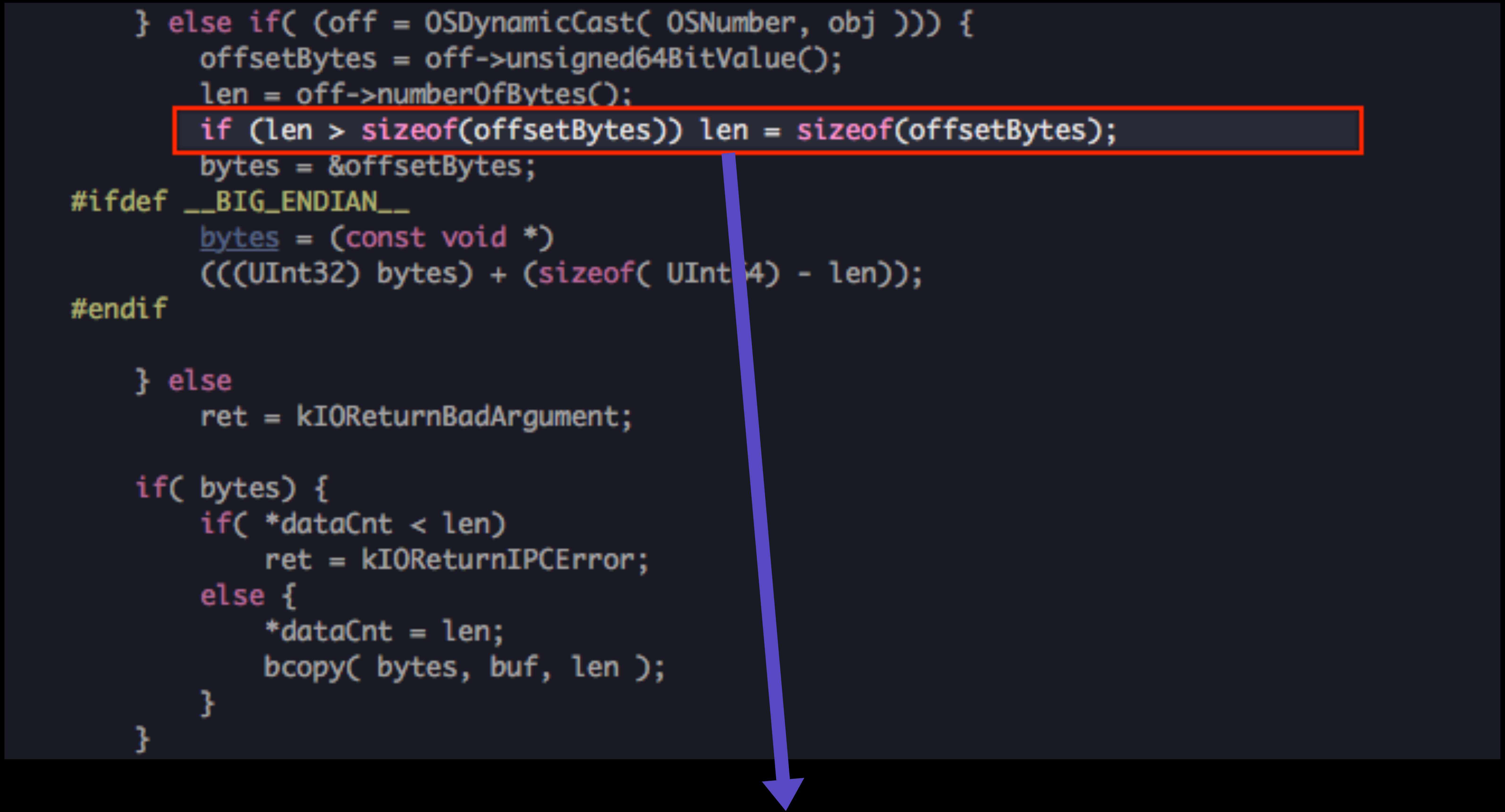


```
bool OSNumber::init(unsigned long long inValue, unsigned int newNumberOfBits)
{
    if (!super::init())
        return false;
    if (newNumberOfBits > 64)
        return false;

    size = newNumberOfBits;
    value = (inValue & sizeMask);

    return true;
}
```

During OSNumber initialization,
newNumberOfBits cannot exceed 64

OSNumber bug 2: additional fix

- Add check in is_io_registry_entry_get_property_bytes, dual protection!

