Big Game Fuzzing: Going on a Pwn2Own Safari

Alex Plaskett, Fabian Beterke, Georgi Geshev

Introduction

- Provide an overview of our tooling / approach
  - As a bug hunter (but thinking a lot about automated software testing).
- Highlight our experiences / lessons learned over the years
- Insights into the future of browser security

Agenda

MWR LABS

1) Tooling and Automation

2) Browser Vulnerabilities

- Wasm vulnerability (CVE-2018-4121)
- SVG vulnerability (CVE-2018-4199)

3) Sandbox Escape

- Dock vulnerability (CVE-2018-4196)

3) Conclusions

About us

- Fabian Beterke (@pwnfl4k3s) – Security Research @ Bytegeist doing VR / OS security etc. (Pwn2Own Safari 2018)
- Alex Plaskett (@alexjplaskett) – Security Researcher @ MWR doing VR (WP7 jailbreak, Huawei Mate Pwn2Own 2017, Pwn2Own Safari 2018 etc.)
- Georgi Geshev (@munmap) – Security Research @ MWR doing VR (Pwnie Award Winner, Samsung Pwn2Own (2016/2017 etc)

Tooling and Automation

LABS

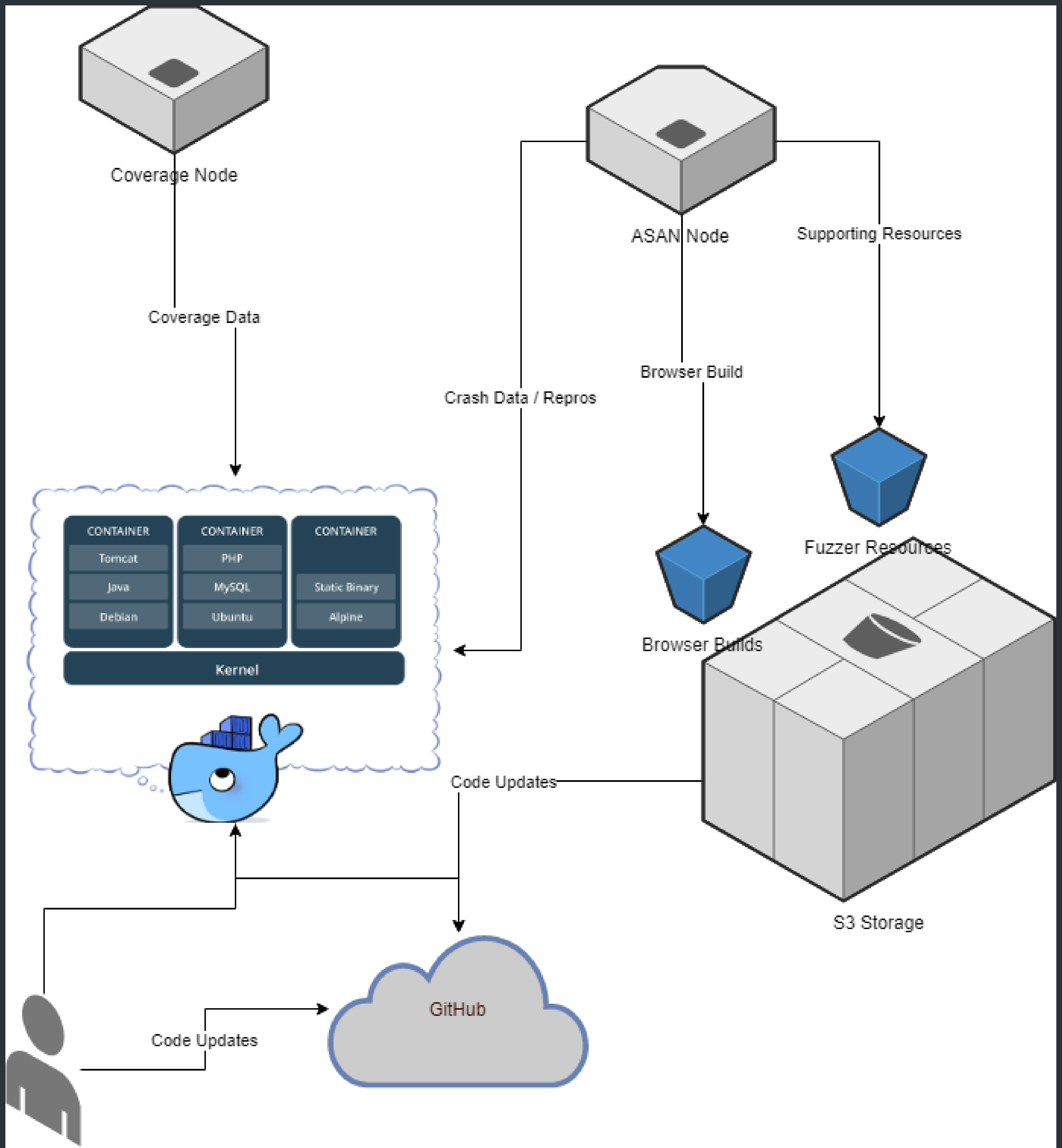
Fuzzing Aims

- High throughput of testcases / code coverage
- Reproducable test cases
- Robust and scalable infrastructure
- Extensible architecture (plug and play deployment of new modules)
- Don’t re-invent the wheel (I keep doing this!)
  - https://github.com/MozillaSecurity have some awesome tools ☺
  - OSS-Fuzz ideas (https://github.com/google/oss-fuzz)
- Allow focus more on bug hunting than infrastructure babysitting!

Fuzzing Modules

- DOM Fuzzers
  - Grammar based
  - Reflection based
  - Mutation based
- JavaScript Interpreter Fuzzers
  - Grammar based
  - AST mutation based (this one is novel in its own right!)
- Specialist Fuzzers
  - WASM / RegEx / JSON

Fuzzing Infrastructure Diagram



AWS Cluster Management

- Initial fuzzing with Azure
  - Collection of PowerShell automation
  - Held together with string! ☺

Moved to AWS:

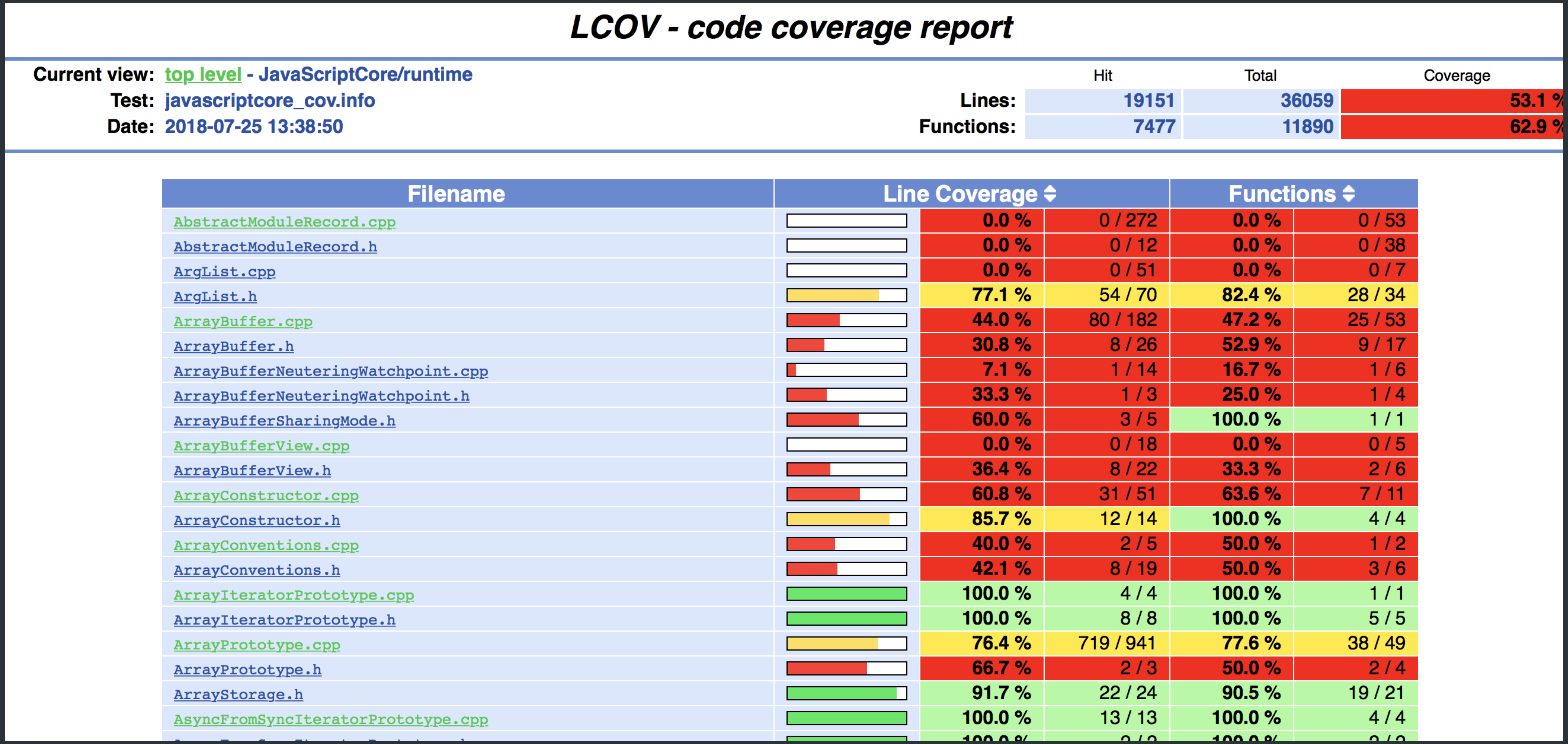
  - Laniakea (https://github.com/MozillaSecurity/laniakea) – Userdata scripts
  - Portainer (https://portainer.io/)

