

가장 먼저 디버거에서 main 함수를 찾고, main 함수에서 check_password() 호출 부분을 찾는다.

```

004019A7 55      PUSH EBP
004019A8 89E5    MOV EBP,ESP
004019AA 83E4 F0 AND ESP,FFFFFFF0
004019AD 83EC 10 SUB ESP,10
004019B0 E8 5B060000 CALL 3.00401910
004019B5 E8 A6FFFFFF CALL 3.00401960
004019BA 85C0    TEST EAX,EAX
004019BC 0F95C0 SETNE AL
004019BF 84C0    TEST AL,AL
004019C1 74 26   JE SHORT 3.004019E9
004019C3 C70424 0B5040 MOV DWORD PTR SS:[ESP],3.0040500B
004019CA E8 012B0000 CALL <JMP.>3.0040500B
004019CF C70424 275040 MOV DWORD PTR SS:[ESP],3.00405027
004019D6 E8 F52A0000 CALL <JMP.>3.0040500B
004019DB C70424 0B5040 MOV DWORD PTR SS:[ESP],3.0040500B
004019E2 E8 E92A0000 CALL <JMP.>3.0040500B
004019E5 EB 13    JMP SHORT 3.004019F5
004019E8 > C70424 495040 MOV DWORD PTR SS:[ESP],3.00405043
004019F0 E8 DB2A0000 CALL <JMP.>3.00405043
004019F5 > E8 00000000 MOV EAX,0
004019FA C9      LEAVE
004019FB C3      RETN

```

함수 프로로그 부분부터 check_password() 함수인 것을 확인할 수 있다.

```

00401361 55          PUSH EBP
00401363 89E5       MOV EBP,ESP
00401363 83EC 48    SUB ESP,48
00401366 C745 F4 000001 MOV DWORD PTR SS:[EBP-C],0
0040136D C70424 00504001 MOV DWORD PTR SS:[ESP],3.00405000
00401374 E8 572B0000 CALL <JMP.0nsvert.printf>
00401379 8D45 D6    LEA EAX,DWORD PTR SS:[EBP-2A]
0040137C 890424     MOV DWORD PTR SS:[ESP],EAX
0040137F E8 742B0000 CALL <JMP.0nsvert.gets>
00401384 C74424 04 00410001 MOV DWORD PTR SS:[ESP+4],3.00404000
0040138C 8D45 D6    LEA EAX,DWORD PTR SS:[EBP-2A]
00401390 890424     MOV DWORD PTR SS:[ESP],EAX
00401392 E8 212B0000 CALL <JMP.0nsvert.stromp>
00401397 85C0       TEST EAX,EAX
00401399 75 07      JNC SHORT 3.004013A2
0040139B C745 F4 010001 MOV DWORD PTR SS:[EBP-C],1
004013A2 > 8B45 F4    MOV EAX,DWORD PTR SS:[EBP-C]
004013A5 C9         LEAVE
004013A6 C3         RETN

```

get()을 통해 사용자에게 password 입력받는 부분에 buffer[] 크기인 30보다 긴 31글자를 입력하였다.

```

. 890424 MOV DWORD PTR SS:[ESP],EAX
. E8 742B0000 CALL <JMP.&msvcrt.gets>
. C74424 04 0041 MOV DWORD PTR SS:[ESP+4],3.00404000 ASCII "stack overflow"
. 8045 D6
. 890424
. E8 212B
. 85C0
. 75 07
. C745 F4
> 8B45 F4
. C9

```

내가 입력한 값이 저장된 곳을 스택 창에서 확인 후, dump 창에서 해당 주소로 가봤다. 가서 저장된 위치의 주소를 잘 기억해둔다. (0022FEE0 ~ 0022FF15)

```
0022FEE0 0022FEFE ■■". ASCII "1234567890123456789012345678909"
0022FEE4 00000008 0...
0022FEE8 777F118E 240w RETURN to nsvert.777F118E from nsvert.777EF40B
0022FEEC 777F1162 b40w RETURN to nsvert.777F1162 from nsvert.777E987B
0022FEF0 AE212F34 4/?"
0022FEF4 00000000 ....

0022FEF6 00 00 00 00 00 00 00 00 .....
0022FEFE 31 32 33 34 35 36 37 38 12345678
0022FF06 39 30 31 32 33 34 35 36 90123456
0022FF0E 37 38 39 30 31 32 33 34 78901234
0022FF16 35 36 37 38 39 30 39 00 5678909.
0022FF1E 00 00 C4 5B 84 77 70 19 ..-[swp+
0022FF26 40 00 48 FF 22 00 BA 13 0.H ".!!!
0022FF2F 40 00 70 19 40 00 80 00 0.n!@.C.
```