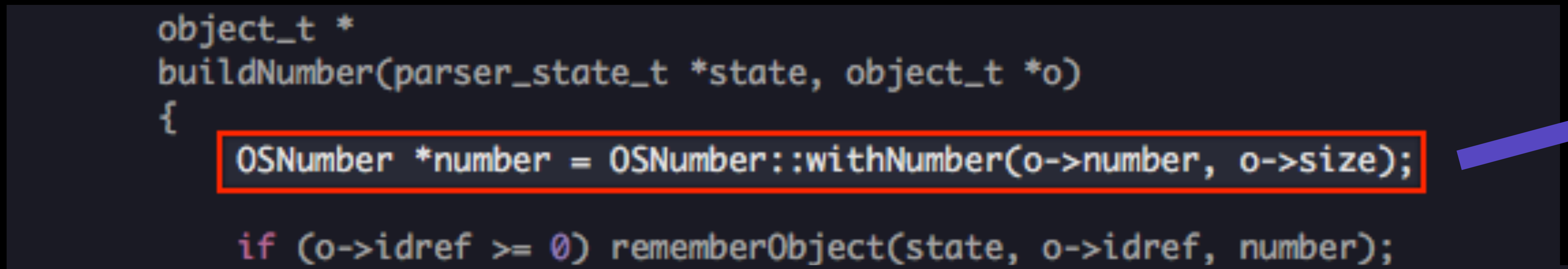


```
    } else if( (off = OSDynamicCast( OSNumber, obj ))) {
        offsetBytes = off->unsigned64BitValue();
        len = off->numberOfBytes();
        if (len > sizeof(offsetBytes)) len = sizeof(offsetBytes);
        bytes = &offsetBytes;
#ifdef __BIG_ENDIAN__
        bytes = (const void *)
        (((UInt32) bytes) + (sizeof( UInt64) - len));
#endif

    } else
        ret = kIOReturnBadArgument;

    if( bytes) {
        if( *dataCnt < len)
            ret = kIOReturnIPCError;
        else {
            *dataCnt = len;
            bcopy( bytes, buf, len );
        }
    }
```

len cannot exceed 8 bytes

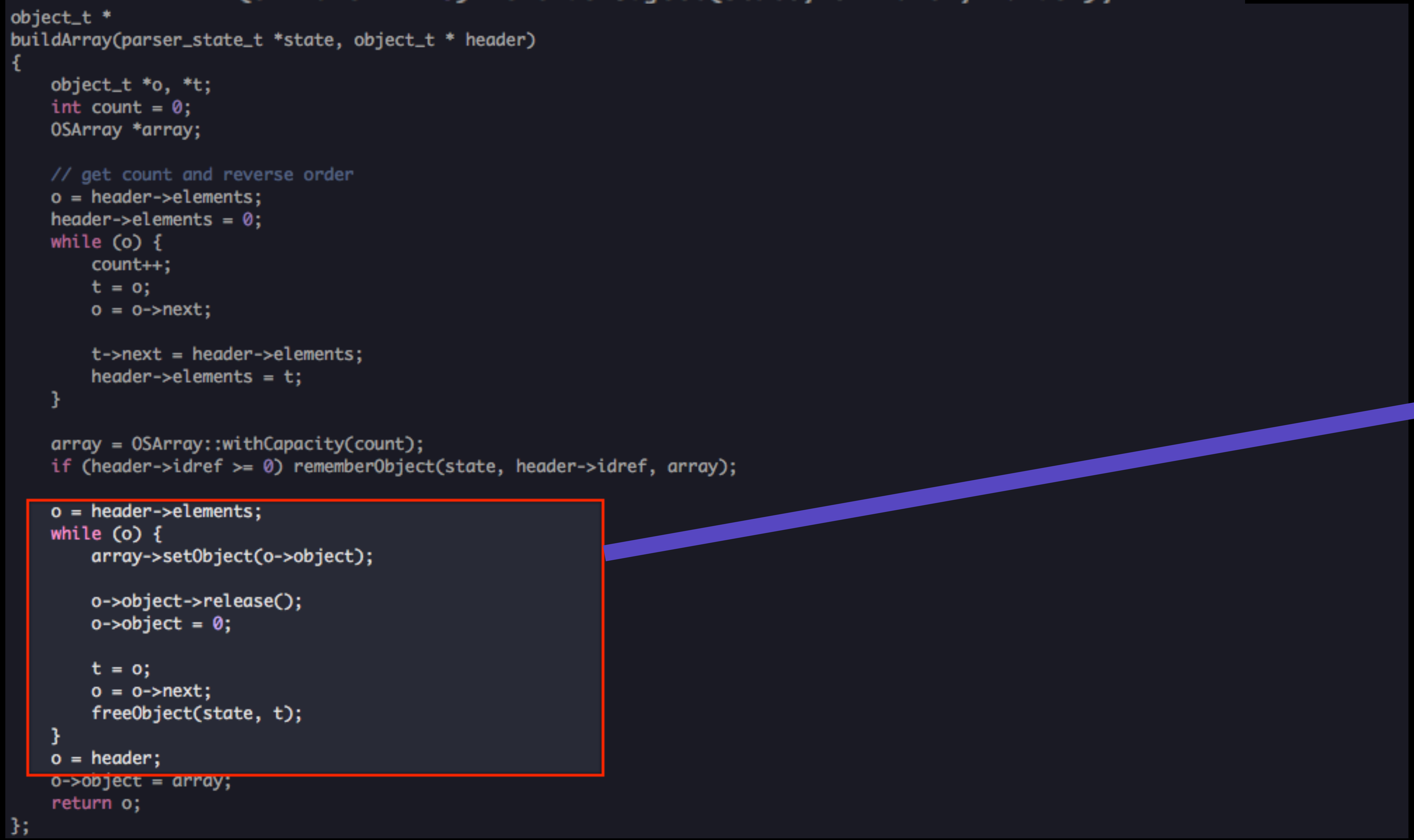
OSNumber bugs: all sorted?