Continuous Fuzzer Code Deployment

- Important to be able to re-deploy to all fuzz nodes (grammar updates etc).
- Want to do this without creating a whole new instance deployment
  - boto / paramiko / GitHub deploy keys
- Code and updated resources pushed to all nodes

Continuous Coverage Monitoring

- lcov / gcov / CovManager
- libfuzzer / sancov



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - JavaScriptCore/runtime				
Test:	javascriptcore_cov.info	Lines:	19151	36059	53.1 %
Date:	2018-07-25 13:38:50	Functions:	7477	11890	62.9 %

Filename	Line Coverage ⬍			Functions ⬍	
AbstractModuleRecord.cpp		0.0 %	0 / 272	0.0 %	0 / 53
AbstractModuleRecord.h		0.0 %	0 / 12	0.0 %	0 / 38
ArgList.cpp		0.0 %	0 / 51	0.0 %	0 / 7
ArgList.h		77.1 %	54 / 70	82.4 %	28 / 34
ArrayBuffer.cpp		44.0 %	80 / 182	47.2 %	25 / 53
ArrayBuffer.h		30.8 %	8 / 26	52.9 %	9 / 17
ArrayBufferNeuteringWatchpoint.cpp		7.1 %	1 / 14	16.7 %	1 / 6
ArrayBufferNeuteringWatchpoint.h		33.3 %	1 / 3	25.0 %	1 / 4
ArrayBufferSharingMode.h		60.0 %	3 / 5	100.0 %	1 / 1
ArrayBufferView.cpp		0.0 %	0 / 18	0.0 %	0 / 5
ArrayBufferView.h		36.4 %	8 / 22	33.3 %	2 / 6
ArrayConstructor.cpp		60.8 %	31 / 51	63.6 %	7 / 11
ArrayConstructor.h		85.7 %	12 / 14	100.0 %	4 / 4
ArrayConventions.cpp		40.0 %	2 / 5	50.0 %	1 / 2
ArrayConventions.h		42.1 %	8 / 19	50.0 %	3 / 6
ArrayIteratorPrototype.cpp		100.0 %	4 / 4	100.0 %	1 / 1
ArrayIteratorPrototype.h		100.0 %	8 / 8	100.0 %	5 / 5
ArrayPrototype.cpp		76.4 %	719 / 941	77.6 %	38 / 49
ArrayPrototype.h		66.7 %	2 / 3	50.0 %	2 / 4
ArrayStorage.h		91.7 %	22 / 24	90.5 %	19 / 21
AsyncFromSyncIteratorPrototype.cpp		100.0 %	13 / 13	100.0 %	4 / 4
[illegible]		[illegible]	[illegible]	[illegible]	[illegible]

Enhancing Coverage

- Feedback Driven
- Enhanced Sample Sets (Stress tests)
- Improved Grammars (new code etc).
- Specialist Fuzzers

Enhanced Crash Detection and Deployment

- Continuous Deployment
  - Build process patches (WebKitGTK)
  - ASAN/MSAN/UBSan
  - Docker all the things!
    - docker-webkit-asan-build
    - docker-webkit-release-build
    - docker-webkit-libfuzzer
  - S3 bucket deployment

Try #1: Wasm Vulnerability

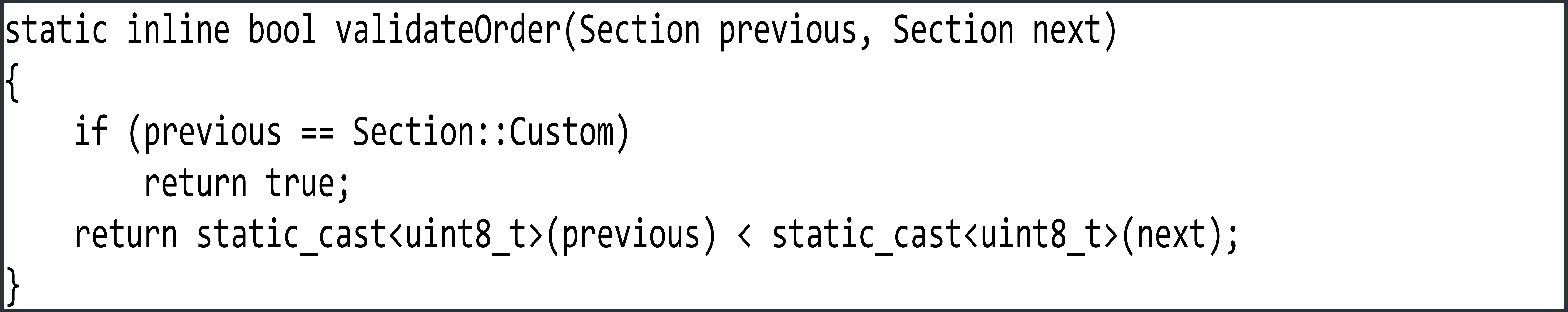
LABS

WebAssembly Heap-Buffer-Overflow

- AKA. CVE-2018-4121
- Found through dumb fuzzing of binary Wasm modules (specialist fuzzer)
- Independently discovered by GPZ’s @natashenka (by code review)
- Writeup released by us in April ☺
- Fairly unstable exploit – reliability of only ~70-80%

CVE-2018-4121

- WebAssembly binaries contain sections
- e.g. type-section, function-section or custom sections
- Expected to be in order, unless…



```
static inline bool validateOrder(Section previous, Section next)
{
    if (previous == Section::Custom)
        return true;
    return static_cast<uint8_t>(previous) < static_cast<uint8_t>(next);
}
```

CVE-2018-4121 (cont.)

- Assumptions about order and uniqueness are wrong
- Results in multiple overflow bugs
- We chose a heap-based buffer-overflow in function section parsing
- PoC: “Type-Section/Function-Section/Custom-Section/Function Section”
  - ModuleParser::parseFunction will be called twice

  => Vector m_info->internalFunctionSignatureIndices will overflow

Exploitation

```
m_info->internalFunctionSignatureIndices.uncheckedAppend(signatureIndex);
```

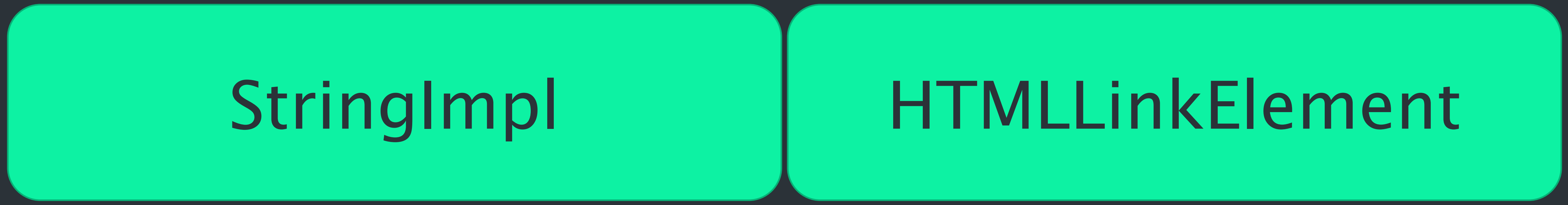
- “Signature Index” refers to index of functions’ type in type-section
- Size of internalFunctionSignatureIndices-Array depends on number of functions in “legit” (first) function-section
- We have influence on all of these ☺
- Caveat: wasm with more than ~1000 sections won’t parse
  - signatureIndex must be < 1000

Exploitation (cont.)

- StringImpl-objects (underlying JS-Strings) have a nice memory layout:

  <4b refcount><4b size><8b dataptr><4b hash & flags><4b mask>

  => can be sprayed with a size of our choice
- Corrupting a StringImpl's size-field allows us to leak some heap memory
- General plan: trigger vuln 2x, first to leak, then to redirect execution
- What to leak though in round #1? We chose HTMLLinkElement's vtable-ptr
- In round #2: overwrite vtable-ptr to get RCE



StringImpl

HTMLLinkElement

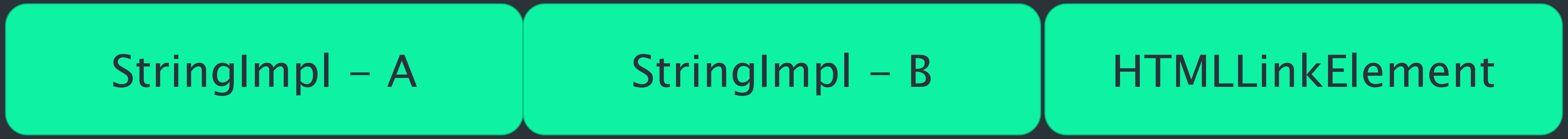
Heap Spray #2

- Use @saelo’s and @niklasb’s Heap-Spray technique
  - Spray 24.5GB worth of ArrayBuffers
  - Some of those will fairly reliably end up at 0x800000000
  - Create fake vtable here, also good space for payload
- Gives controlled, readable and writable data at a known address
- Takes a while, but works well ☺

Exploitation (fin)

- We have a plan now!

