Jailbreaking Apple Watch

Max Bazaliy

whoami

- Security researcher at Lookout
- Lead researcher on Pegasus exploit chain
- Focused on advanced exploitation techniques
- Fried Apple team co-founder
- iOS/tvOS/WatchOS jailbreak author

What is Apple Watch ?

- Released in 2015
- Apple S1/S2 processor
- ARMv7k 32 bit architecture
- Taptic engine
- 512 MB RAM
- WatchOS

Why to jailbreak a watch ?

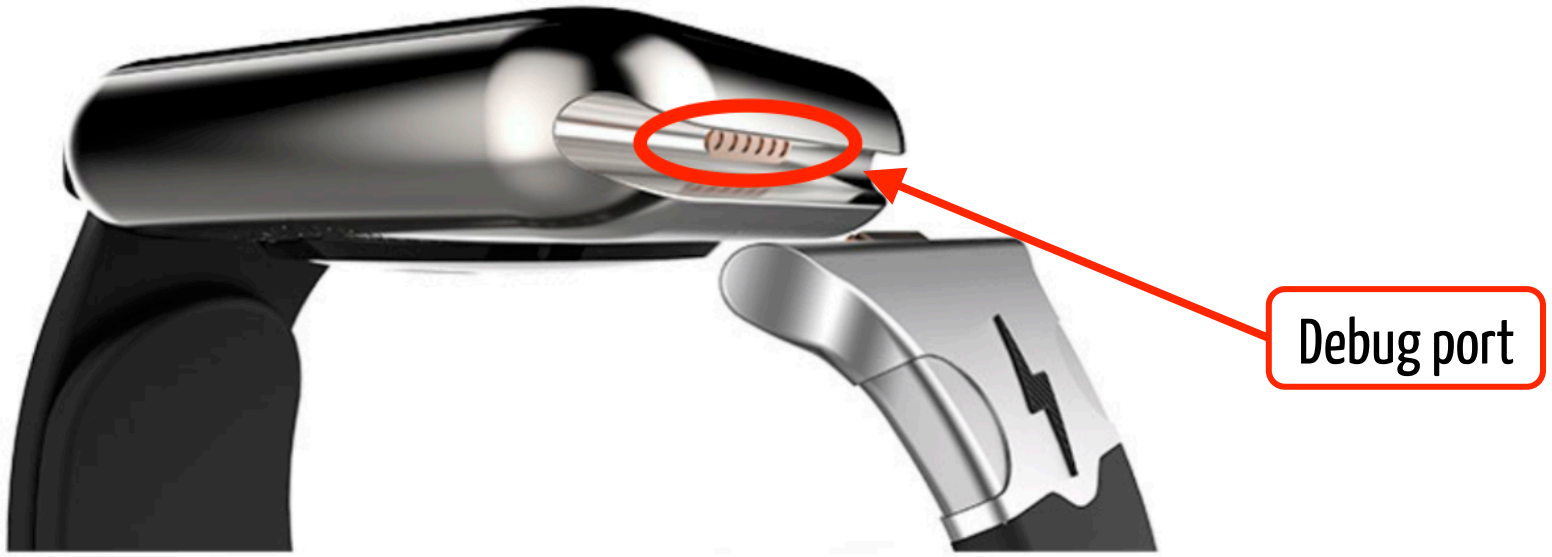
- Access to file system
- Run tools like radare or frida on a watch
- iPhone attack vector

Apple Watch security

- Secure boot chain
- Mandatory Code Signing
- Sandbox
- Exploit Mitigations
- Secure Enclave Processor (2-nd gen only)
- Data Protection

Possible attack vectors

- Malformed USB descriptor over debug port



Possible attack vectors

- Malformed email, message, photo, etc
  - Still limited by sandbox
- Application extension based
  - More freedom on bug choice

Jailbreak step by step

- Leak kernel base
- Dump whole kernel
- Find gadgets and setup primitives
- Disable security restrictions
- Run ssh client on a watch

Bugs of interest

- WatchOS 2.x
  - CVE-2016-4656 - osunserialize bug
  - CVE-2016-4669 - mach_port register bug
- WatchOS 3.1.3
  - CVE-2016-7644 - set_dp_control_port bug
  - CVE-2017-2370 - voucher extract recipe bug

Leaking kernel base

- CVE-2016-4655 and CVE-2016-4680
- Object constructor missing bounds checking
- OSNumber object with high number of bits
- Object length used to copy value from stack
- Kernel stack memory leaked
- Can be triggered from an app's sandbox

```
OSObject * OSUnserializeBinary(const char *buffer, size_t bufferSize,
                                                    OSString **errorString) {
uint32_t key, len, wordLen;
len = (key & kOSSerializeDataMask);
...
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;
```

No number length check

```
bool OSNumber::init(unsigned long long inValue,
                                        unsigned int newNumberOfBits) {
    if (!super::init())
        return false;
    size = newNumberOfBits;
    value = (inValue & sizeMask);
    return true;
}

unsigned int OSNumber::numberOfBytes() const {
        return (size + 7) / 8;
}
```

No number length check

Return value is under control

```
kern_return_t is_io_registry_entry_get_property_bytes( io_object_t registry_entry,
io_name_t property_name, io_struct_inband_t buf, … ) {
...
UInt64 offsetBytes;          // stack based buffer
...
} else if( (off = OSDynamicCast( OSNumber, obj ))) {
            offsetBytes = off->unsigned64BitValue();
            len = off->numberOfBytes();
            bytes = &offsetBytes;
...
if (bytes) {
            if( *dataCnt < len)
                ret = kIOReturnIPCError;
            else {
           *dataCnt = len;
            bcopy( bytes, buf, len );        // copy from stack based buffer
```

Points to stack based buffer

Will be returned to userland

We control this value

CVE-2016-4656 exploitation

- Kernel mode UAF in OSUnserializeBinary
- OSString object deallocated
- retain() called on deallocated object
- Fake object with fake vtable –> code exec

```
OSObject * OSUnserializeBinary(const char *buffer, size_t bufferSize, …) {

newCollect = isRef = false;
…
case kOSSerializeDictionary:
        o = newDict = OSDictionary::withCapacity(len);
        newCollect = (len != 0);
        break;
…
        if (!isRef)
        {
            setAtIndex(objs, objsIdx, o);
            if (!ok) break;
            objsIdx++;
        }
```

Save object to objs array

```
if (dict) {
        if (sym)
         …
        else {
            sym = OSDynamicCast(OSSymbol, o);
            if (!sym && (str = OSDynamicCast(OSString, o))) {
                sym = (OSSymbol *) OSSymbol::withString(str);
                o->release();
                o = 0;
            }
            ok = (sym != 0);
        }
    }
```