- In XML mode, if size > 64, panic will occur
- Null pointer dereference



```
object_t *
buildNumber(parser_state_t *state, object_t *o)
{
    OSNumber *number = OSNumber::withNumber(o->number, o->size);

    if (o->idref >= 0) rememberObject(state, o->idref, number);
```

After the leak was fixed, OSNumber::withNumber returns Null for invalid len

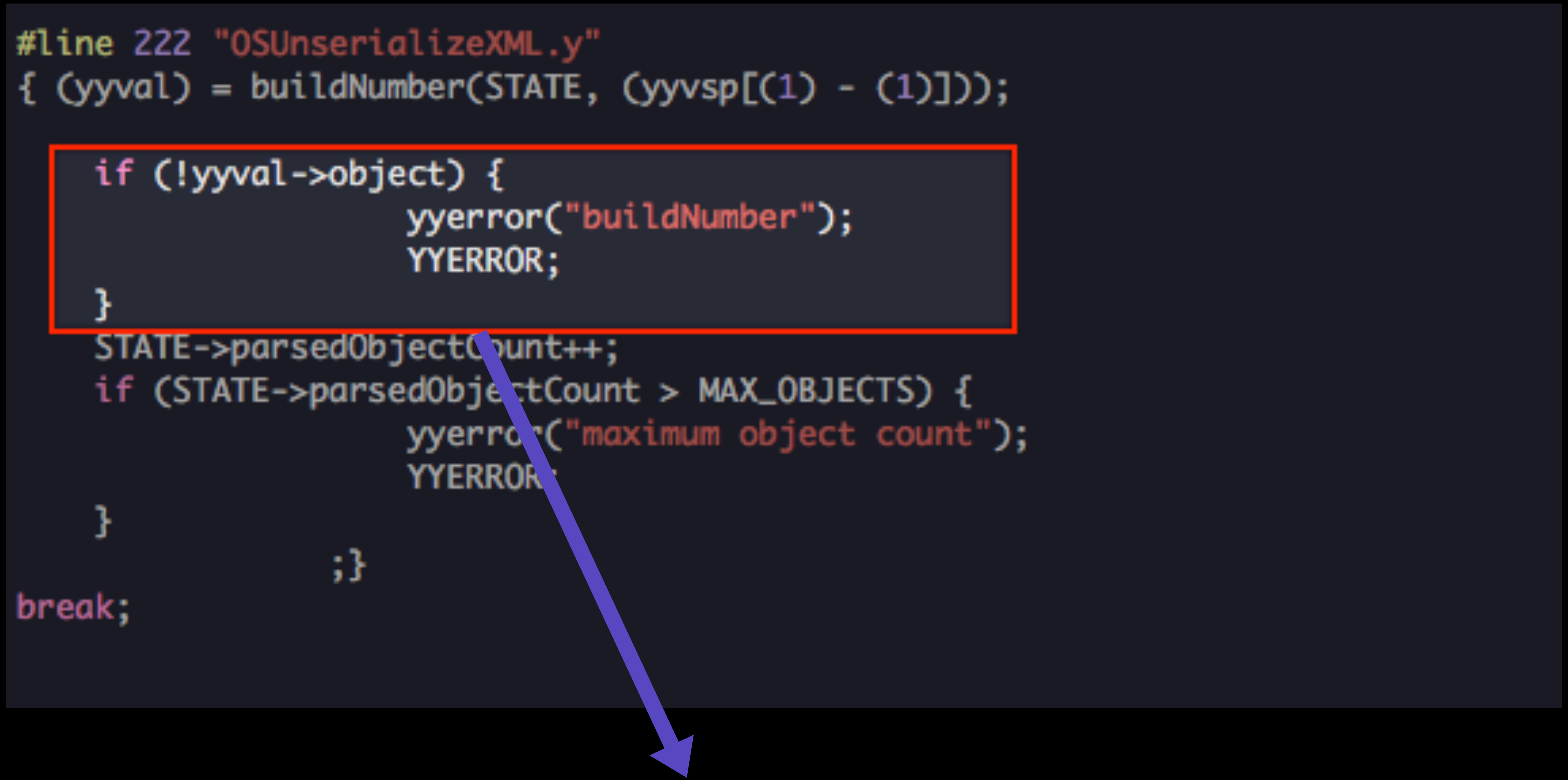


```
object_t *
buildArray(parser_state_t *state, object_t * header)
{
    object_t *o, *t;
    int count = 0;
    OSArray *array;

    // get count and reverse order
    o = header->elements;
    header->elements = 0;
    while (o) {
        count++;
        t = o;
        o = o->next;

        t->next = header->elements;
        header->elements = t;
    }

    array = OSArray::withCapacity(count);
    if (header->idref >= 0) rememberObject(state, header->idref, array);

    o = header->elements;
    while (o) {
        array->setObject(o->object);

        o->object->release();
        o->object = 0;

        t = o;
        o = o->next;
        freeObject(state, t);
    }
    o = header;
    o->object = array;
    return o;
};
```