1. Spray a pattern of 2x appropriately sized StringImpl (A&B) + 1x target object



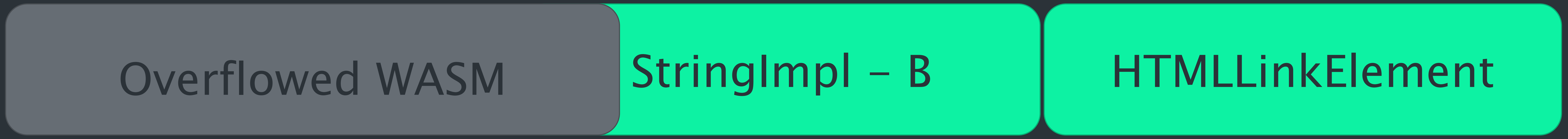
StringImpl – A

StringImpl – B

HTMLLinkElement

Exploitation (fin)

- We have a plan now!
  1. Spray a pattern of 2x appropriately sized StringImpl (A&B) + 1x target object
  2. Free every A, leaving space for the buffer to be overflowed
  3. Trigger bug the 1st time to overwrite B’s size, read back for leaked vtable-ptr



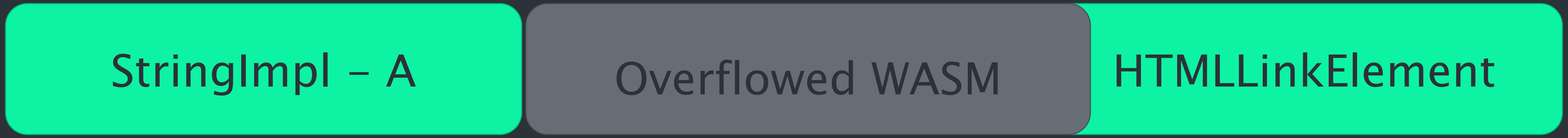
Overflowed WASM

StringImpl – B

HTMLLinkElement

Exploitation (fin)

- We have a plan now!
  1. Spray a pattern of 2x appropriately sized StringImpl (A&B) + 1x target object
  2. Free every A, leaving space for the buffer to be overflowed
  3. Trigger bug the 1st time to overwrite B's size, read back for leaked vtable-ptr
  4. Spray the same pattern again, but this time, freeing every B
  5. Spray ROP-chain and trigger bug a 2nd time, corrupting vtable-ptr of target obj



Exploitation (fin)

- We have a plan now!

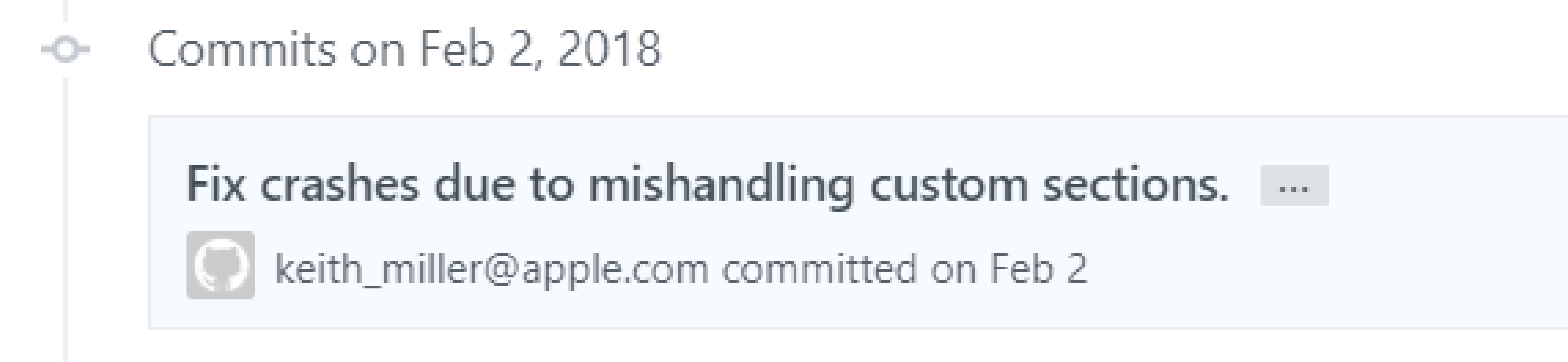
1. Spray a pattern of 2x appropriately sized StringImpl (A&B) + 1x target object
2. Free every A, leaving space for the buffer to be overflowed
3. Trigger bug the 1st time to overwrite B’s size, read back for leaked vtable-ptr
4. Spray the same pattern again, but this time, freeing every B
5. Spray ROP-chain and trigger bug a 2nd time, corrupting vtable-ptr of target obj

=> RCE \o/

https://github.com/mwrlabs/CVE-2018-4121 ☺

The Darkest Day

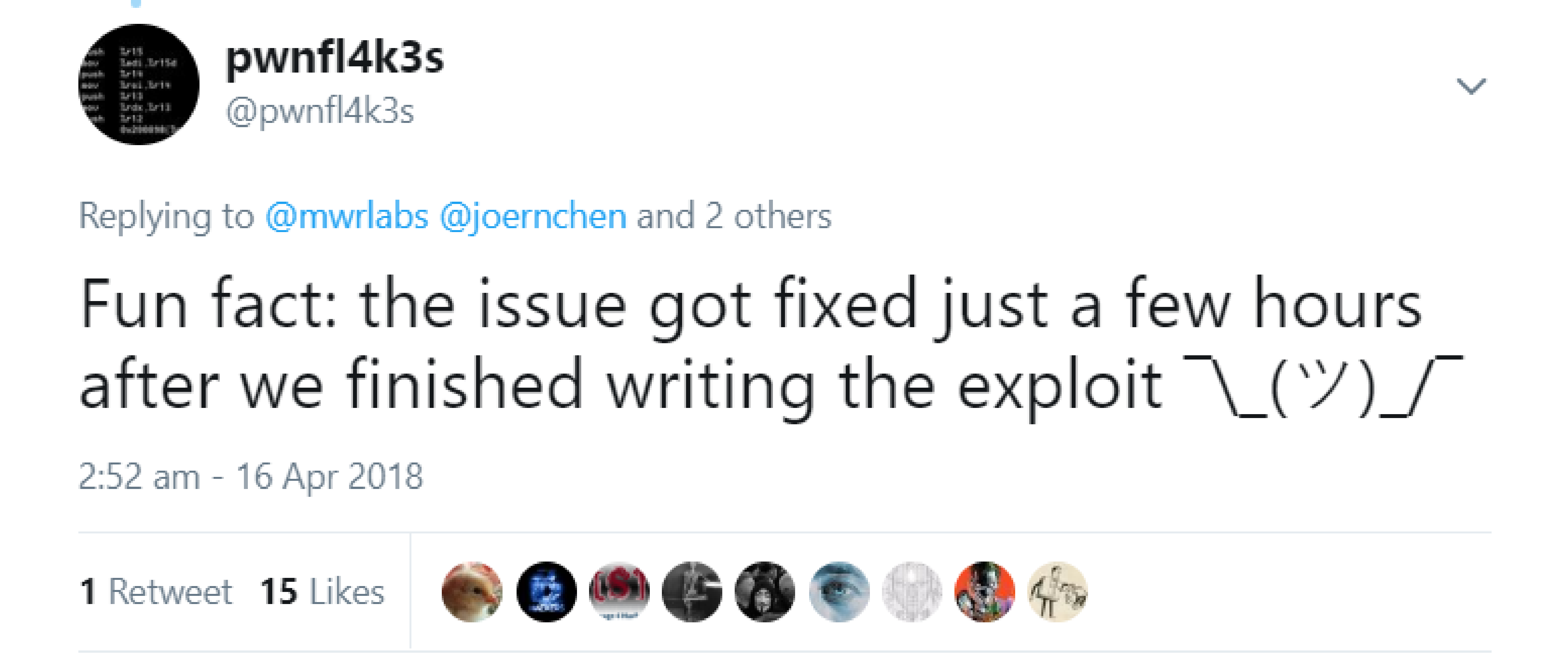
- Commit c6deeea41e524d071382a5d0fe380fbd7b634c32



Commits on Feb 2, 2018

Fix crashes due to mishandling custom sections. ...

keith_miller@apple.com committed on Feb 2



pwnfl4k3s
@pwnfl4k3s

Replying to @mwrlabs @joernchen and 2 others

Fun fact: the issue got fixed just a few hours after we finished writing the exploit ¯_(ツ)_/¯

2:52 am - 16 Apr 2018

1 Retweet 15 Likes

Try #2: SVG Vulnerability

LABS

SVG Heap-Buffer “Overflow”

- AKA. CVE-2018-4199 / ZDI-CAN-5828
- Found using bytegeist’s DOM-fuzzer
- Very powerful bug (even better than the first one)
- Nearly 100% reliability

SVG Path Segments

- SVG paths (think lines or curves) consist of lists of path segments
- segment lists provide a rich interface for manipulating paths
  - $(“#svgpath”).pathSegList.getItem(1)
- Other than that, you can use the “classic” XML-style
  - <svg><path id=“svgpath” d=“M 0 1 1 2”/><svg>
- What happens if we do both?

Meet CVE-2018-4199!

- PathSegList-API provides an interesting function: insertItemBefore(seg, idx)
- Specs require that seg “is the item itself and not a copy”
  - if it’s in another list already, remove it from that one
  - if it’s already at the correct index, do nothing
  - browser must keep track of old path segment lists
- What happens if we replace the whole PathSegList and then insertItemBefore?
  - e.g. by doing $(“#svgpath”).setAttribute(“d”, “M 13 37”);
- You guessed it: chaos ☺

CVE-2018-4196 (cont.)

```
var seglist = $(“path”).pathSegList;
seglist.insertItemBefore(seg, 1);
$(“path”).setAttribute(“d”,”M 0 0”);
seglist.insertItemBefore(seg, 1); // BOOM
```

- As the segment is still associated with a list, it is determined to be removed
- A “find” call is used to retrieve the index, but returns -1 as the segment is not in the (new) segment list
- Logical conclusion: replace the “item” at segment_list[-1]; ☺

Heap-Buffer Underflow!

- Interesting situation – treats uint64 right before the buffer as SVGPathSeg ptr
- Two questions come to mind:
  - A) can we control that memory?
  - B) if yes, what can we do with this?
- A: yes, we can!
  - High degree of control as size of the underlying pointer-vector is up to us
  - spray SVG transform lists to get adjacent read-write float-vectors of arbitrary size

B) What can we do with this?

