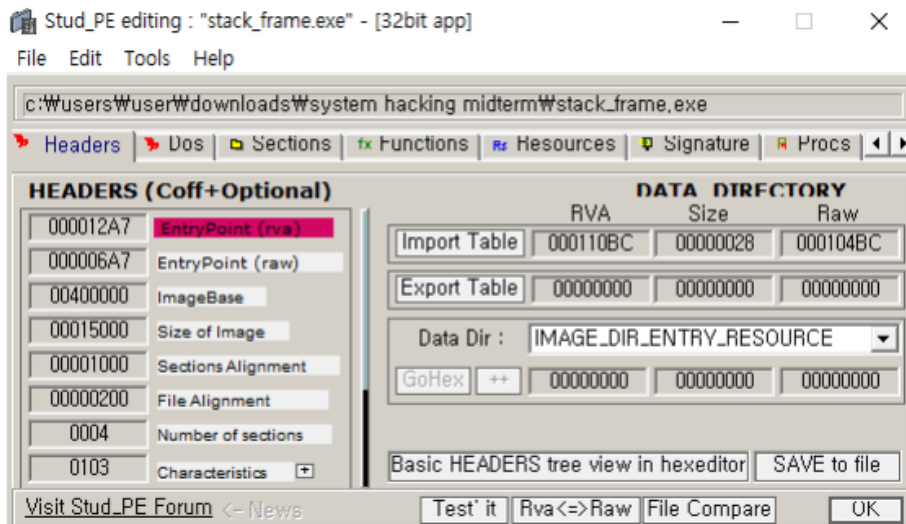


< 실행파일 및 코드 분석 영역 (Stud_PE, IDA) >

Q1. 해당 실행 파일이 매핑될 메모리상의 가상주소(Imagebase)는 무엇입니까?

정답 : 0x00400000

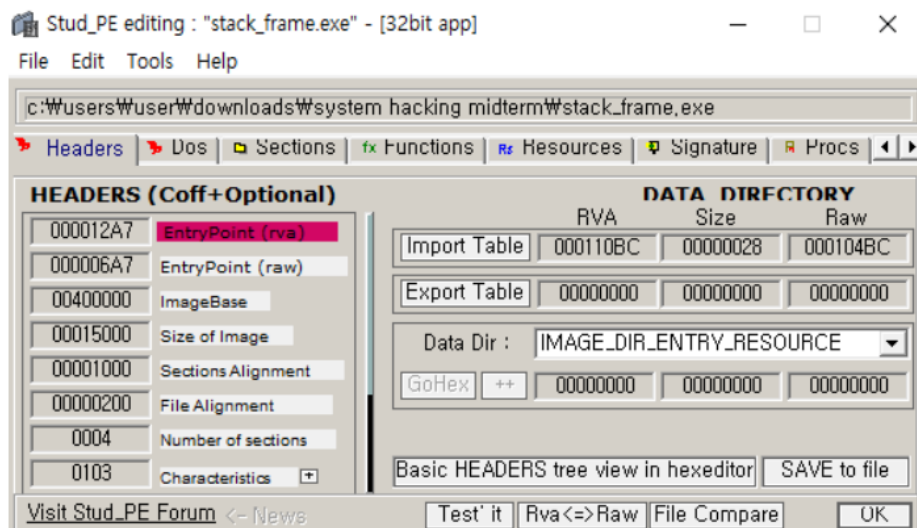
증명 : Stud_Pe를 통해 실행 파일을 열어 ImageBase값을 확인해 보면 0x00400000 인 것을 확인할 수 있다.



Q2. 해당 실행 파일이 메모리에 로드된 후 가장먼저 실행되는 코드의 주소(Address of Entrypoint)는 무엇입니까? (옳은 값 말고 메모리 상의 가상주소 값을 적어 주세요)

정답 : 0x004012A7

증명 : Stud_PE에서 볼 수 있는 Imagebase값인 0x00400000과 EntryPoint(rva)값인 000012A7을 더해주면 $0x00400000 + 0x000012A7 = 0x004012A7$ 이 된다.



Q3. 함수 "Func_A"가 호출되는 지점의 가상주소는 무엇입니까?

정답 : "Func_A"는 CALL을 통해 0x0040106C지점에서 호출되어 0x00401010부터 실행됨.