In following array initialization, its element object is traversed and object->release is called, causing null pointer dereference

Final fix of OSNumber problem

- Thoroughly fixed in iOS 10.2



```
#line 222 "OSUnserializeXML.y"
{ (yyval) = buildNumber(STATE, (yyvsp[(1) - (1)]));

    if (!yyval->object) {
                    yyerror("buildNumber");
                    YYERROR;
    }
    STATE->parsedObjectCount++;
    if (STATE->parsedObjectCount > MAX_OBJECTS) {
                    yyerror("maximum object count");
                    YYERROR;
    }
                ;}
break;
```

Check if OSNumber is created successfully

OOL Race Condition issue

- Discovered by Qidan He of Keen Lab

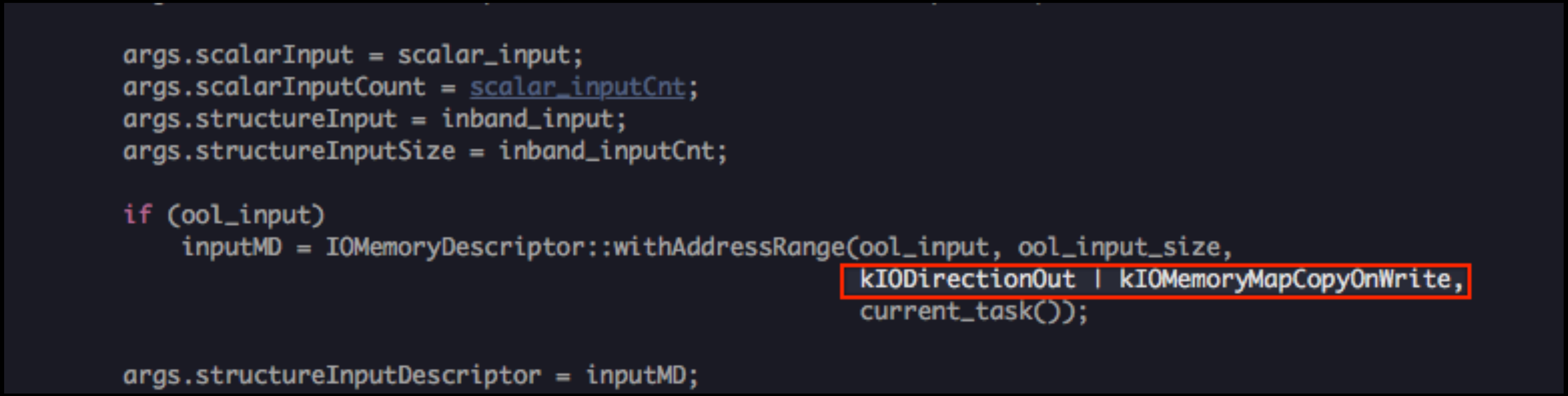
- Several drivers have the issue
  - CVE-2016-7624
  - CVE-2016-7625
  - CVE-2016-7714
  - CVE-2016-7620