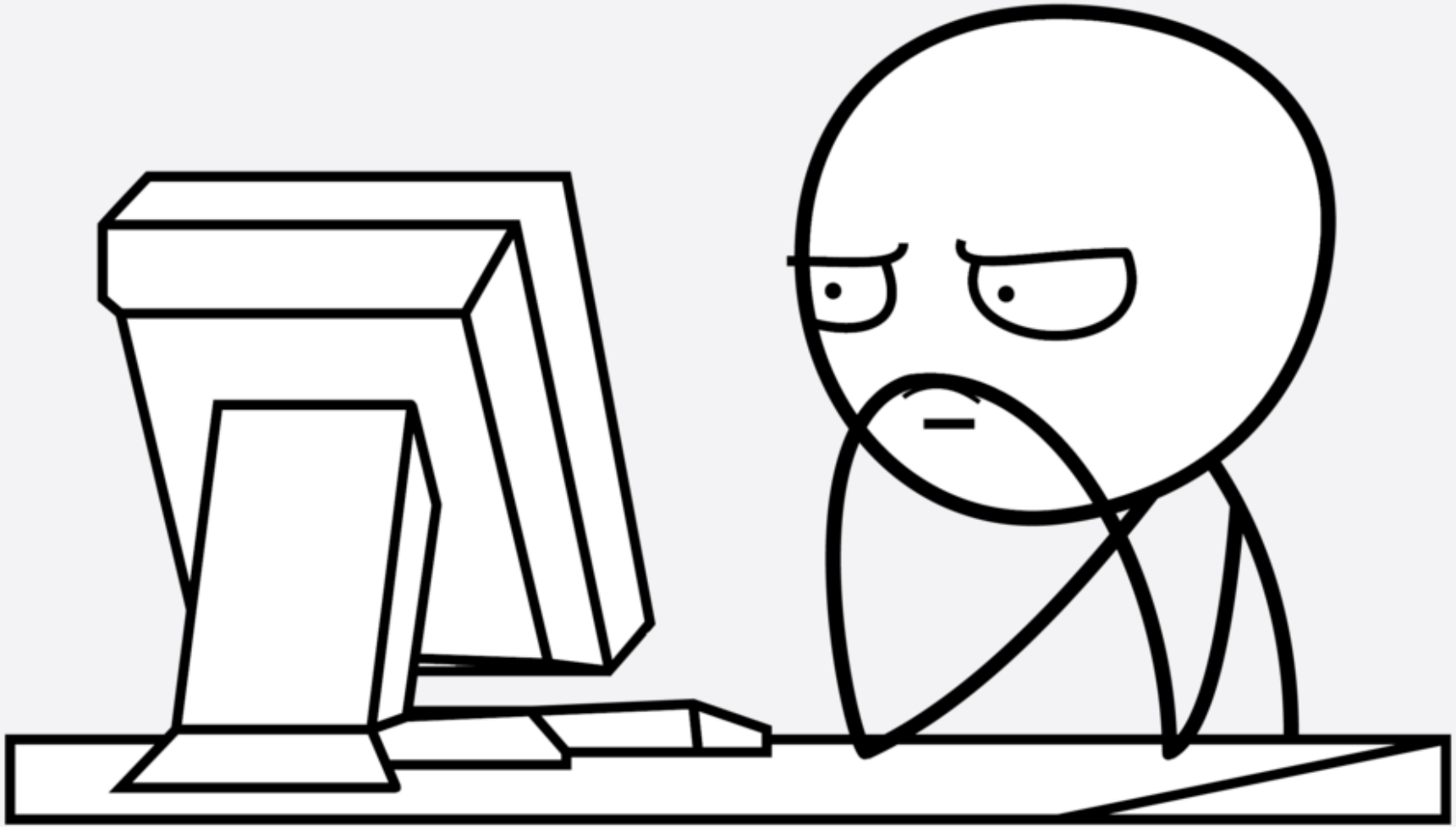
Object saved to objs array destroyed

Deallocated object retained

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

Dumping kernel

- Problem: No WatchOS kernel dumps
- No keys for WatchOS kernels
- Idea: read kernel as OSString chunks
- vtable offset required to fake OSString
- vtable stored in __DATA.__const in kernel



Getting vtable - __DATA.__const leak

- __DATA.__const address is in Mach-O header
- Kernel base + 0x224 == __DATA.__const
- Deref and branch to address via fake vtable

```
:HEADER:80001158 ; Sections
:HEADER:80001158                 section <"__nl_symbol_ptr", "__DATA", 0x80
:HEADER:80001158                          0, 0, 0, 0, 0>
:HEADER:8000119C                 section <"__mod_init_func", "__DATA", 0x80
:HEADER:8000119C                          0, 0, 9, 0, 0>
:HEADER:800011E0                 section <"__mod_term_func", "__DATA", 0x80
:HEADER:800011E0                          0, 0, 0xA, 0, 0>
:HEADER:80001224                 section <"__const", "__DATA", 0x803E7000,
:HEADER:80001224                          0, 0, 0>
:HEADER:80001268                 section <"__data", "__DATA", 0x803F7000, 0
```

Getting vtable - known offset

- Get vtable offset from similar XNU build
- Known delta from __DATA.__const start
- Tune address with +/- delta

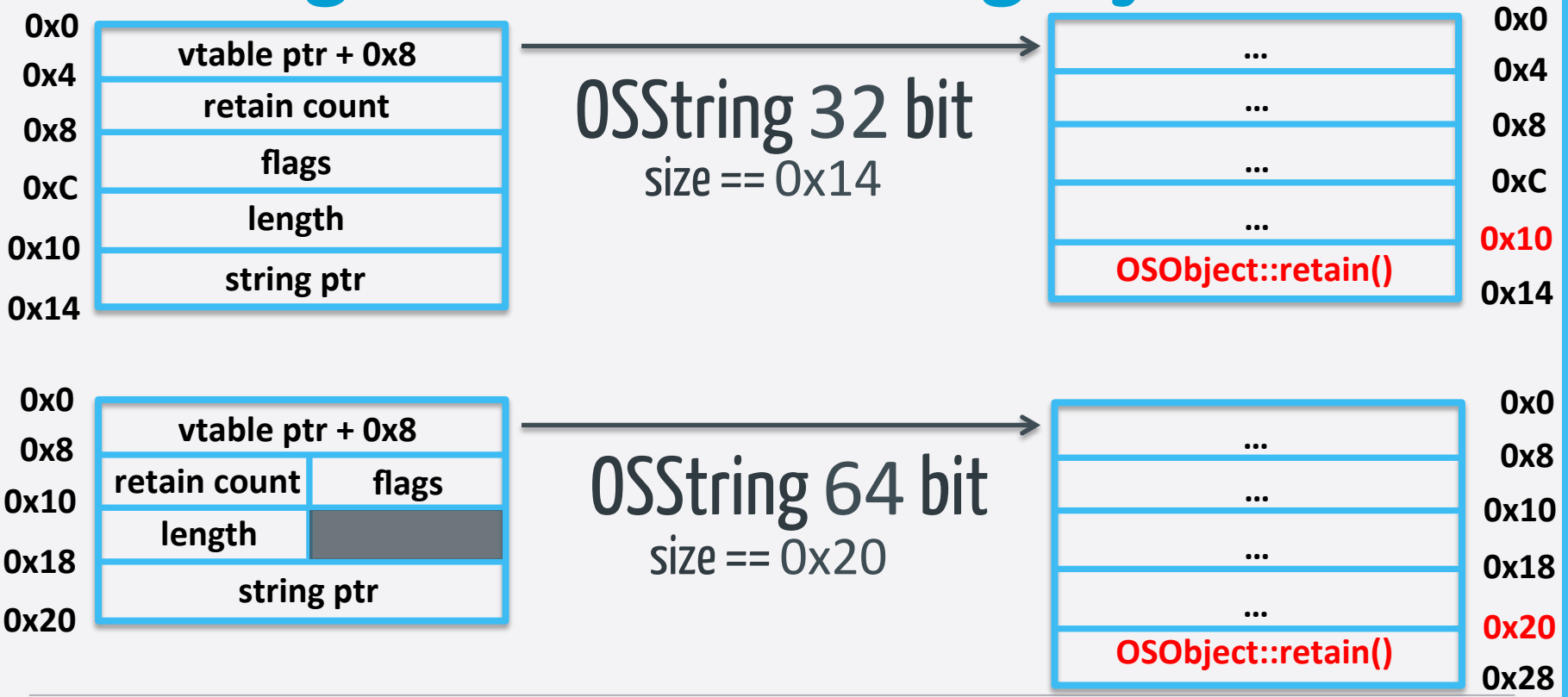
```
__DATA:__const:803ECE8C                         EXPORT __ZTV8OSString
__DATA:__const:803ECE8C ; `vtable for'OSString
__DATA:__const:803ECE8C __ZTV8OSString  DCB     0
__DATA:__const:803ECE8C
__DATA:__const:803ECE8D                 DCB     0
__DATA:__const:803ECE8E                 DCB     0
__DATA:__const:803ECE8F                 DCB     0
__DATA:__const:803ECE90                 DCB     0
```