증명 : IDA를 통해 확인한 결과 함수 "Func_A"는 CALL을 통해 0x0040106C에서 호출되고, 호출 된 후 함수가 시작되는 지점의 주소는 0x00401010인 것을 확인 할 수 있다.

```
.text:00401050
.text:00401050 ; ===== S U B R O U T I N E =====
.text:00401050
.text:00401050 ; Attributes: bp-based frame
.text:00401050
.text:00401050 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401050 _main      proc near          ; CODE XREF: __scrt_common_main_seh+F4↓p
.text:00401050
.text:00401050 var_8      = dword ptr -8
.text:00401050 var_4      = dword ptr -4
.text:00401050 argc      = dword ptr  8
.text:00401050 argv      = dword ptr  0Ch
.text:00401050 envp      = dword ptr  10h
.text:00401050
.text:00401050 push     ebp
.text:00401051 mov      ebp, esp
.text:00401053 sub      esp, 8
.text:00401056 mov      [ebp+var_8], 3
.text:0040105D mov      [ebp+var_4], 4
.text:00401064 mov      eax, [ebp+var_4]
.text:00401067 push     eax
.text:00401068 mov      ecx, [ebp+var_8]
.text:0040106B push     ecx
.text:0040106C call     Func_A
.text:00401071 add      esp, 8
.text:00401074 xor      eax, eax
.text:00401076 mov      esp, ebp
.text:00401078 pop      ebp
.text:00401079 retn
.text:00401079 _main      endp
```

```
text:00401010
text:00401010 ; ===== S U B R O U T I N E =====
text:00401010
text:00401010 ; Attributes: bp-based frame
text:00401010
text:00401010 Func_A      proc near          ; CODE XREF: _main+1C↓p
text:00401010
text:00401010 var_C      = dword ptr -0Ch
text:00401010 var_8      = dword ptr -8
text:00401010 var_4      = dword ptr -4
text:00401010 arg_0      = dword ptr  8
text:00401010 arg_4      = dword ptr  0Ch
text:00401010
text:00401010 push     ebp
text:00401011 mov      ebp, esp
text:00401013 sub      esp, 0Ch
text:00401016 mov      [ebp+var_C], 0
text:0040101D mov      [ebp+var_4], 5
text:00401024 mov      [ebp+var_8], 6
text:0040102B mov      eax, [ebp+arg_0]
text:0040102E add      eax, [ebp+arg_4]
text:00401031 add      eax, [ebp+var_4]
text:00401034 add      eax, [ebp+var_8]
text:00401037 mov      [ebp+var_C], eax
text:0040103A call     _Func_B
text:0040103F mov      esp, ebp
text:00401041 pop      ebp
text:00401042 retn
text:00401042 _Func_A      endp
```

Q4. 함수 "MAIN" 에서 할당하는 지역변수 공간의 크기는 몇 바이트 입니까?

정답 : 8바이트

증명 : 0x00401053 주소에서 'sub esp, 8' 코드가 실행되는 것을 볼 수 있다. 이를 통해 레지스터에서 변수 공간 8바이트를 확보하는 것을 확인할 수 있다.

```
.text:00401050 _main      proc near      ;
.text:00401050
.text:00401050 var_8      = dword ptr -8
.text:00401050 var_4      = dword ptr -4
.text:00401050 argc      = dword ptr  8
.text:00401050 argv      = dword ptr  0Ch
.text:00401050 envp      = dword ptr  10h
.text:00401050
.text:00401050      push    ebp
.text:00401051      mov     ebp, esp
.text:00401053      sub     esp, 8
.text:00401056      mov     [ebp+var_8], 3
.text:0040105D      mov     [ebp+var_4], 4
.text:00401064      mov     eax, [ebp+var_4]
.text:00401067      push    eax
.text:00401068      mov     ecx, [ebp+var_8]
.text:0040106B      push    ecx
.text:0040106C      call    _Func_A
.text:00401071      add     esp, 8
.text:00401074      xor     eax, eax
.text:00401076      mov     esp, ebp
.text:00401078      pop     ebp
.text:00401079      retn
.text:00401079 _main      endp
```

Q5. 함수 "MAIN"의 지역변수 공간을 사용하는 변수가 정수(Integer, 사이즈 4바이트)일 경우 몇 개의 지역변수가 할당되었다고 추정되나요? 그리고 그 이유는 무엇입니까?