- Apple found 20+ bugs caused by this mechnism

- When inputStruct length exceed 4096 in IOKit API IOConnectCallMethod will map the user mode buffer into kernel as the user input data:
  - Both userland and kernelland virtual memory share the same physical memory
  - Changing the content of the userlane buffer will change the kernel buffer content immediatel
  - Causing race condition problems

OOL Race Condition issue: the fix

- Fixed in iOS 10.2
- For all user supplied OOL buffer, map the kernel memory via Copy-On-Write



```
    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;
```

Part III: Exploitation

Object creation number limitation

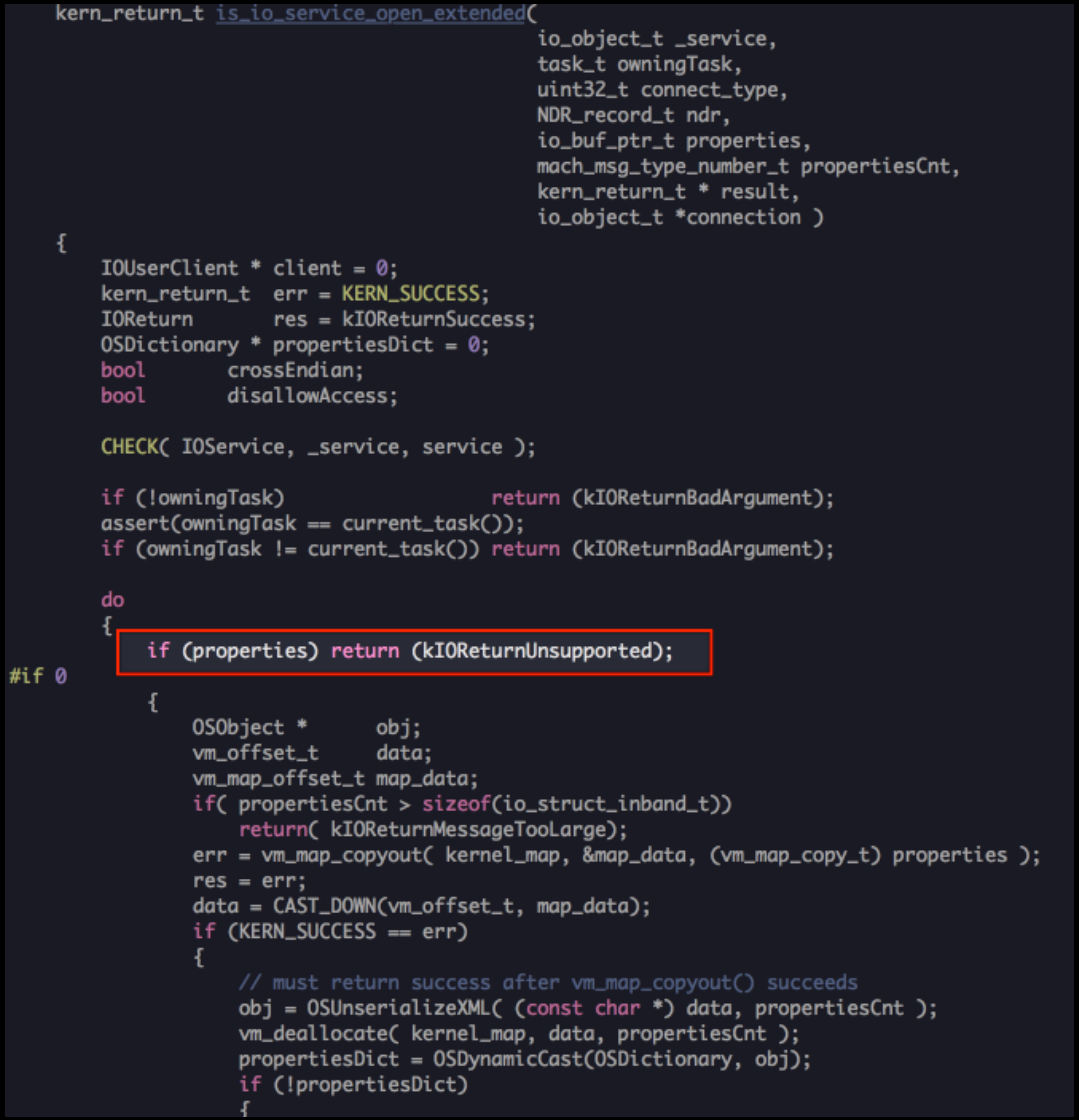
- Within sandbox several kernel objects can be created
  - With various size, in various kalloc zone
  - Perfect for heap fengshui