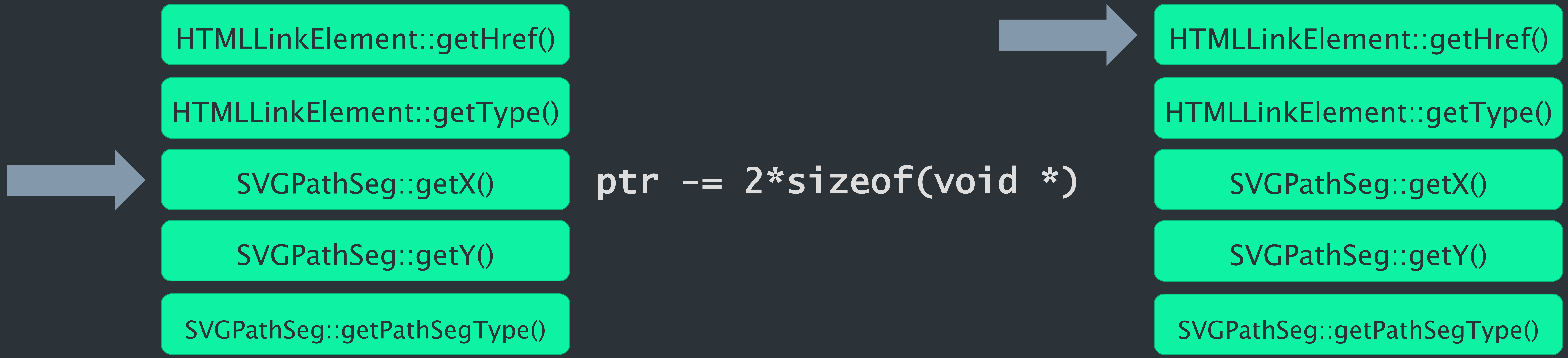
- insertItemBefore actually has different behavior depending on what it finds at the index of insertion:
  1. If non-null: need to remove existing item first
  2. If null: nothing more to do, simply place seg here
- Could hardly be any better for us:
  - behavior #1 will try to drop a reference -> gives a (nearly) arbitrary decrement
    - nearly because if refcount == 1, ptr will be passed to free() and we crash
  - We can use behavior #2 to leak a pointer to a SVGPathSeg ☺

Exploitation Battle Plan

- Recap: we now have a pointer to one of our SVGPathSeg-items and a pretty-much-arbitrary decrement primitive
- Also, since we can replace the “confused” memory at will, we can retrigger the vuln as often as we want without risking a crash
- Idea: turn this into a full-fledged arbitrary write using arbitrary read
  - arbitrary decrement + arbitrary read = arbitrary write
  - use read to check if *(int32*)target is 1, if so, decrement target-1 until wraps to 0
- How to achieve an arbitrary read though?

Arbitrary Read?

- Crazy idea: decrement vtable pointer of our leaked seg to call a virtual function of another class on our object
- How to use that?
  - Decrement the ptr so that a getter (e.g. pathSegType) points to a different func

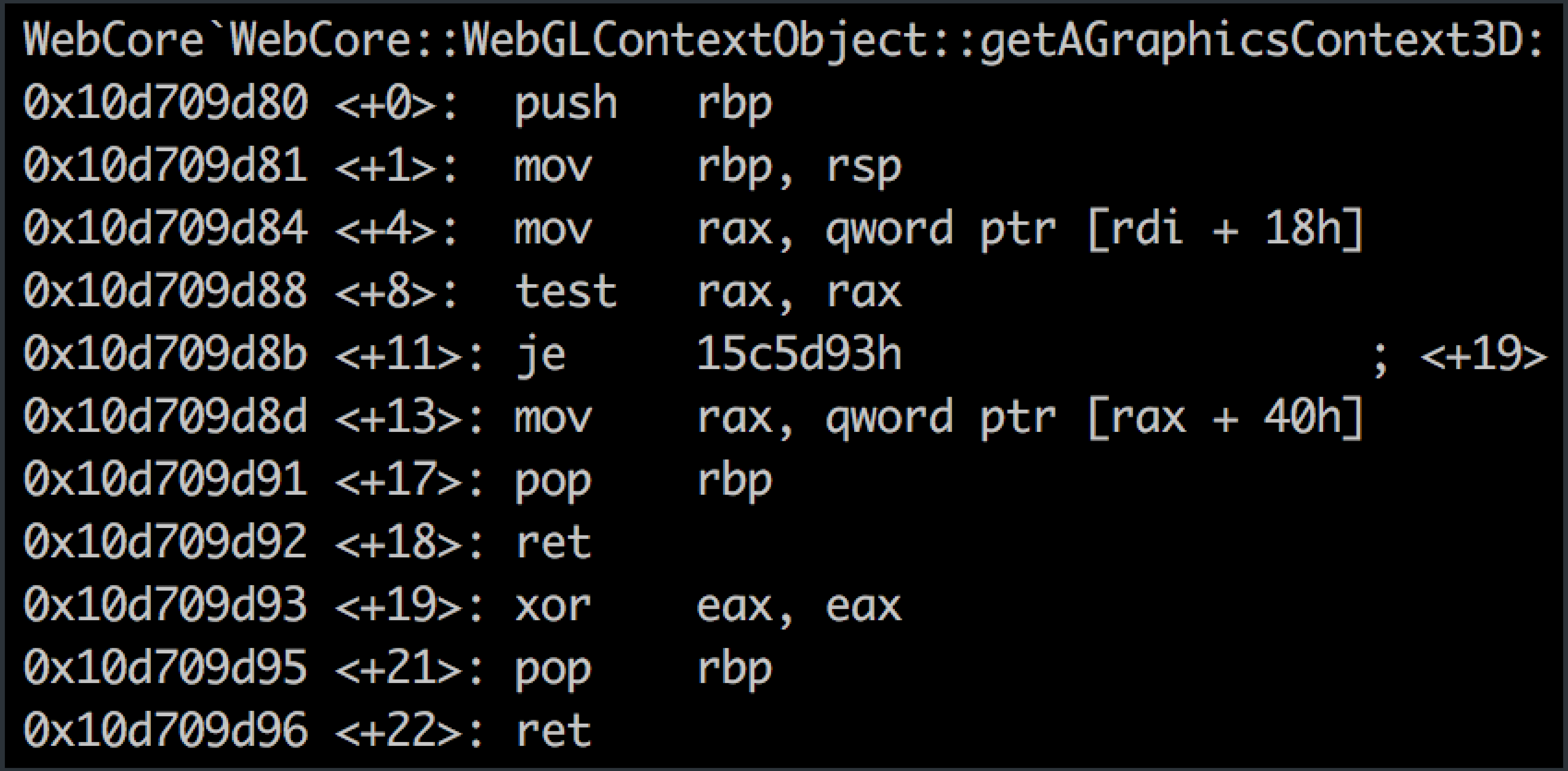


HTMLLinkElement::getHref()
HTMLLinkElement::getType()
SVGPathSeg::getX()
SVGPathSeg::getY()
SVGPathSeg::getPathSegType()
ptr -= 2*sizeof(void *)
HTMLLinkElement::getHref()
HTMLLinkElement::getType()
SVGPathSeg::getX()
SVGPathSeg::getY()
SVGPathSeg::getPathSegType()

Arbitrary Read!

- But what function to call?
  - Setting our seg.x and seg.y coordinates writes two float32s into the seg object at offsets +0x18 and +0x1c, respectively
- Is there a virtual function that derefs rdi+0x18 and returns the result?
  - good ol’ grep to the rescue!
  - `grep "mov.*24(.rdi.," -A4 disas.txt | grep "\(mov.*(%r..), %.ax\)\|\(ret\)"`

Well, hello there!



```
WebCore`WebCore::WebGLContextObject::getAGraphicsContext3D:
0x10d709d80 <+0>:  push   rbp
0x10d709d81 <+1>:  mov    rbp, rsp
0x10d709d84 <+4>:  mov    rax, qword ptr [rdi + 18h]
0x10d709d88 <+8>:  test   rax, rax
0x10d709d8b <+11>: je     15c5d93h                  ; <+19>
0x10d709d8d <+13>: mov    rax, qword ptr [rax + 40h]
0x10d709d91 <+17>: pop    rbp
0x10d709d92 <+18>: ret
0x10d709d93 <+19>: xor    eax, eax
0x10d709d95 <+21>: pop    rbp
0x10d709d96 <+22>: ret
```

☺

Arbitrary Read/Write to RCE

- Equipped with full r/w, what to do next? ROP is for the 99%...
- There are JITStubRoutine objects on the heap
  - contain a ptr to MacroAssemblerCodeRef obj, which contains a ptr to rwx memory
  - following those pointers gives us an address of rwx memory
- Write shellcode there, then change a vtable-entry to that pointer
- Call corresponding virtual func to enter shellcode ☺

From Shellcode to Stage2

- Fairly straightforward path of action:
  1. data = document.createComment(<bytestring of compiled dylib>)
  2. pathElement.appendChild(data)
  3. use read to follow a few pointers from one of the leaked segments
     - Segment -> Path element -> firstChild (comment) -> string -> contents
  4. write "contents"-pointer into shellcode
  5. In shellcode: write dylib code to a file and dlopen() it => WIN ☺