정답 : 2개의 지역변수 할당

설명 : 'sub esp, 8'을 통해 8바이트의 지역변수 공간이 확보되었고, 이 공간에는 4바이트 변수 총 두 개가 할당되었다. 아래 사진에서 0x00401056주소의 'mov [ebp+var_8], 3'과 0x0040105D주소의 'mov [ebp+var_4], 4'코드를 통해 정수형 지역 변수 3과 4가 할당된 것을 확인할 수 있다.

```
.text:00401050 _main      proc near      ;
.text:00401050
.text:00401050 var_8      = dword ptr -8
.text:00401050 var_4      = dword ptr -4
.text:00401050 argc      = dword ptr  8
.text:00401050 argv      = dword ptr  0Ch
.text:00401050 envp      = dword ptr  10h
.text:00401050
.text:00401050      push    ebp
.text:00401051      mov     ebp, esp
.text:00401053      sub     esp, 8
.text:00401056      mov     [ebp+var_8], 3
.text:0040105D      mov     [ebp+var_4], 4
.text:00401064      mov     eax, [ebp+var_4]
.text:00401067      push    eax
.text:00401068      mov     ecx, [ebp+var_8]
.text:0040106B      push    ecx
.text:0040106C      call    _Func_A
.text:00401071      add     esp, 8
.text:00401074      xor     eax, eax
.text:00401076      mov     esp, ebp
.text:00401078      pop     ebp
.text:00401079      retn
.text:00401079 _main      endp
```

Q6. 함수 "Func_A" 가 종료된 이후 실행되는 코드의 주소(리턴 주소)는 무엇입니까?

정답 : 0x00401071

증명 : CALL은 PUSH EIP와 JMP를 합친 개념이므로, JMP가 실행되기 전에 다음에 실행될 주소인 0x00401071을 스택에 PUSH하고, 그 다음에 "Func_A"로 JMP를 한다. 그러므로 "Func_A"의 retn을 만나면 "call func_A"의 다음 줄인 0x00401071 add esp, 8로 리턴된다.

```
.text:00401050 _main      proc near      ;
.text:00401050
.text:00401050 var_8      = dword ptr -8
.text:00401050 var_4      = dword ptr -4
.text:00401050 argc      = dword ptr  8
.text:00401050 argv      = dword ptr 0Ch
.text:00401050 envp      = dword ptr 10h
.text:00401050
.text:00401050          push    ebp
.text:00401051          mov     ebp, esp
.text:00401053          sub     esp, 8
.text:00401056          mov     [ebp+var_8], 3
.text:0040105D          mov     [ebp+var_4], 4
.text:00401064          mov     eax, [ebp+var_4]
.text:00401067          push    eax
.text:00401068          mov     ecx, [ebp+var_8]
.text:0040106B          push    ecx
.text:0040106C          call   _Func_A
.text:00401071          add     esp, 8
.text:00401074          xor     eax, eax
.text:00401076          mov     esp, ebp
.text:00401078          pop     ebp
.text:00401079          retn
.text:00401079 _main      endp
```

< 디버깅 영역 (Immunity Debugger, xDbg) >

Q7. 디버를 이용하여 함수 "MAIN"의 "0x00401053 SUB ESP, 8" 코드까지 실행해 주세요.

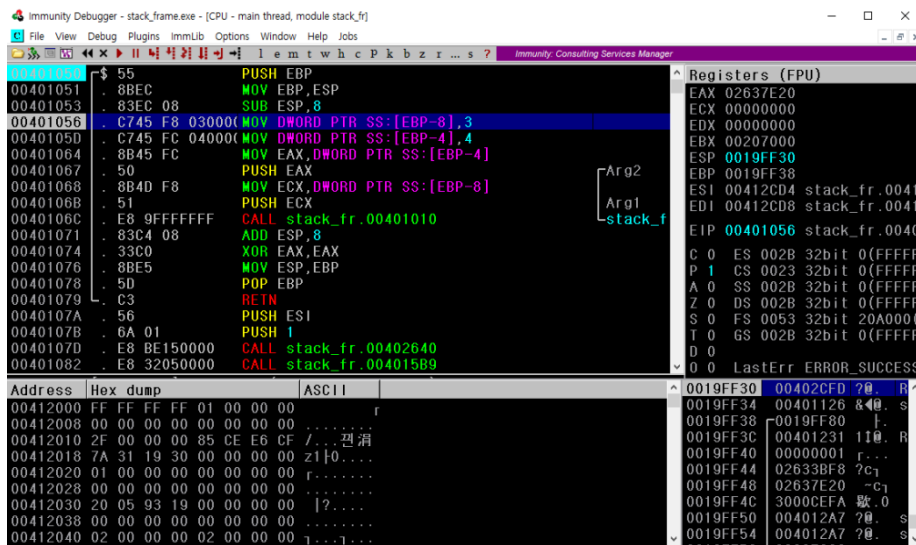
해당 코드까지 실행된 상태를 증명해 주세요. 이 때 EBP 레지스터는 Func_A 함수의 SFP(Saved Frame Pointer)를 가리킵니다.