- iOS 10 limits quite some kernel objects

- E.g IOAccelResource2

Simplify some "dangerous" interface

- Famous API is_io_service_open_extended
  - Accepts serialized user data and call OSUnserializeXML, pefect for heap fengshui
- Simplified in iOS 10.2



```
kern_return_t is_io_service_open_extended(
                                           io_object_t _service,
                                           task_t owningTask,
                                           uint32_t connect_type,
                                           NDR_record_t ndr,
                                           io_buf_ptr_t properties,
                                           mach_msg_type_number_t propertiesCnt,
                                           kern_return_t * result,
                                           io_object_t *connection )
{
    IOUserClient * client = 0;
    kern_return_t  err = KERN_SUCCESS;
    IOReturn       res = kIOReturnSuccess;
    OSDictionary * propertiesDict = 0;
    bool        crossEndian;
    bool        disallowAccess;

    CHECK( IOService, _service, service );

    if (!owningTask)                 return (kIOReturnBadArgument);
    assert(owningTask == current_task());
    if (owningTask != current_task()) return (kIOReturnBadArgument);

    do
    {
        if (properties) return (kIOReturnUnsupported);
#if 0
        {
            OSObject *      obj;
            vm_offset_t     data;
            vm_map_offset_t map_data;
            if( propertiesCnt > sizeof(io_struct_inband_t))
                return( kIOReturnMessageTooLarge);
            err = vm_map_copyout( kernel_map, &map_data, (vm_map_copy_t) properties );
            res = err;
            data = CAST_DOWN(vm_offset_t, map_data);
            if (KERN_SUCCESS == err)
            {
                // must return success after vm_map_copyout() succeeds
                obj = OSUnserializeXML( (const char *) data, propertiesCnt );
                vm_deallocate( kernel_map, data, propertiesCnt );
                propertiesDict = OSDynamicCast(OSDictionary, obj);
                if (!propertiesDict)
                {
```

Enhanced KPP/AMCC

- From iOS 10.0 beta 2 got table is protected by KPP/AMCC

- The approach in Pangu 9.3.3 to modify got table was prohibited
  - PE_i_can_has_debugger

- Luca iOS 10.1.1 AMCC bypass approach was fixed
  - Can refer to Luca's talk: A Look at Modern iOS Exploit Mitigation Techniques

Neutering task_for_pid 0

- Obtaining kernel task port has become a standard for Jailbreaks

- Ian Beer mach_portal uses a very neat way to get tfp0

- iOS 10.3 limits the use of tpf0
  - Prohibit any usermode process to read/write kernel memory using tfp0
  - Ian Beer's mach_portal approach is mitigated

- iOS 11 extended the limit to the use of all task ports for app processes
  - Ian Beer's userland port hijack approach is mitigated

SMAP on 64bit platform(iPhone 7 only)

- Early in iOS 6, userland and kernelland address space are isolated
  - Accessing userland memory in kernelland is prohibited

- But on ARM64, only SMEP is present
  - Disallow executing userland code in kernelland
  - Kernelland can still access userland memory

- Provide convenience on ARM64 kernel exploitation
  - Leaking kernel heap address is not necessary
  - E.g Both Pangu 9.3.3 jailbreak and Yalu102 attempt to access userland memory in kernelland