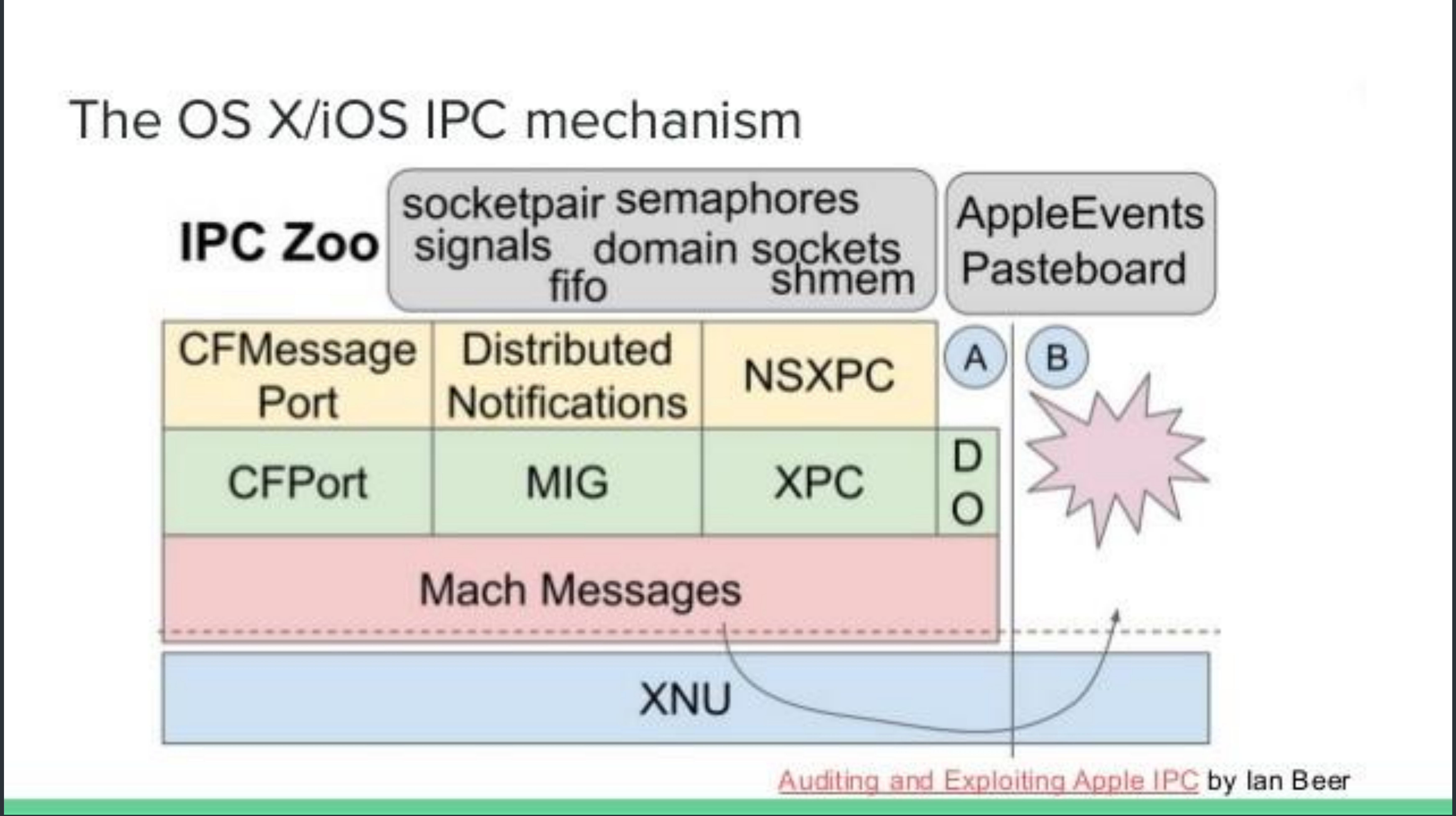
Sandbox Escape

LABS

WebCore Sandbox Details

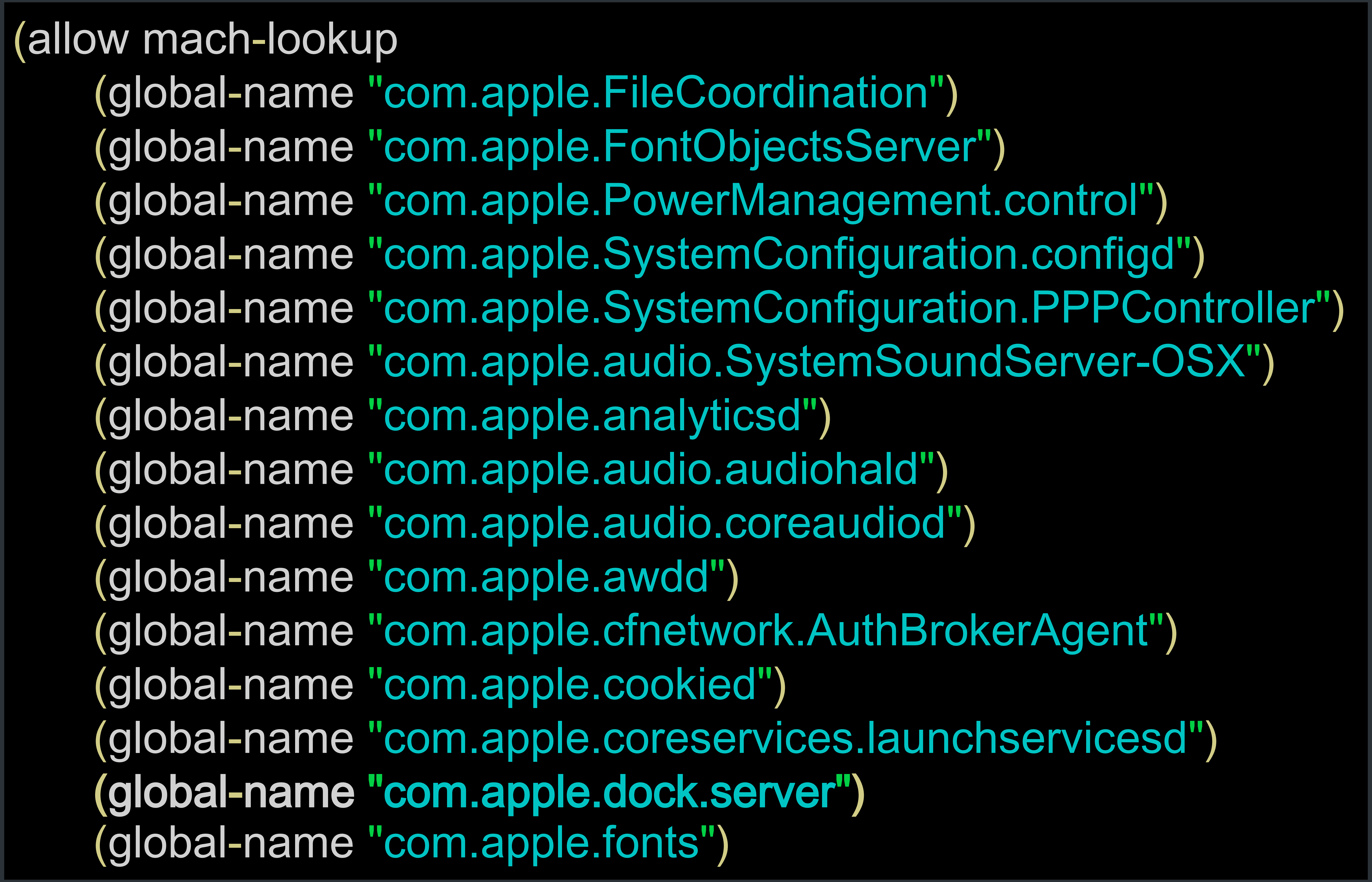
- At this point achieved code execution in the content process.
- Potential Approaches:
  - IPC Vulnerability
  - UIProcess Vulnerability
  - Kernel Vulnerability
- Previous work:
  - Nemo
  - Ian Beer
  - https://labs.mwrinfosecurity.com/publications/biting-the-apple-that-feeds-you-macos-kernel-fuzzing/

macOS IPC Overview



Auditing and Exploiting Apple IPC by Ian Beer

WebCore Sandbox Profile



```
(allow mach-lookup
       (global-name "com.apple.FileCoordination")
       (global-name "com.apple.FontObjectsServer")
       (global-name "com.apple.PowerManagement.control")
       (global-name "com.apple.SystemConfiguration.configd")
       (global-name "com.apple.SystemConfiguration.PPPController")
       (global-name "com.apple.audio.SystemSoundServer-OSX")
       (global-name "com.apple.analyticsd")
       (global-name "com.apple.audio.audiohald")
       (global-name "com.apple.audio.coreaudiod")
       (global-name "com.apple.awdd")
       (global-name "com.apple.cfnetwork.AuthBrokerAgent")
       (global-name "com.apple.cookied")
       (global-name "com.apple.coreservices.launchservicesd")
       (global-name "com.apple.dock.server")
       (global-name "com.apple.fonts")
```

Dock Overview

- Used to manage the Dock GUI on macOS
- Runs as same permissions as logged in user (however, unsandbox’d!).
- Multiple different endpoint’s (XPC, Mach IPC etc.).
- Focused on the MIG based Mach IPC

MIG Introduction

- Mach Interface Generator (MIG)
- Generates C/C++ messages for sending messages between tasks
- .defs file contains the description of the interface.
- mach_msg trap

Reversing Mach Messages (osfmk/mach/mig.h)

- Start from bootstrap_check_in function and xref MSHCreateMIGServerSource function.
- CFRunLoopSourceRef MSHCreateMIGServerSource(CFAllocatorRef, CFIndex order, mig_subsystem_t **sub_system**, MSHCreateOptions, mach_port_t, void* user_data);

Reversing Mach Messages (osfmk/mach/mig.h)

```
typedef struct mig_subsystem {
        mig_server_routine_t server; /* pointer to demux routine */
        mach_msg_id_t start; /* Min routine number */
        mach_msg_id_t end; /* Max routine number + 1 */
        mach_msg_size_t maxsize; /* Max reply message size */
        vm_address_t reserved; /* reserved for MIG use */
        mig_routine_descriptor routine[1]; /* Routine descriptor array */
} *mig_subsystem_t;
```