1)EBP 레지스터의 값, 2)EBP 레지스터가 가리키는 SFP 값은 무엇입니까?

정답 :

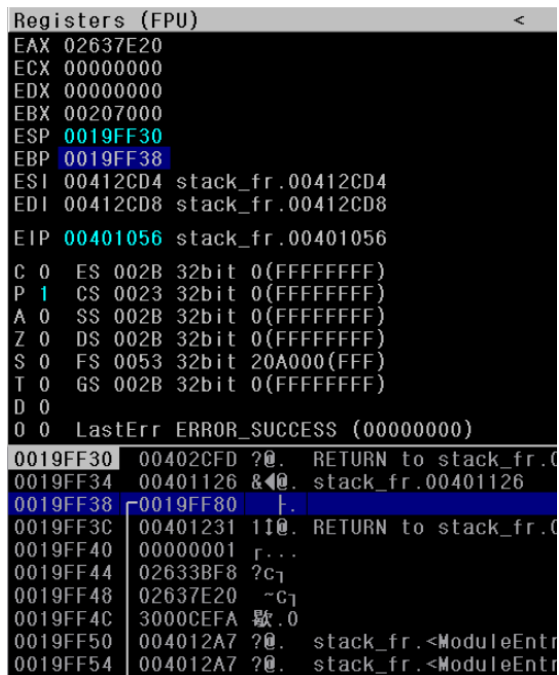
1)EBP 레지스터의 값 : 0x0019FF38

2)EBP 레지스터가 가리키는 SFP 값 : 0x0019FF80



F7키를 눌러 "0x00401053 SUB ESP, 8" 코드까지 실행

증명 : Register창에서 EBP 레지스터 값이 0019FF38인 것을 확인할 수 있고, 그 아래 스택 창에서 선택된 부분 보면, EBP 레지스터가 갖고 있는 주소 값인 0019FF38이 SFP 값인 0019FF80을 가리키고 있다.



Q8. 디버를 이용하여 함수 "Func_A"의 "0x00401041 POP EBP" 코드까지 실행해 주세요.

해당 코드까지 실행된 상태를 증명해 주세요. 이제 해당 함수는 "RETN" 코드를 실행하기 직 전 상태이며 종료되기 전입니다. 현재 스택의 최상단에 있는 값은 "어떤의미"를 가집니까?

정답 : 현재 스택의 최상단에 있는 값은 "Func_A" 실행이 끝나면 돌아갈 주소이다. "Func_A"를 호출할 때 PUSH했던 EIP의 값이 스택 맨 위에 있고, "Func_A"가 끝나서

"RETN"을 만나면 POP EIP를 통해 다음에 실행할 주소가 된다.

Immunity Debugger - stack_frame.exe - [CPU - main thread, module stack_fr]

Assembly window:

```

0040100E C3 RETN
0040100F CC INT3
00401010 55 PUSH EBP
00401011 8BEC MOV EBP,ESP
00401013 83EC OC SUB ESP,OC
00401016 C745 F4 00000 MOV DWORD PTR SS:[EBP-C],0
0040101D C745 FC 05000 MOV DWORD PTR SS:[EBP-4],5
00401024 C745 F8 06000 MOV DWORD PTR SS:[EBP-8],6
0040102B 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
0040102E 0345 OC ADD EAX,DWORD PTR SS:[EBP+C]
00401031 0345 FC ADD EAX,DWORD PTR SS:[EBP-4]
00401034 0345 F8 ADD EAX,DWORD PTR SS:[EBP-8]
00401037 8945 F4 MOV DWORD PTR SS:[EBP-C],EAX
0040103A E8 C1FFFFFF CALL stack_fr.00401000
0040103F 8BE5 MOV ESP,EBP
00401041 5D POP EBP
00401042 C3 RETN
00401043 CC INT3
00401044 CC INT3

```

Registers (FPU):

```

EAX 00000012
ECX 00000003
EDX 00000000
EBX 00207000
ESP 0019FF24
EBP 0019FF38
ESI 00412CD4 stack_fr.00412CD4
EDI 00412CD8 stack_fr.00412CD8
EIP 00401042 stack_fr.00401042