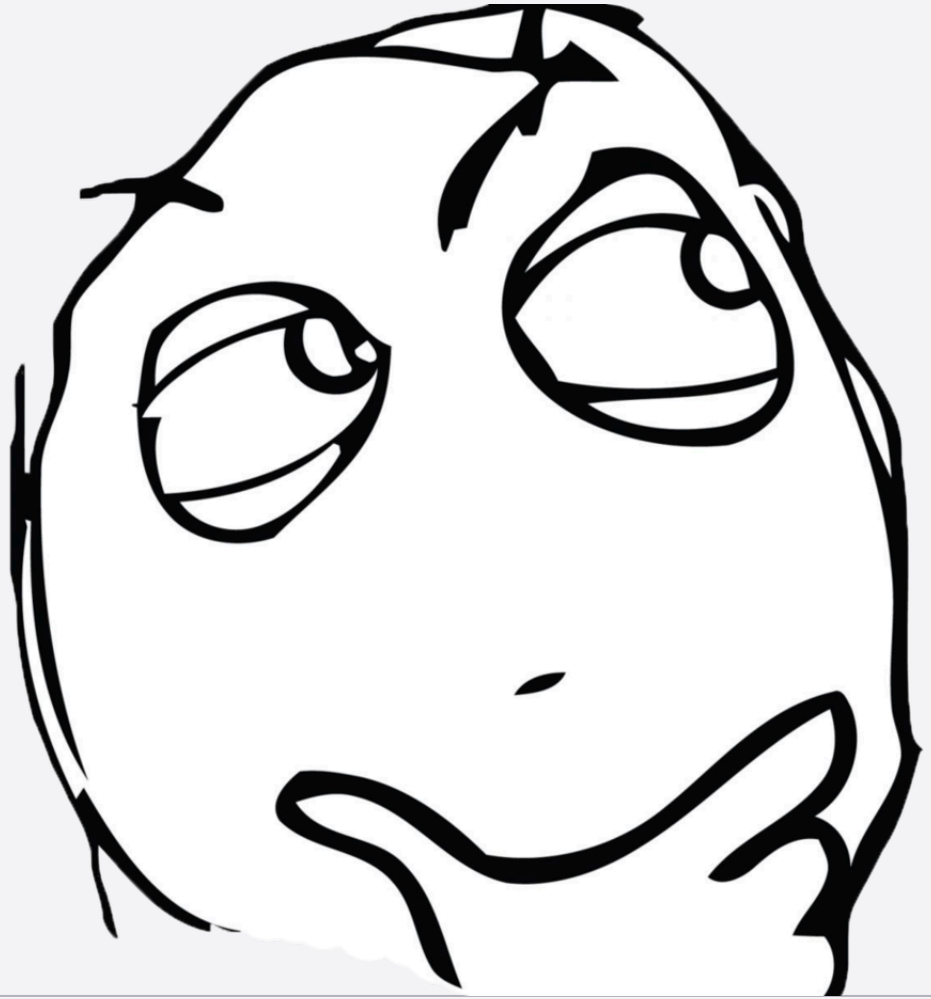
Getting vtable - known offset

- Get vtable offset from similar XNU build
- Known delta from __DATA:__const start
- Tune addresses with +/- delta

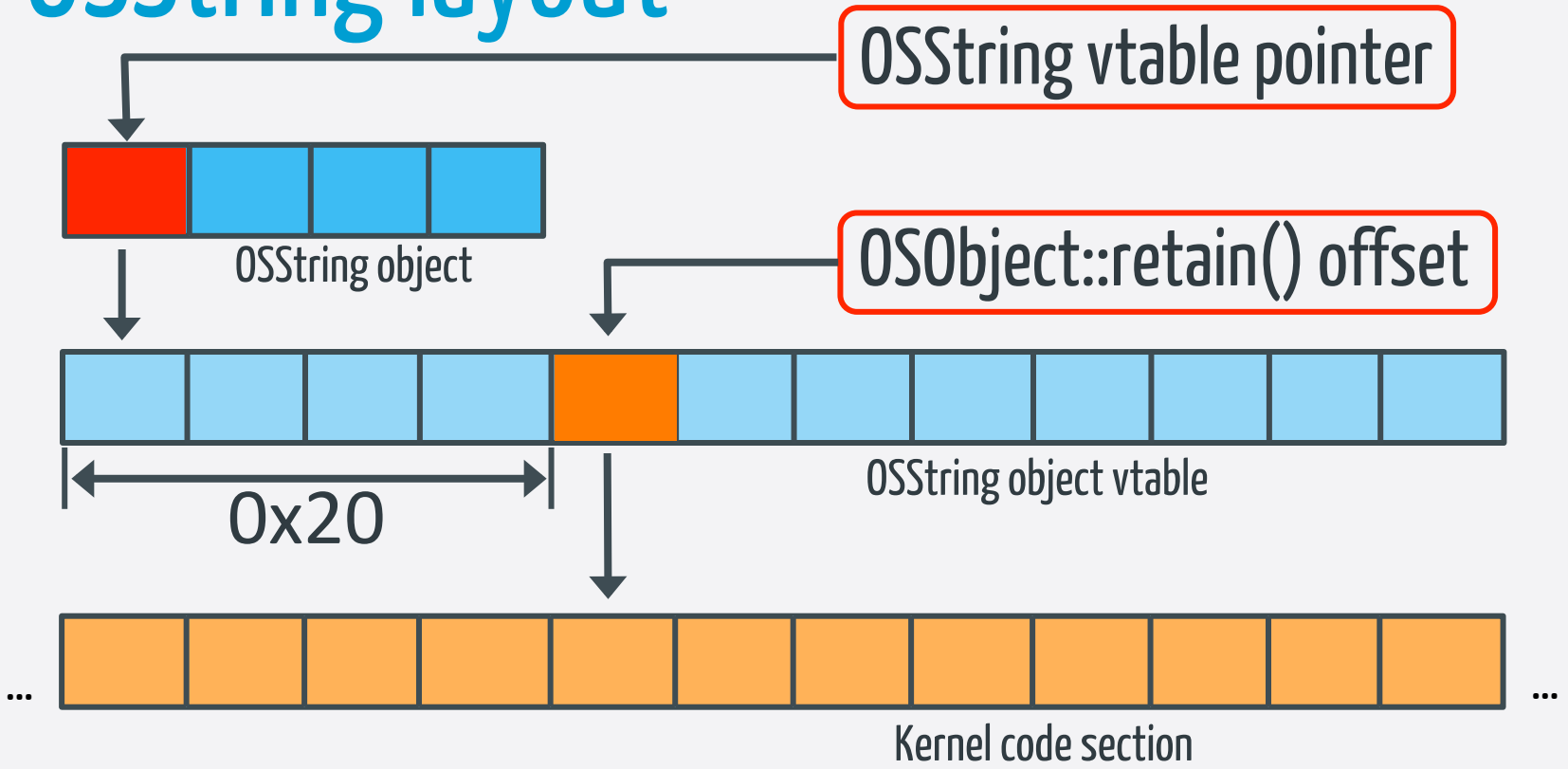

```
__DATA:__const:803ECE8C                 EXPORT __ZTV8OSString
__DATA:__const:803ECE8C ; vtable for'OSString
__DATA:__const:803ECE8C __ZTV8OSString  DCD 0
__DATA:__const:803ECE8C
__DATA:__const:803ECE8D                 DCB 0
__DATA:__const:803ECE8E                 DCB 0
__DATA:__const:803ECE8F                 DCB 0
__DATA:__const:803ECE90                 DCB 0
```