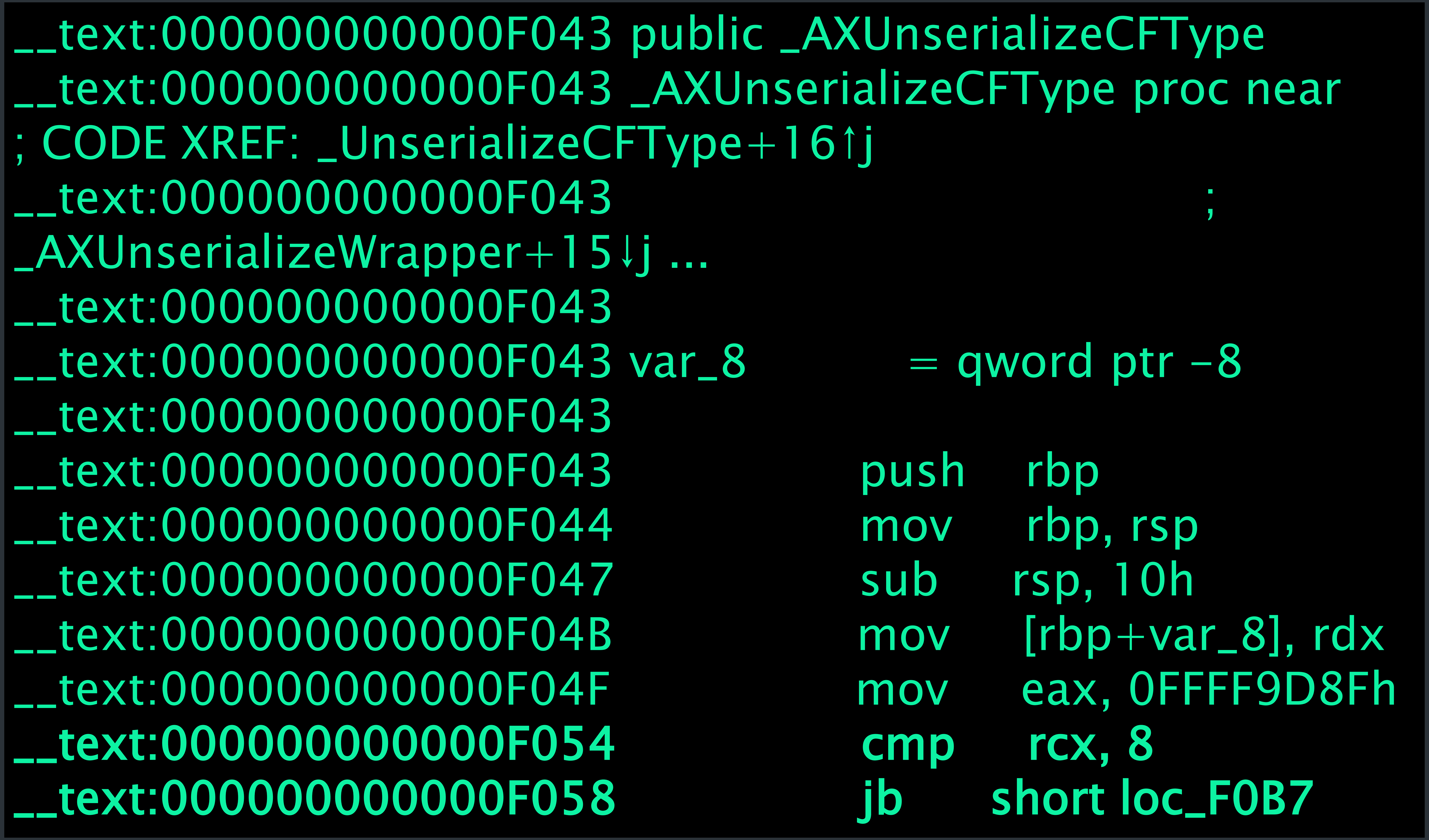
```
struct routine_descriptor {
        mig_impl_routine_t impl_routine; /* Server work func pointer */
        mig_stub_routine_t stub_routine; /* Unmarshalling func pointer */
        unsigned int argc; /* Number of argument words */
        unsigned int descr_count; /* Number complex descriptors */
        routine_arg_descriptor_t arg_descr; /* pointer to descriptor array*/
        unsigned int max_reply_msg; /* Max size for reply msg */
};
```

Dock Vulnerability (CVE-2018-4196)

```
Vuln Routine:
mov     esi, r14d
lea     r15, [rbp+var_48]
mov     rdi, r12
mov     rdx, r15
call    _UnserializeCFType ; Call ‘UnserializeCFType’ and
store unserialised data in $r15.
mov     r13d, eax
mov     rdi, [r15]
call    _objc_autorelease ; Pass the unserialised object to
‘objc_autorelease’.

_UnserializeCFType:
__text:000000000000F03A                 pop     rbp
__text:000000000000F03B                 jmp
_AXUnserializeCFType
```

AXUnserializeCFType



```
__text:000000000000F043 public _AXUnserializeCFType
__text:000000000000F043 _AXUnserializeCFType proc near
; CODE XREF: _UnserializeCFType+16↑j
__text:000000000000F043                                   ;
_AXUnserializeWrapper+15↓j ...
__text:000000000000F043
__text:000000000000F043 var_8          = qword ptr -8
__text:000000000000F043
__text:000000000000F043                push    rbp
__text:000000000000F044                mov     rbp, rsp
__text:000000000000F047                sub     rsp, 10h
__text:000000000000F04B                mov     [rbp+var_8], rdx
__text:000000000000F04F                mov     eax, 0FFFF9D8Fh
__text:000000000000F054                cmp     rcx, 8
__text:000000000000F058                jb      short loc_F0B7
```

Dock Vulnerability (Trigger Code)

```
mov     esi, r14d
lea     r15, [rbp+var_48]
mov     rdi, r12
mov     rdx, r15
call    _UnserializeCFType ; Call ‘UnserializeCFType’
and store unserialised data in $r15.
mov     r13d, eax
mov     rdi, [r15] ; [R15] can be uninitialized
call    _objc_autorelease ; Pass the unserialised
object to ‘objc_autorelease’.
```

Dock Vulnerability (objc_autorelease)

```
…


 0x7fff54c97991 <+113>: mov    qword ptr
gs:[0x160], 0x1
    0x7fff54c9799e <+126>: jmp    0x7fff54c9798d
; <+109>
    0x7fff54c979a0 <+128>: lea    rax, [rip +
0x3a10bbd1]   ; SEL_autorelease
    0x7fff54c979a7 <+135>: mov    rsi, qword ptr
[rax]
    0x7fff54c979aa <+138>: jmp    0x7fff54c91e80
; objc_msgSend
…
```

Uninitialized Memory Exploitation

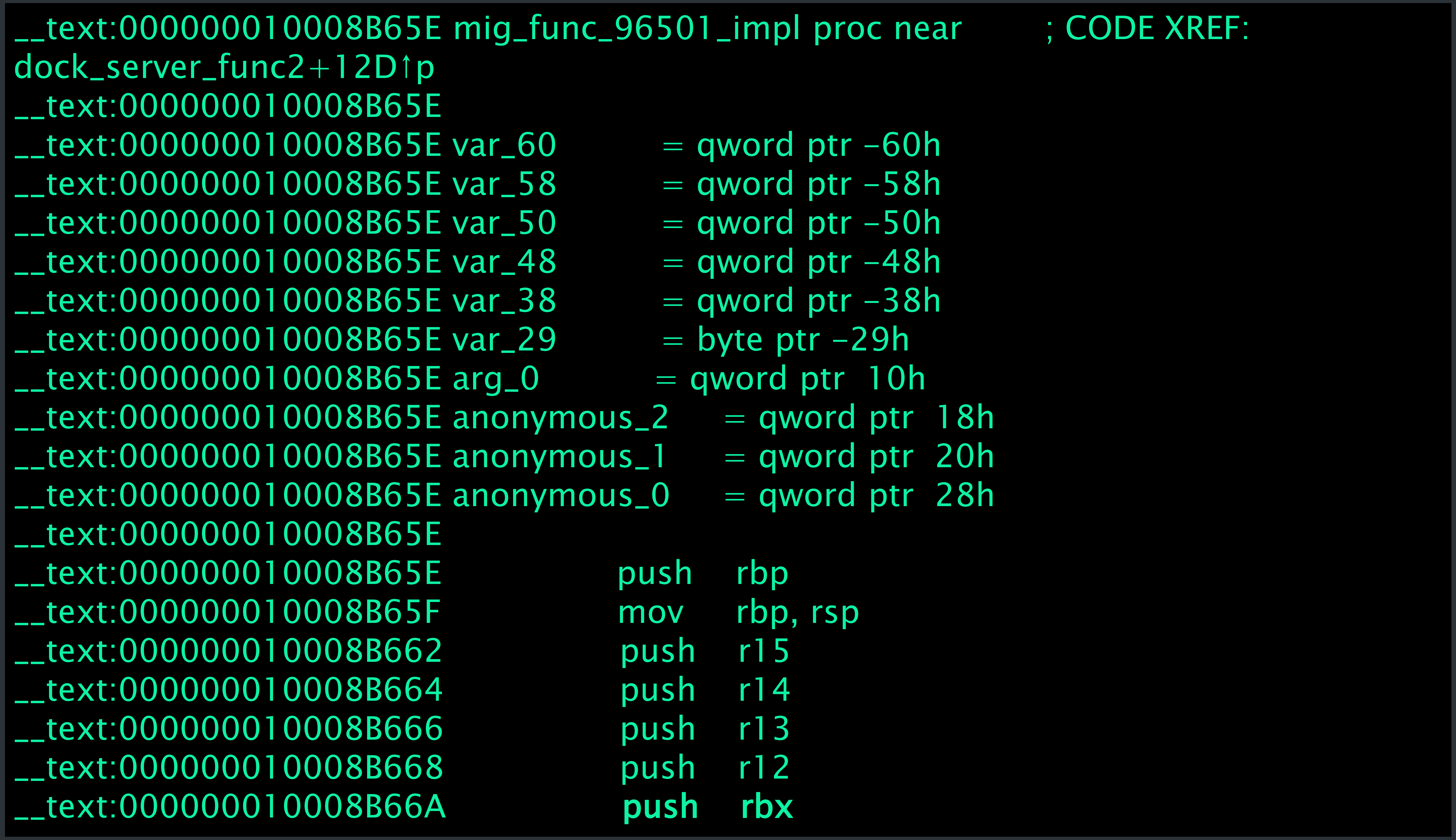
- Need to initialize the stack pointer to something attacker controlled.
- https://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Flake.pdf
- One function stood out due to large number of ‘push’ instructions.
- A ‘push rbx’ instruction hit our offset on the stack whilst setting ‘rsp’ to value of ‘rbx’
- Coincidentally rbx pointing at start of mach message which is also on the stack.