```

Stack dump:

| Address | Hex dump | ASCII |
|----------|-------------------------|----------|
| 00412000 | FF FF FF FF 01 00 00 00 | r |
| 00412008 | 00 00 00 00 00 00 00 00 | |
| 00412010 | 2F 00 00 00 85 CE E6 CF | /...권 |
| 00412018 | 7A 31 19 30 00 00 00 00 | z10... |
| 00412020 | 01 00 00 00 00 00 00 00 | r... |
| 00412028 | 00 00 00 00 00 00 00 00 | |
| 00412030 | 20 05 93 19 00 00 00 00 | ?... |
| 00412038 | 00 00 00 00 00 00 00 00 | |
| 00412040 | 02 00 00 00 02 00 00 00 | 1...1... |

F7을 눌러 RETN을 실행하기 전

Immunity Debugger - stack_frame.exe - [CPU - main thread, module stack_fr]

Assembly window:

```

0040100E C3 RETN
0040100F CC INT3
00401010 55 PUSH EBP
00401011 8BEC MOV EBP,ESP
00401013 83EC OC SUB ESP,OC
00401016 C745 F4 00000 MOV DWORD PTR SS:[EBP-C],0
0040101D C745 FC 05000 MOV DWORD PTR SS:[EBP-4],5
00401024 C745 F8 06000 MOV DWORD PTR SS:[EBP-8],6
0040102B 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
0040102E 0345 OC ADD EAX,DWORD PTR SS:[EBP+C]
00401031 0345 FC ADD EAX,DWORD PTR SS:[EBP-4]
00401034 0345 F8 ADD EAX,DWORD PTR SS:[EBP-8]
00401037 8945 F4 MOV DWORD PTR SS:[EBP-C],EAX
0040103A E8 C1FFFFFF CALL stack_fr.00401000
0040103F 8BE5 MOV ESP,EBP
00401041 5D POP EBP
00401042 C3 RETN
00401043 CC INT3
00401044 CC INT3

```

Registers (FPU):

```

EAX 00000012
ECX 00000003
EDX 00000000
EBX 00207000
ESP 0019FF24
EBP 0019FF38
ESI 00412CD4 stack_fr.00412CD4
EDI 00412CD8 stack_fr.00412CD8
EIP 00401042 stack_fr.00401042

```

Stack dump:

| Address | Hex dump | ASCII |
|----------|-------------------------|----------|
| 00412000 | FF FF FF FF 01 00 00 00 | r |
| 00412008 | 00 00 00 00 00 00 00 00 | |
| 00412010 | 2F 00 00 00 85 CE E6 CF | /...권 |
| 00412018 | 7A 31 19 30 00 00 00 00 | z10... |
| 00412020 | 01 00 00 00 00 00 00 00 | r... |
| 00412028 | 00 00 00 00 00 00 00 00 | |
| 00412030 | 20 05 93 19 00 00 00 00 | ?... |
| 00412038 | 00 00 00 00 00 00 00 00 | |
| 00412040 | 02 00 00 00 02 00 00 00 | 1...1... |

F7을 눌러 RETN을 실행 한 후, 함수 "Func_A"가 종료된 후 스택 최상단에

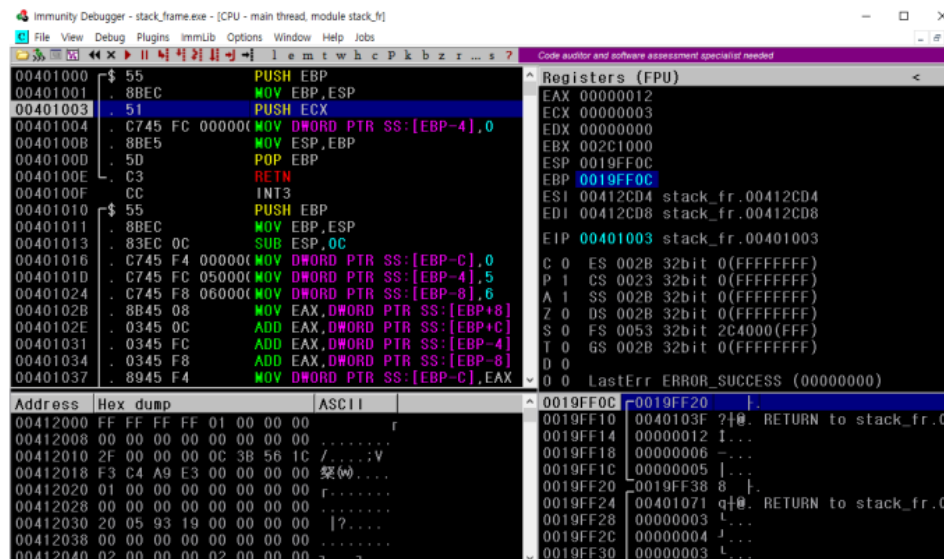
있던 값인 0x00401071로 리턴됨.