Getting vtable – OSString layout





OSString layout



Getting vtable – next free node trick

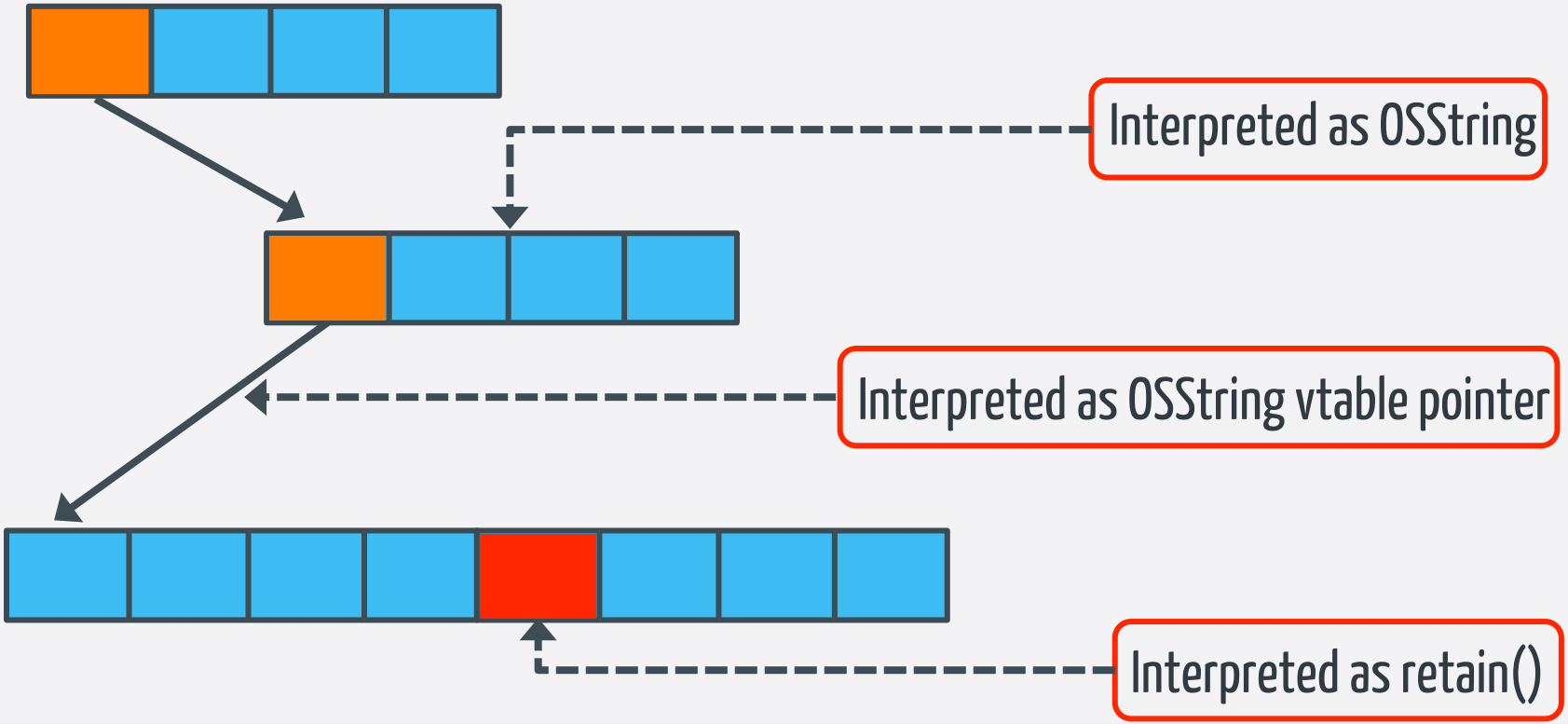
- vtable ptr is first 4/8 bytes of a on object
- What if object is not reallocated ?
- Memory marked as free
- New node pointing to next node in freelist

Heap zone freelist

Getting vtable – next free node trick

- OSString memory marked as free
- Now it's a node pointing to next node
- Next node ptr will be interpreted as vtable
- Call to retain() will branch out of node bounds
- What if OSString **size** == retain() **offset** ?
- We can branch out to the start of next node