Uninitialized Memory Exploitation (Setup Function)

- Mach message buffer allocated by ‘mshMIGPerform’ function.
- Receives a pointer to our message via ‘rdi’ which is later moved to ‘rbx’
- This then end’s up pointing at the message

```
__text:0000000100070CF1 mig_func_96501 proc near          ; DATA XREF:
__const:000000010052B970↓o
….
__text:0000000100070CF1
__text:0000000100070CF1             push    rbp
__text:0000000100070CF2             mov     rbp, rsp
__text:0000000100070CF5             push    r15
__text:0000000100070CF7             push    r14
__text:0000000100070CF9             push    r13
__text:0000000100070CFB             push    r12
__text:0000000100070CFD             push    rbx
__text:0000000100070CFE             sub     rsp, 48h
__text:0000000100070D02             mov     r14, rsi
__text:0000000100070D05             mov     rbx, rdi
__text:0000000100070D08             mov     r12d, [rbx+4]
__text:0000000100070D0C             lea     eax, [r12-2Ch]
__text:0000000100070D11             cmp     eax, 400h
```

Uninitialized Memory Exploitation (Setup Function)

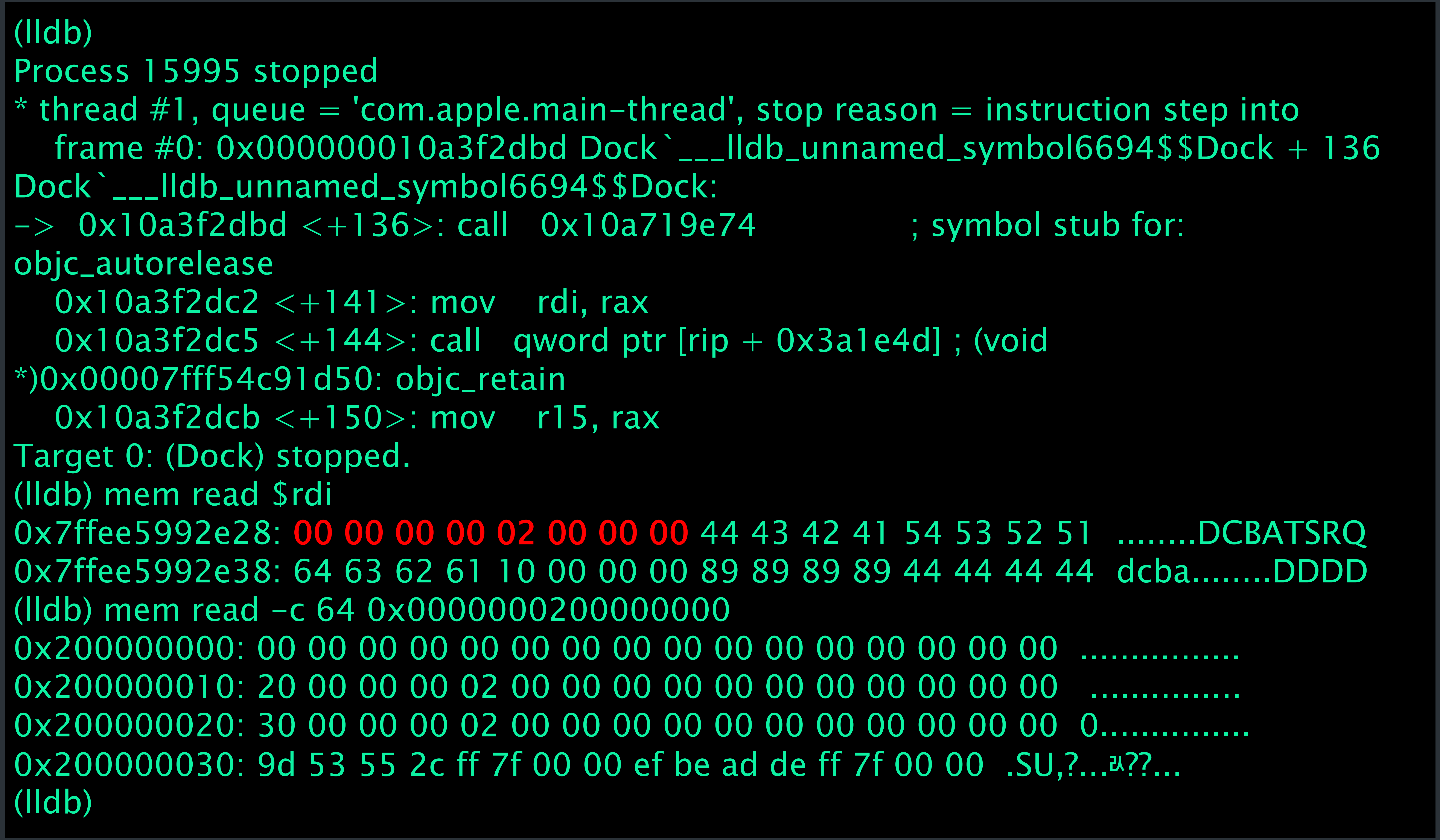


```
__text:000000010008B65E mig_func_96501_impl proc near       ; CODE XREF:
dock_server_func2+12D↑p
__text:000000010008B65E
__text:000000010008B65E var_60          = qword ptr -60h
__text:000000010008B65E var_58          = qword ptr -58h
__text:000000010008B65E var_50          = qword ptr -50h
__text:000000010008B65E var_48          = qword ptr -48h
__text:000000010008B65E var_38          = qword ptr -38h
__text:000000010008B65E var_29          = byte ptr -29h
__text:000000010008B65E arg_0           = qword ptr  10h
__text:000000010008B65E anonymous_2         = qword ptr  18h
__text:000000010008B65E anonymous_1         = qword ptr  20h
__text:000000010008B65E anonymous_0         = qword ptr  28h
__text:000000010008B65E
__text:000000010008B65E                 push    rbp
__text:000000010008B65F                 mov     rbp, rsp
__text:000000010008B662                 push    r15
__text:000000010008B664                 push    r14
__text:000000010008B666                 push    r13
__text:000000010008B668                 push    r12
__text:000000010008B66A                 push    rbx
```

Uninitialized Memory Exploitation

- We need to ensure that this pointer will not be changed between different messages
- Can use LLDB to attach to Dock
  - Initialize the pointer with our first message.
  - Trigger the bug with the second message.
- Pointer remained unchanged between the two messages.
- However message trigger resulted in slightly different stack frame setup
  - 40 bytes into mach message.

Uninitialized Memory Exploitation



```
(lldb)
Process 15995 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = instruction step into
    frame #0: 0x000000010a3f2dbd Dock`___lldb_unnamed_symbol6694$$Dock + 136
Dock`___lldb_unnamed_symbol6694$$Dock:
->  0x10a3f2dbd <+136>: call   0x10a719e74               ; symbol stub for:
objc_autorelease
    0x10a3f2dc2 <+141>: mov    rdi, rax
    0x10a3f2dc5 <+144>: call   qword ptr [rip + 0x3a1e4d] ; (void
*)0x00007fff54c91d50: objc_retain
    0x10a3f2dcb <+150>: mov    r15, rax
Target 0: (Dock) stopped.
(lldb) mem read $rdi
0x7ffee5992e28: 00 00 00 00 02 00 00 00 44 43 42 41 54 53 52 51  ........DCBATSRQ
0x7ffee5992e38: 64 63 62 61 10 00 00 00 89 89 89 89 44 44 44 44  dcba........DDDD
(lldb) mem read -c 64 0x0000000200000000
0x200000000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x200000010: 20 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00   ...............
0x200000020: 30 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00  0...............
0x200000030: 9d 53 55 2c ff 7f 00 00 ef be ad de ff 7f 00 00  .SU,?...ﾭ??...
(lldb)
```

Overall Exploit Stages (Stage 1)

- Spray ‘VM_ALLOCATE’ zone with forged Objective-C objects.
- 1088 Mach Messages each carrying 0x400000 as an ool descriptor
- This results in coving the page at 0x0000000200000000
- This is how we exploit the obj-c autorelease part (Nemo et al).