Q9. 디버거를 이용하여 함수 "Func_B"의 "0x00401001 MOV EBP, ESP" 코드까지 실행해 주세요. 해당 코드까지 실행된 상태를 증명해 주세요. 해당 함수의 Stack Frame이 빌드된 상태입니다. 이제 "PUSH ECX" 코드가 실행되기 직전입니다. 현재 ESP 레지스터와 EBP 레지스터는 동일한 값을 가지고 있을 것입니다. 두 레지스터는 현재 SFP를 포인팅 하고 있습니다. SFP 값(EBP 혹은 ESP 레지스터)은 무엇입니까?

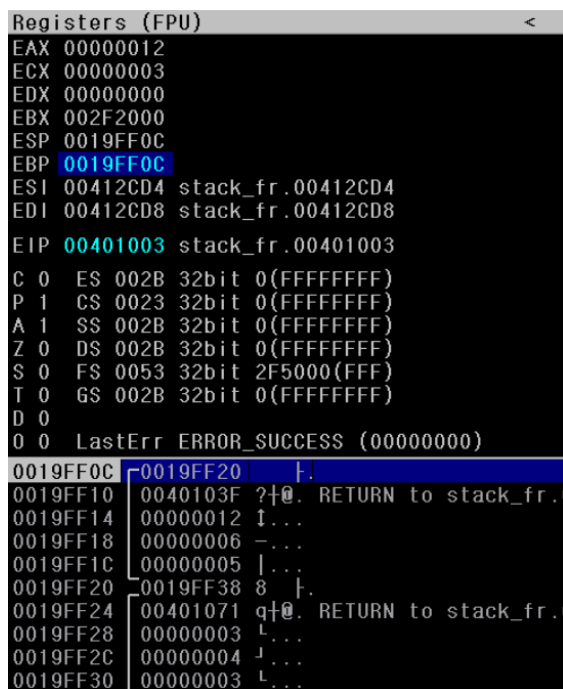
정답 : EBP 레지스터와 ESP 레지스터가 포인팅하고 있는 SFP 값은 0x0019FF20이다.

증명 : Register창에서 보면 ESP 레지스터랑 EBP 레지스터는 0019FF0C로 똑같은 값을 가지

고 있다. 스택창에서 선택한 줄을 보면, 0019FF0C 주소가 가리키는 SFP 값은 0019FF20이라는 것을 확인 할 수 있다.



"0x00401001 MOV EBP, ESP" 코드까지 실행



Q10. 디버거를 이용하여 함수 "Func_A"의 "0x00401037 [EBP+var_C], EAX" 코드까지 실행 해 주세요. 해당 코드까지 실행된 상태를 증명해 주세요. 이제 함수 "Func_B"가 호출되기 직 전 상태일 것입니다. "[EBP+var_C]"는 해당 함수의 지역변수 영역입니다. 스택을 확인하여 어떠한 값이 기록되는지 증명하세요.

정답 : 0x00000012값이 기록된다.

증명 : 0x0019FF14가 가리키는 값은 0x00000012가 되고, 스택 영역에서 또한 0019FF14 옆

에 00000012가 기록된 것을 확인 할 수 있다.

The screenshot shows the Immunity Debugger interface with the following components:

- Assembly View:** Displays assembly instructions for `stack_frame.exe`. The instruction at address `0040103A` is highlighted: `CALL stack_fr.00401000`. Other instructions include `PUSH EBP`, `MOV EBP,ESP`, `SUB ESP,0C`, and several `MOV` and `ADD` instructions involving the stack and `EAX`.
- Registers (FPU):** Shows the state of CPU registers. `EAX` is `00000012`. Other registers like `ECX`, `EDX`, `EBX`, `ESP`, `ESI`, and `EDI` are also listed with their values.
- Memory Dump:** Shows a hex dump of memory starting at address `00412000`. The first few bytes are `FF FF FF FF 01 00 00 00`, which correspond to the ASCII string `.....r`.