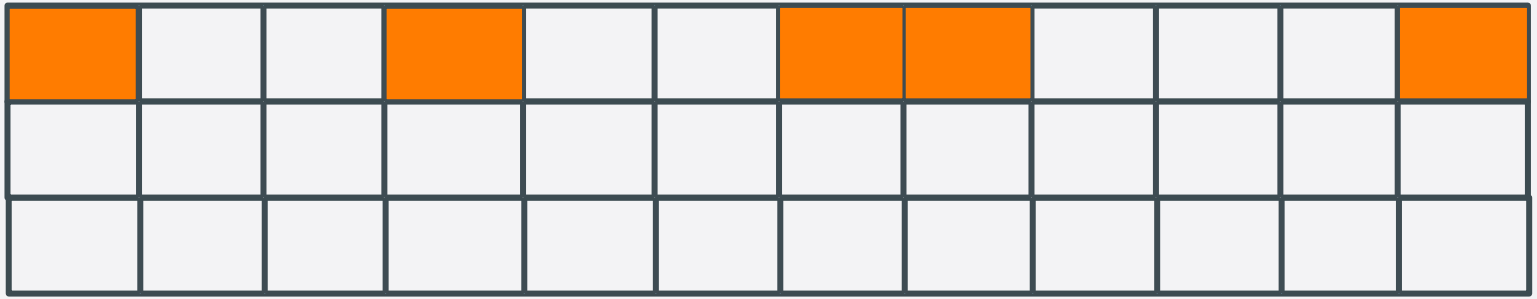
Next node ptr as a vtable ptr



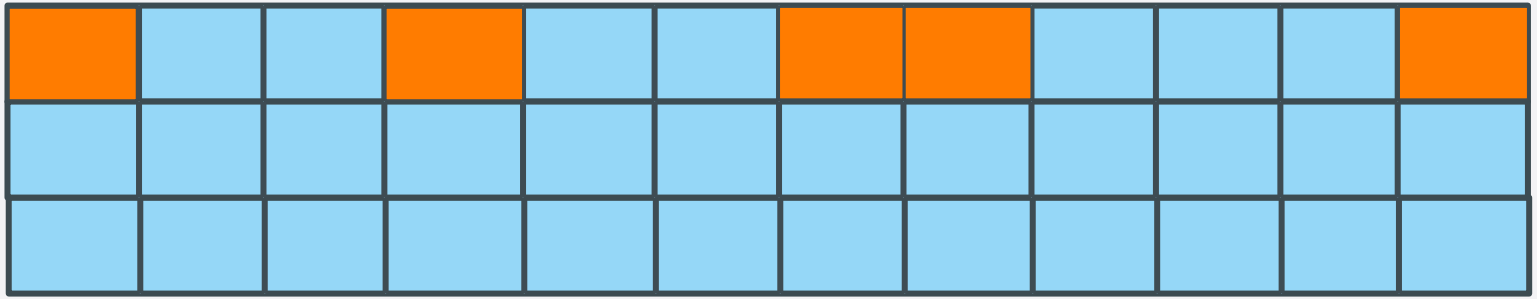
Getting vtable – next free node trick

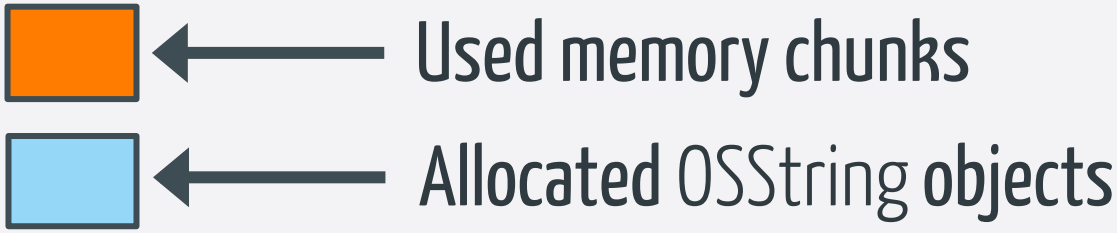
- Heap spray OSString objects
- Free few OSString's
- Next free chunk pointer dereferenced as vtable
- Free chunk is surrounded by OSStrings
- retain() -> OOB branch to next OSString