strcmp() 전 00404000에 있는 값을 [ESP+4]로 MOV 해오는 것을 알 수 있다.

이는 saved_password인 것을 알 수 있다. 주석으로 stack overflow라는 것을 알 수 있지만 덤프에서 해당 위치의 값을 확인해 보겠다.



22FEE4의 위치에 saved_password 값이 저장되어 있는 것을 확인할 수 있다.

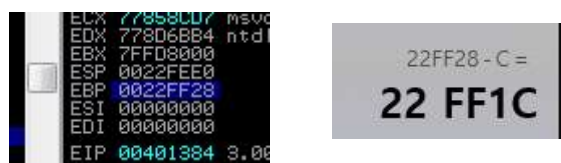


따라서 위에서 확인한 saved_password와 내가 입력한 값이 다르기 때문에 strcmp()의 결과 password가 일치하지 않아 SS[EBP-C]를 1로 mov 해주는 명령을 실행하지 않고 건너뛰게 된다.



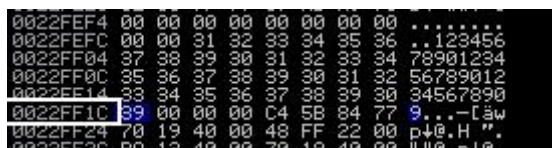
그리고 흐름상 [EBP-C]가 flag가 저장되는 위치인 것을 알 수 있다.

따라서 해당 위치를 계산해 보겠다.



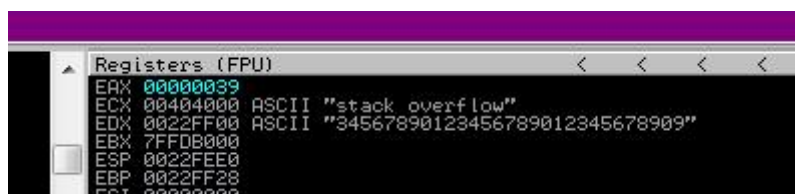
EBP 값인 0022FF28에서 C를 빼면 22FF1C인 것을 알 수 있다.

덤프 창에서 22FF1C 주소로 가보겠다.



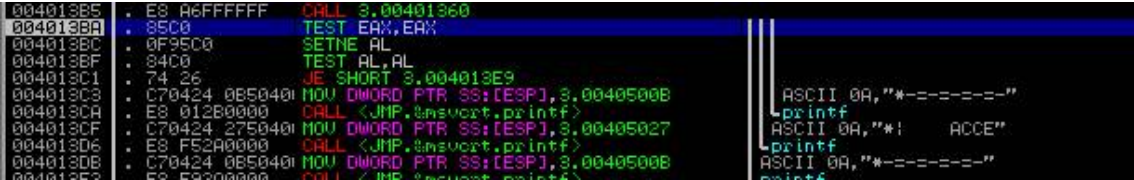
해당 위치를 확인해 보니 아까 내가 입력한 값 바로 밑에 저장된 것을 알 수 있다.

buffer의 크기가 30인데 나는 31글자를 입력해서 buffer 바로 밑인 flag가 저장되는 곳에 9가 저장된 것을 알 수 있다. 즉 flag는 9로 리턴 되게 된다.



해당 서브루틴 종료 후 다시 main 함수로 돌아오면 eax 값이 39인 상태에서

test eax, eax를 하게 된다. 그 결과 eax 값은 0이 아니라 ACCESS ALLOWED를 출력하고 끝나게 된다.



오버플로우 공격 성공 사진