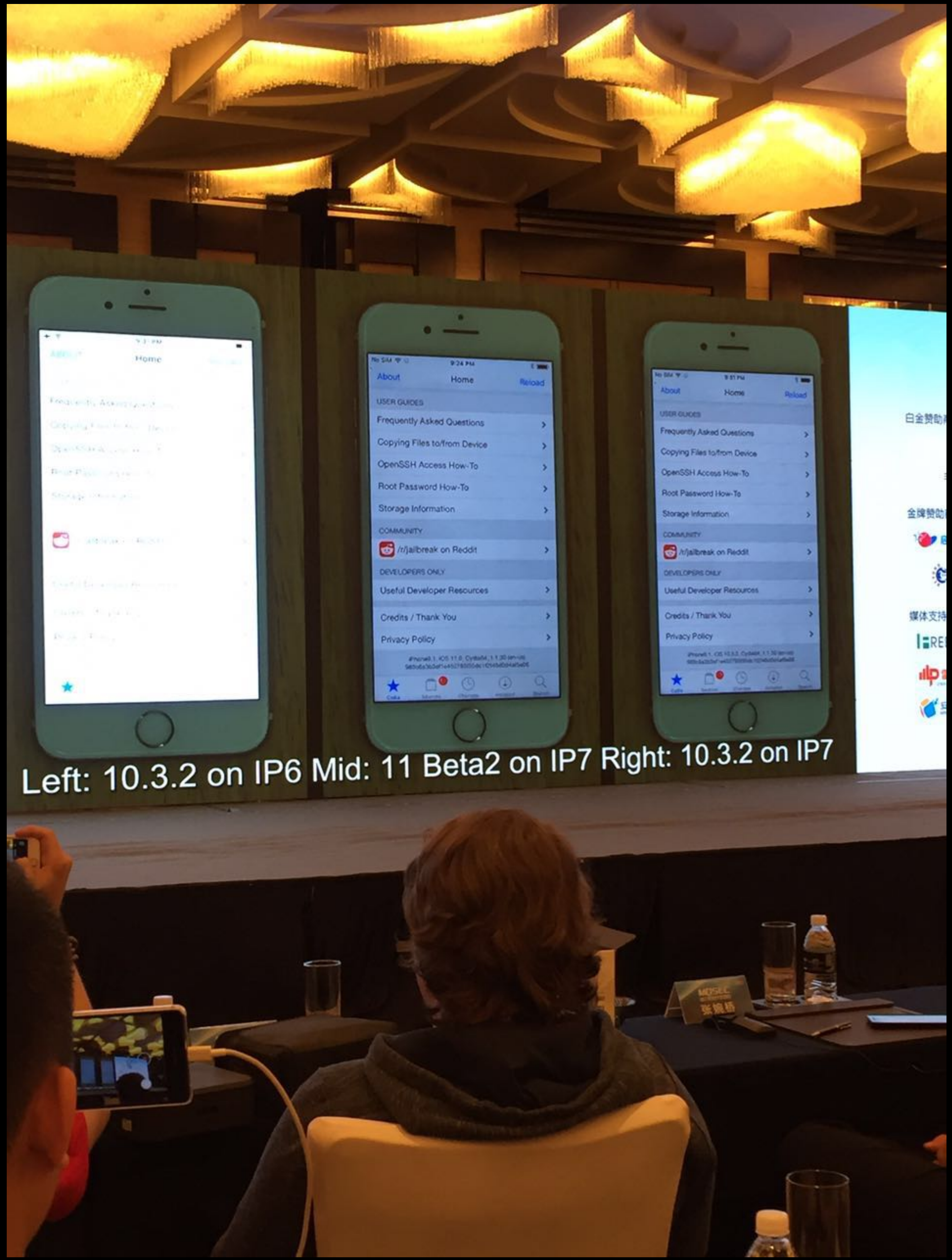
- iPhone 7 prohibits userland memory access in kernelland
  - Higher requirement on kernel info leak bug

Part IV: Summary

Summary

- This year in iOS 10, Apple enhanced iOS kernel security a lot
- Bus within container sandbox are almost extinct
  - future jailbreaks need to chain exploits on sandbox bypass + out-of-sandbox kernel bugs
- For some typical bugs, Apple tend to fix via mechanism instead of bug itself, to eliminate the whole set of problems
- Apple actively mitigates some common exploit techniques, making kernel exploitation harder

One more thing…


Left: 10.3.2 on IP6 Mid: 11 Beta2 on IP7 Right: 10.3.2 on IP7

Thank you!