Heap spray and OOB branch to vtable

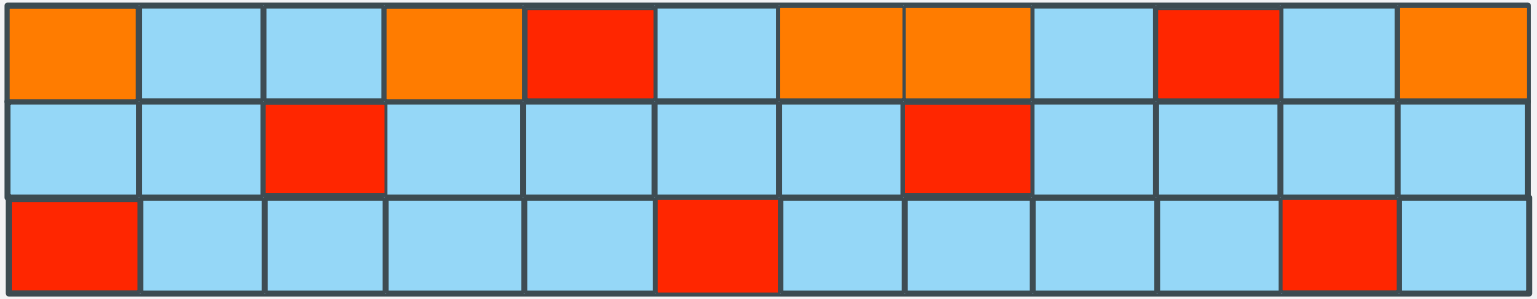


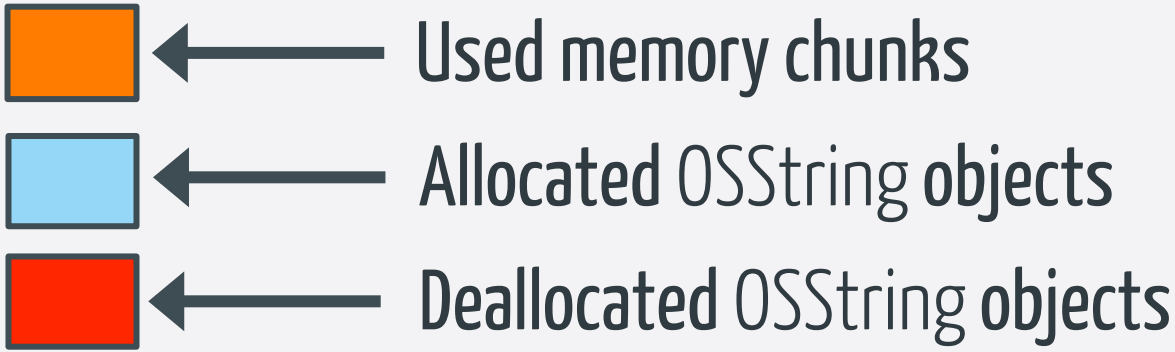
Heap spray and OOB branch to vtable



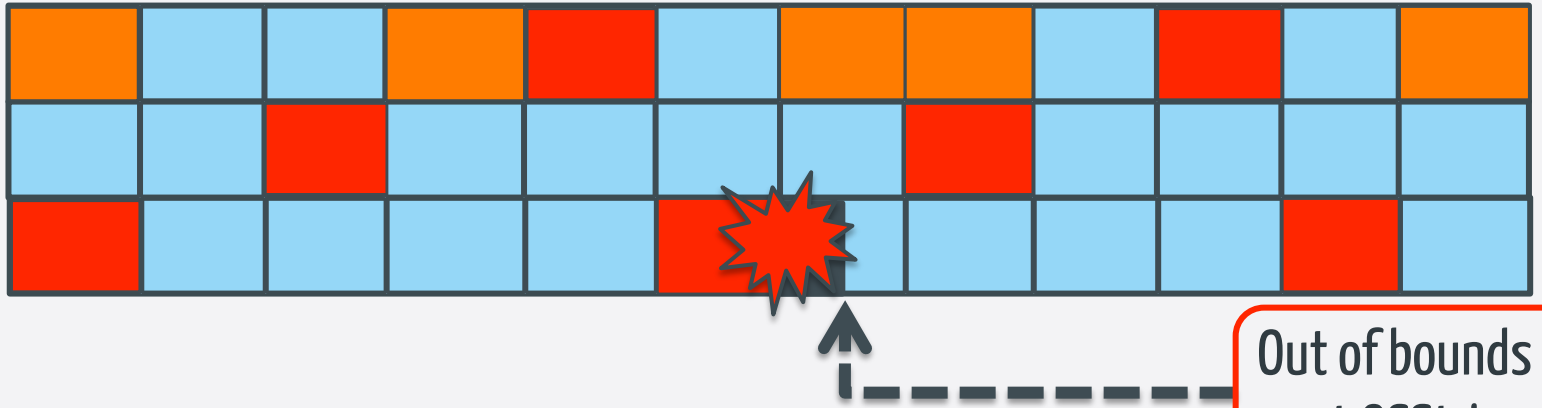


Heap spray and OOB branch to vtable





Heap spray and OOB branch to vtable



Out of bounds branch to next OSString vtable

Used memory chunks

Allocated OSString objects

Deallocated OSString objects

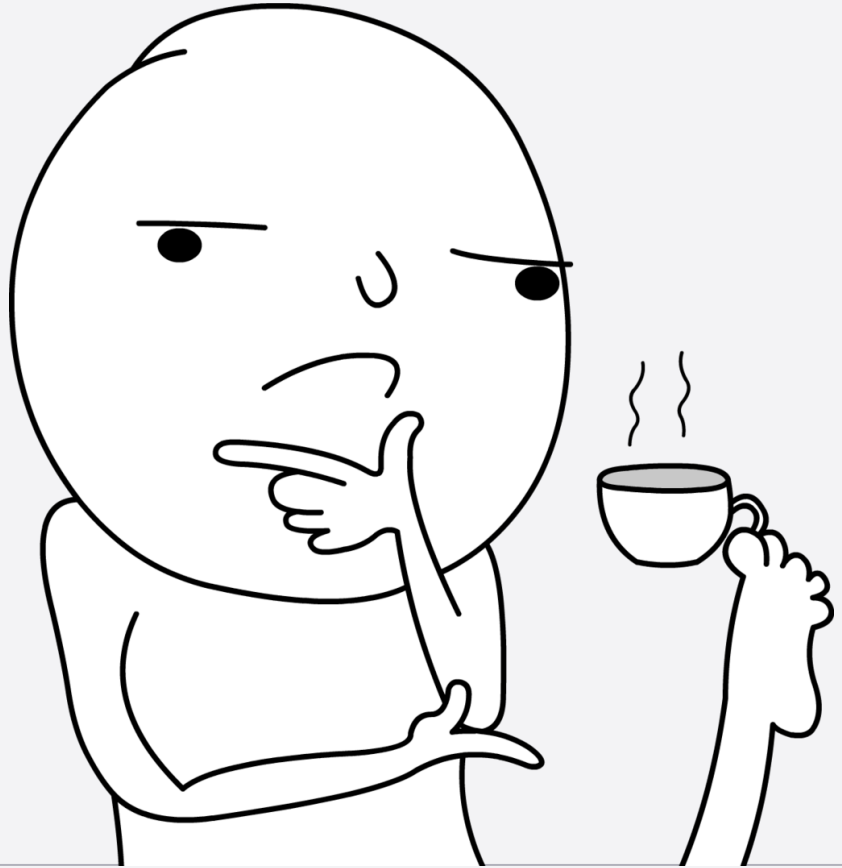
Getting vtable – next free node trick

- Heap spray OSString objects
- Make few OSDictionary with OSString
- Trigger OSDictionary allocation
- retain() dereference next free chunk pointer
- Free chunk is surrounded by OSStrings
- retain() -> OOB branch to next OSString node

Getting vtable – dump over panic

- OSString **vtable reference in** OSUnserializeBinary!
- OSUnserializeBinary **reference in** OSUnserializeXML

```
OSUnserializeBinary                     ; CODE XREF: OSUnserializeXML(char cons
                PUSH            {R4-R7,LR}
                ADD             R7, SP, #0xC
                PUSH.W          {R8,R10}
                MOV             R5, R0
                MOVS            R0, #0x14 ; this
                MOV             R8, R1
                BL              __ZN8OSObjectnwEm ; OSObject::operator new(ulo
                MOVW            R1, #:lower16:(__ZTV8OSString - 0x8031A9C0) ;
                MOV             R4, R0
                MOVT.W          R1, #:upper16:(__ZTV8OSString - 0x8031A9C0) ;
                MOV             R0, #(__ZN8OSString10gMetaClassE - 0x8031A9C2)
                ADD             R1, PC  ; `vtable for'OSString
```



Getting vtable – dump over panic

- Crash in OSUnserializeBinaryXML
- Copy panic log from a watch
- Get LR register value from panic
- We got OSUnserializeBinaryXML address

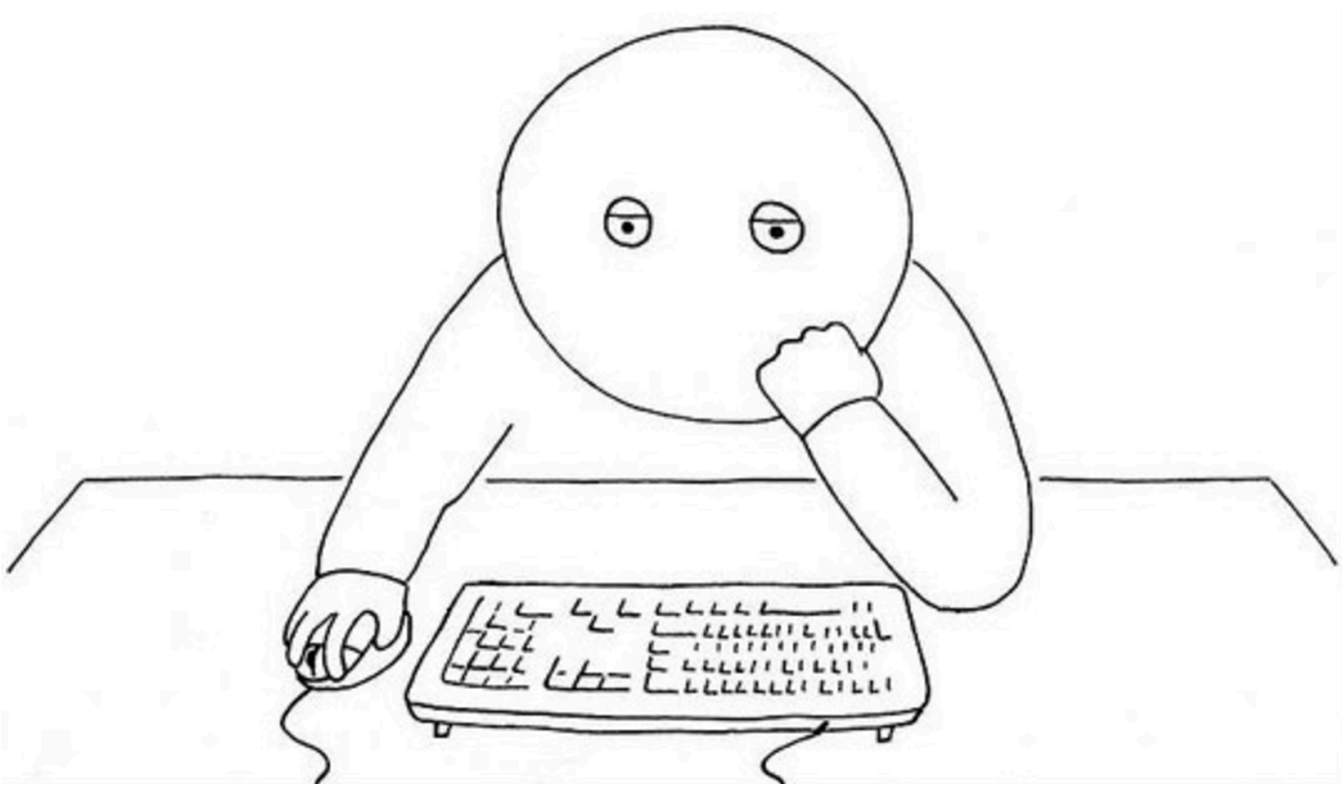
Dumping kernel by panic logs

- retain() offset in vtable is 0x10
- Use address to leak as vtable_addr - 0x10
- vtable will be interpreted and branch to address
- Kernel will crash, but save panic log
- Address content appear in panic registers state

Dumping kernel by 4 bytes

- Use address to leak as fake vtable address
- Watch will crash, wait until it restore
- ssh to a iPhone and run synchronization service
- Copy panic from Watch to iPhone and to Mac
- Parse panic, read 4 bytes and disassemble !
- Update address with 4 bytes delta and upload app
- Repeat

It's fun !