Overall Exploit Stages (Stage 2)

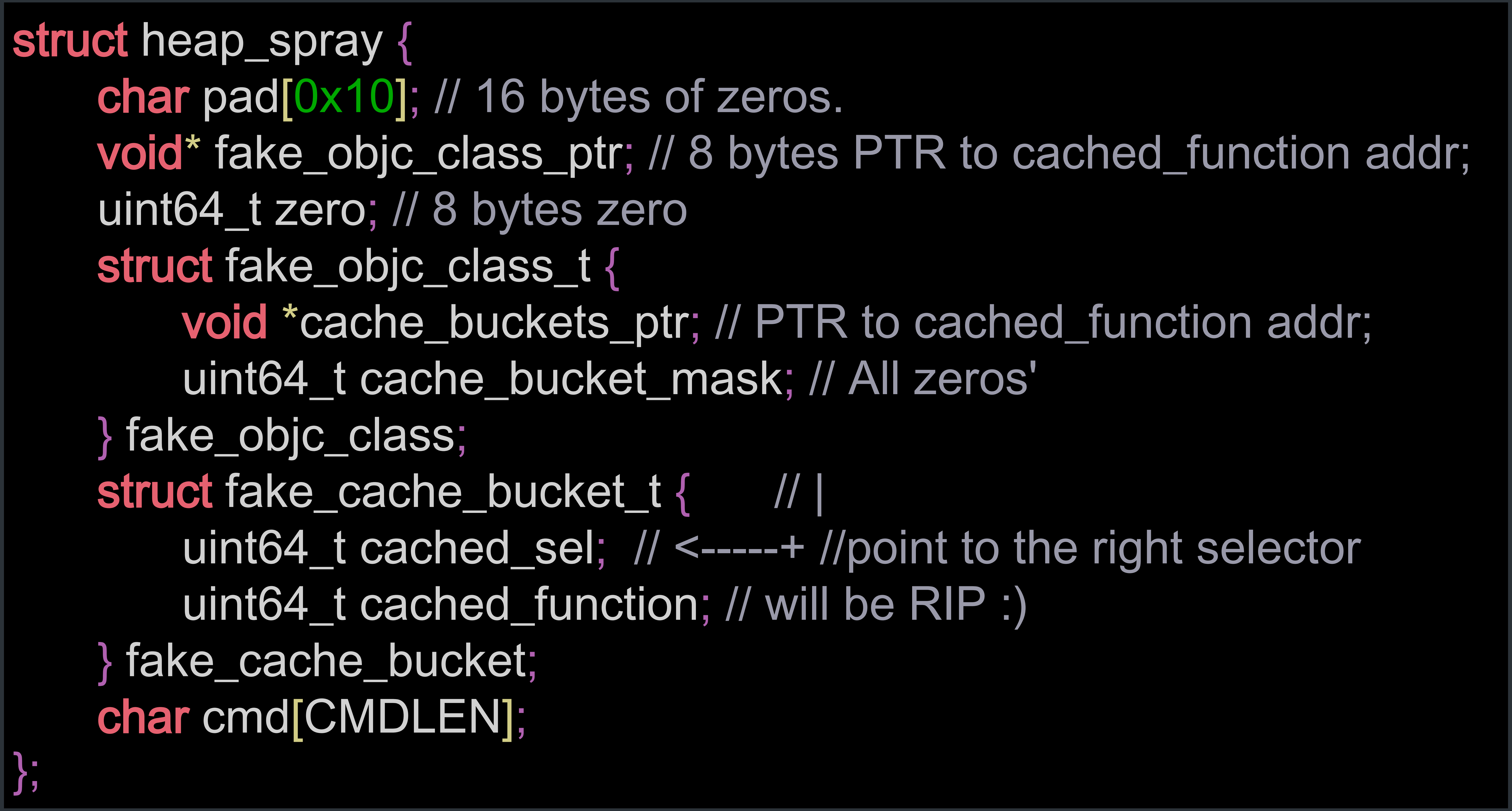
- Send single message of type 96501 to initialize the offset on the stack to be a pointer into the currently processed Mach message.
- This pointer remains on the stack!

Overall Exploit Stages (Stage 3)

- Send message of type 96548 (trigger). Pointer is now referencing current mach message + 40 bytes.
- UnserializeCFType calls AXUnserializeCFType which fails due to length check.
- This controlled pointer is then passed to objc_autorelease.
- Boom!

Objective-C Autorelease

Nemo – http://phrack.org/issues/69/9.html



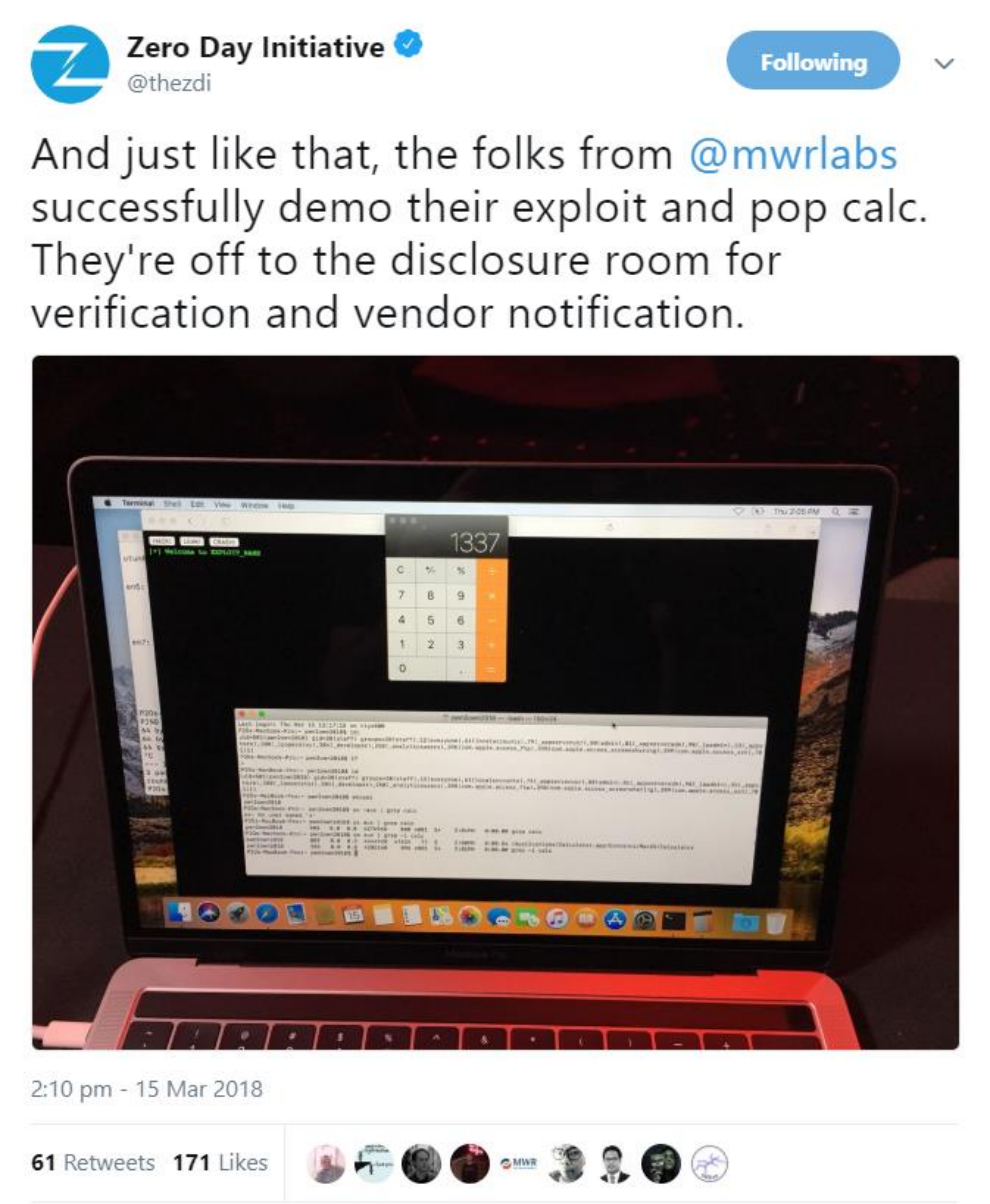
```
struct heap_spray {
    char pad[0x10]; // 16 bytes of zeros.
    void* fake_objc_class_ptr; // 8 bytes PTR to cached_function addr;
    uint64_t zero; // 8 bytes zero
    struct fake_objc_class_t {
        void *cache_buckets_ptr; // PTR to cached_function addr;
        uint64_t cache_bucket_mask; // All zeros'
    } fake_objc_class;
    struct fake_cache_bucket_t {      // |
        uint64_t cached_sel;  // <-----+ //point to the right selector
        uint64_t cached_function; // will be RIP :)
    } fake_cache_bucket;
    char cmd[CMDLEN];
};
```

ROP’Time!

- What about the ROP chain?
  - Not a problem: addresses of dynamically loaded libraries are randomized on boot
  - We can find addresses by calling dlsym from the compromised renderer, they will be the same in the Dock-process ☺

ROP to command exec:

```
#define COMMAND "osascript -e 'tell application \"Terminal\" to do script \"id;\"'; osascript -e 'tell application \"Calculator\" to activate'; osascript -e 'tell application \"System Events\" to keystroke \"1337\"'; osascript -e 'tell application \"Terminal\" to activate';"
```



Zero Day Initiative
@thezdi

Following

And just like that, the folks from @mwrlabs successfully demo their exploit and pop calc. They're off to the disclosure room for verification and vendor notification.

2:10 pm - 15 Mar 2018

61 Retweets 171 Likes

Conclusion

LABS

The Situation Today

- SVG float vectors are still on the unprotected FastMalloc heap
- Same for WebAssembly int vectors
- Huge heap-sprays to predictable addresses still work on both macOS and iOS
- The JITStubRoutine exploit technique has been mitigated
  - now uses tagged pointers instead of raw pointers to executable code
  - might still be bypassable given arbitrary read if the poison value can be leaked
- Apple are doing attack surface reduction for IPC in Mojave (WindowServer) is outside of the profile now.

Code Releases

- https://github.com/mwrlabs/

- Exploit code and whitepaper released soon!

Credits!

- Nemo (http://phrack.org/issues/66/4.html)
- Ian Beer (https://thecyberwire.com/events/docs/IanBeer_JSS_Slides.pdf)
- Halvar (https://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Flake.pdf)
- Saelo & niklasb (https://phoenhex.re/)