OSString vtable in kernel

```
MOV             R0, R10
STR             R2, [SP,#0x88+var_34]
BL              __Z19OSUnserializeBinaryPKcmPP8OSString ; OSUnserializeBinary
B               loc_8031533A
```

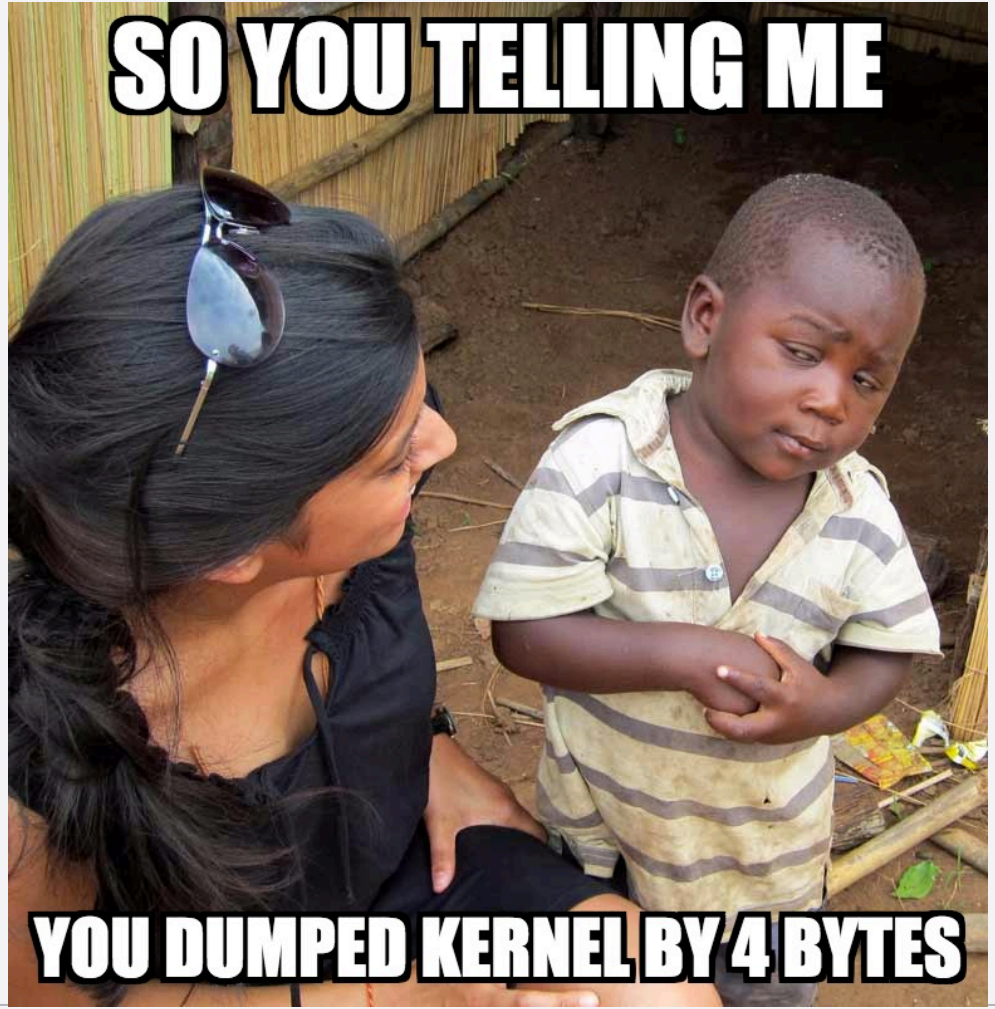
OSUnserializeBinary address

OSString vtable offset

```
MOVS            R0, #0x14
MOV             R8, R1
BL              __ZN8OSObjectnwEm ; OSObject::operator new(ulong)
MOVW            R1, #:lower16:(__ZTV8OSString - 0x80311340) ; `vtable for'OSString
MOV             R4, R0
MOVT.W          R1, #:upper16:(__ZTV8OSString - 0x80311340) ; `vtable for'OSString
MOV             R0, #(__ZN8OSString10gMetaClassE - 0x80311342) ; OSString::gMetaClass
ADD             R1, PC ; `vtable for'OSString ; `vtable for'OSString
```

Getting vtable – final steps

- Crash in OSUnserializeXML
- Dump 4 bytes, disassemble, read opcode
- Leak opcode until 'BL OSUnserializeBinary'
- Leak OSUnserializeBinary opcodes
- Finally leak OSString vtable offset


SO YOU TELLING ME
YOU DUMPED KERNEL BY 4 BYTES

Getting vtable – results

- 5 minutes for recover watch after crash
- 5 minutes to fetch panic from watch
- 2 minutes to copy to Mac and parse
- No way to automate a process
- It takes me just 2 weeks to dump a vtable

Next step – full kernel dump

- Now use fake OSString obj to read kernel
- Read data via IORegistryEntryGetProperty

- Leak kernel header, calculate kernel size
- Dump full kernel to userland by chunks

Next step – kernel symbolication

- Find and list all kexts
- Find sysent and resolve syscalls
- Find and resolve mach traps
- Resolve IOKit objects vtable

Next step – setting up primitives

- Scan kernel dump for gadgets
- Set up exec primitive
- Set up kernel read & write primitives

```
LDR     R1, [R2]
BX      LR
```

```
STR     R1, [R2]
BX      LR
```

Next step – kernel structs layout

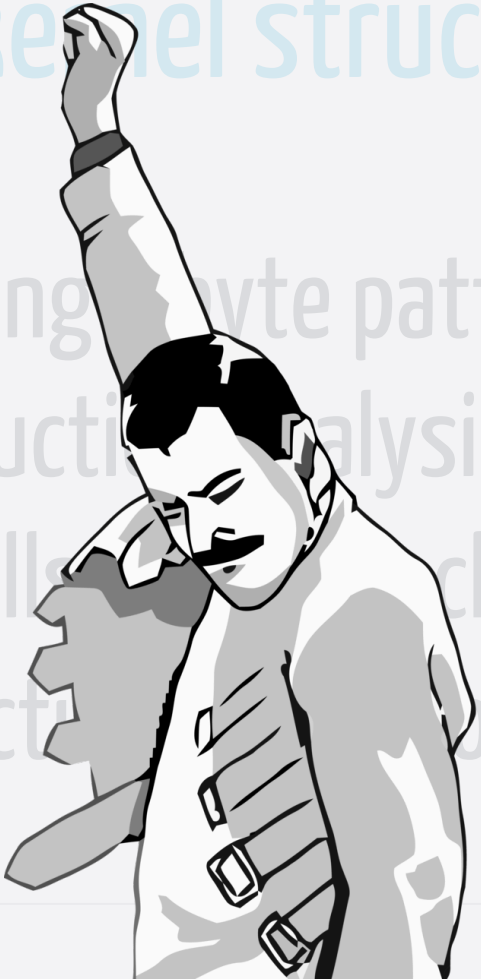
- Look for proc_* functions
- Restore proc structure layout
- Dump memory, check for known values

Next step – patchfinder

- memmem string \ byte pattern
- + xref + instruction analysis
- Resolve syscalls table, mach traps table
- Simple instruction emulation

Next step – kernel structs layout

- memmem string / byte pattern
- + xref + instruction analysis
- Resolve syscalls from switch traps table
- Simple instruction emulation



Getting root and sandbox bypass

- Patch setreuid (no KPP)

- patch ucred in proc structure in kernel
- patch sandbox label value in ucred

Getting kernel task

- Patch task_for_pid()
- Or save kernel sself in task bootstrap port
- Read it back via task_get_special_port()
- Restore original bootstrap port value

Disable codesign checks

- Patch _debug to 1
- patch _nl_symbol_ptr (got) entries
- Patch amfi variables
  - cs_enforcement_disable
  - allow_invalid_signatures

Remount rootfs

- Patch __mac_mount
- Change flags in rootfs vnode and mount RW

- Patch lwvm is_write_protected check
- Patch PE_i_can_has_debugger in lwvm

Spawning ssh client

- Compile dropbear for ARMv7k
- Compile basic tools package for ARMv7k
- Problem: More sandbox restrictions
- Remove WatchOS specific sandbox ops

ssh connection problem…

- WatchOS interfaces

```
"awdl0/ipv6" = "fe80::c837:8aff:fe60:90c2";
"lo0/ipv4”     = "127.0.0.1";
"lo0/ipv6"     = "fe80::1";
"utun0/ipv6" = "fe80::face:5e30:271e:3cd3";
```


ONE DOES NOT SIMPLY
RUN SSH CLIENT ON A WATCH

Watch <-> iPhone port forwarding

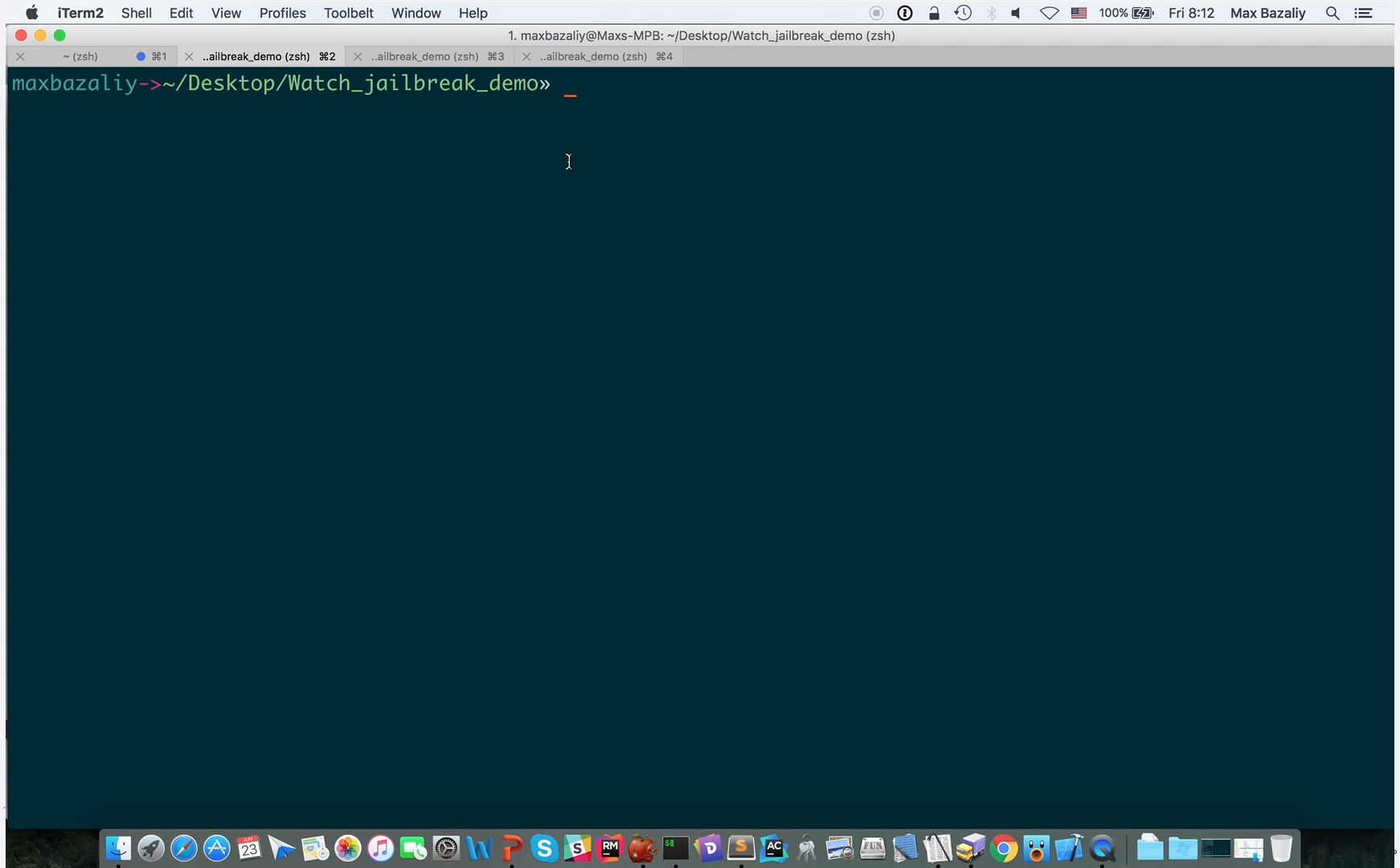
```
NSDictionary *comm = @{
        @"Command" :@"StartForwardingServicePort",
        @"ForwardedServiceName" :@"com.apple.syslog_relay",
        @"GizmoRemotePortNumber" :[NSNumber numberWithUnsignedShort: pt],
        @"IsServiceLowPriority" :@0,};

AMDServiceConnectionSendMessage(serviceConnection,
                                (__bridge CFPropertyListRef)(comm),
                                        kCFPropertyListXMLFormat_v1_0);

AMDServiceConnectionReceiveMessage(serviceConnection, &response,
                                        (CFPropertyListFormat*)&format);

NSNumber *iphone_port = response[@"CompanionProxyServicePort"];
```

Thanks to Luca Todesco

ssh connection over bluetooth

```
[!] Setting up bluetooth proxy on a watch
[+] Device connected, binding port 22 on watch to an iPhone port 50308
[+] Port binded, now use port 50308 on your iPhone device
[+] Setting up iproxy with local port 5444 and iPhone port 50308
[+] Done. Now ssh to local port 5444 to access watch
[!] Waiting for connection
```

```
maxbazaliy->~» ssh root@localhost -p 5444
The authenticity of host '[localhost]:5444 ([127.0.0.1]:5444)' can't be established.
ECDSA key fingerprint is SHA256:SCM/doXH/pnJVn6dnHz6An/ZbEYAPMWREQlx2ucplgY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:5444' (ECDSA) to the list of known hosts.
root@localhost's password:
-sh-3.2# uname -a
Darwin Apple-Watch 15.4.0 Darwin Kernel Version 15.4.0: Fri Feb 19 13:32:35 PST 2016; root:xnu-3248.41.4~27/RELEASE_ARM_S7002 Watch1,2
```


iTerm2 Shell Edit View Profiles Toolbelt Window Help
100% Fri 8:12 Max Bazaliy
1. maxbazaliy@Maxs-MPB: ~/Desktop/Watch_jailbreak_demo (zsh)
~ (zsh) ⌘1
..ailbreak_demo (zsh) ⌘2
..ailbreak_demo (zsh) ⌘3
..ailbreak_demo (zsh) ⌘4
maxbazaliy->~/Desktop/Watch_jailbreak_demo»

Apple Watch usage

- Watch has access to SMS, Calls, Health
- Photos and emails synced to Watch
- Fetch GPS location from the phone
- Microphone usage
- Apple Pay

Interesting findings

- Full access to jailbroken watch file system
- Including sqlite3 databases
  - Messages
  - Call history
  - Contacts
  - Emails

What's next ?

- Interpose or trampoline system functions
- Catch data on sync with a iPhone
- Create tweaks for a watch
- Run frida and radare

Takeaways

- WatchOS security is equal to iOS
- But new techniques required
- Easier data forensics on a Watch

References

- Lookout - Technical Analysis of the Pegasus Exploits on iOS
- Luca Todesco - com.apple.companion_proxy client
- Siguza - tfp0 powered by Pegasus
- Stefan Esser - iOS 10 - Kernel Heap Revisited

@mbazaliy