

## 실습과제 1

exe 파일을 디버거에서 열고 한줄씩 실행을 해보았다.

```
00401200 $ 83EC 1C SUB ESP,1C
00401203 . C70424 01000001 MOV DWORD PTR SS:[ESP],1
0040120A . FF15 8C814000 CALL DWORD PTR DS:[<&nsvcrtdll._set_app_type>] nsvcrtdll._set_app_type
004012E0 . E8 BBFEFFFF CALL plus.00401180
004012E5 . 8DB426 00000001 LEA ESI,DWORD PTR DS:[ESI]
004012EC . 8D7426 00 LEA ESI,DWORD PTR DS:[ESI]
```

먼저 main 함수를 찾기 위해 가장 먼저 보이는 함수에 f7을 이용하여 들어가봤다.

```
00401226 . 81 04704000 MOV EAX,DWORD PTR DS:[704000]
0040122B . 890424 MOV DWORD PTR SS:[ESP],EAX
0040122E . E8 4A010000 CALL plus.00401370
00401233 . 89C3 MOV EBX,EAX
00401235 . E8 FE2C0000 CALL <JMP.&nsvcrtdll._cexit> nsvcrtdll._cexit
0040123A . 891C24 MOV DWORD PTR SS:[ESP],EBX
0040123D . E8 D20E0000 CALL <JMP.&KERNEL32.ExitProcess> ExitProcess
00401242 . 8B4D 00000000 MOV EBX,DWORD PTR DS:[EBP+0]
```

들어가보니 exitprocess라는 주석이 보이고, 저번 예제처럼 exitprocess 윗 부분에 있는 함수가 main 함수 아닐까? 하고 바로 위에 보이는 함수에 f7을 통해 들어가봤다.

```
00401370 $ 55 PUSH EBP
0040137E . 89E5 MOV EBP,ESP
00401380 . 83E4 F0 AND ESP,FFFFFFF0
00401383 . 83EC 20 SUB ESP,20
00401386 . E8 85060000 CALL plus.00401A10
0040138B . C74424 1C 0A01 MOV DWORD PTR SS:[ESP+1C],0A
00401393 . C74424 18 0601 MOV DWORD PTR SS:[ESP+18],6
0040139B . C70424 00504001 MOV DWORD PTR SS:[ESP],plus.00405000 ASCII "Start!!"
004013A2 . E8 212B0000 CALL <JMP.&nsvcrtdll.puts> puts
004013A7 . 8B4424 18 MOV EAX,DWORD PTR SS:[ESP+18]
```

그랬더니 다음과 같이 main 함수를 찾을 수 있었다.

이 부분에선 함수의 시작을 알리는 push ebp, mov ebp, esp 후에 start!!를 출력해주는 것 같다.

```
004013A7 . 8B4424 18 MOV EAX,DWORD PTR SS:[ESP+18]
004013AB . 894424 04 MOV DWORD PTR SS:[ESP+4],EAX
004013AF . 8B4424 1C MOV EAX,DWORD PTR SS:[ESP+1C]
004013B3 . 890424 MOV DWORD PTR SS:[ESP],EAX
004013B6 . E8 A5FFFFF CALL plus.00401360
004013BB . 8B4424 18 MOV EAX,DWORD PTR SS:[ESP+18]
```

이 부분에서는 6과 10이라는 값을 eax로 가져와 스택에 저장하고,

Call plus.00401360 부분에서 서브루틴을 호출하는 것을 알 수 있다.

```

00401360 | $ 55      PUSH EBP
00401361 | . 89E5     MOV EBP,ESP
00401363 | . 83EC 10   SUB ESP,10
00401366 | . C745 FC 000001 MOV DWORD PTR SS:[EBP-4],0
0040136D | . 8B55 08    MOV EDI,DWORD PTR SS:[EBP+8]
00401370 | . 8B45 DC    MOV EAX,DWORD PTR SS:[EBP+C]
00401373 | . 0100      ADD EAX,EDI
00401375 | . 8945 FC    MOV DWORD PTR SS:[EBP-4],EAX
00401378 | . 8B45 FC    MOV EAX,DWORD PTR SS:[EBP-4]
0040137B | . C9        LEAVE
0040137C | . C3        RETN
0040137D | $ 55      PUSH EBP

```

서브루틴 부분을 보면 10과 6을 순서대로 edx, eax 레지스터로 가져온 후에

둘을 add 하여 eax에 저장한다는 것을 알 수 있다. 그리고 그 결과를 스택에 저장하고 서브루틴을 마친다.

서브루틴을 마칠 때 retn 부분에서 eip에 서브루틴 call 했던 라인의 바로 밑에 주소가 저장되어 있는 것을 확인했다.

```

004013A7 | . 8B4424 18  MOV EAX,DWORD PTR SS:[ESP+18]
004013AB | . 894424 04  MOV DWORD PTR SS:[ESP+4],EAX
004013AF | . 8B4424 1C  MOV EAX,DWORD PTR SS:[ESP+1C]
004013B3 | . 890424     MOV DWORD PTR SS:[ESP],EAX
004013B6 | . E8 B5FFFFFF CALL plus.00401360
004013BB | . 894424 14  MOV DWORD PTR SS:[ESP+14],EAX
004013BF | . 837C24 14 0A CMP DWORD PTR SS:[ESP+14],0A
004013C4 | . 7E 16      JLE SHORT plus.004013DC
004013C6 | . 8B4424 14  MOV EAX,DWORD PTR SS:[ESP+14]
004013CA | . 894424 04  MOV DWORD PTR SS:[ESP+4],EAX
004013CE | . C70424 085040 MOV DWORD PTR SS:[ESP],plus.00405008
004013D5 | . E8 F62A0000 CALL <JMP.&msvcrt.printf>
004013DA | . EB 14      JMP SHORT plus.004013F0
004013DC | . 8B4424 14  MOV EAX,DWORD PTR SS:[ESP+14]
004013E0 | . 894424 04  MOV DWORD PTR SS:[ESP+4],EAX
004013E4 | . C70424 1C5040 MOV DWORD PTR SS:[ESP],plus.0040501C
004013EB | . E8 E02A0000 CALL <JMP.&msvcrt.printf>
004013F0 | . B8 00000000 MOV EAX,0
004013F5 | . C9        LEAVE
004013F6 | . C3        RETN

```

ASCII "GOOD!! RESULT :Zd B" printf

ASCII "BAD.. RESULT :Zd B" printf

서브루틴의 결과가 eax에 저장되었었는데 이 결과를 10과 비교(cmp)하고 작거나 같다면 0040230c 부분(bad 출력)으로 점프하고, 크다면 바로 밑에 004013ca 부분(good!! 출력)이 실행되는 것을 확인했다.

위의 부분을 보고 if문은 if문 조건이 cmp로 비교되고 jmp를 이용해 cmp결과에 따라 분기하게 하여 if문이 실행되는 것을 알았다.

## 실습과제 2

Exe 파일을 생성 후 디버거를 통해 열었다.

```

00401200 $ 83EC 1C SUB ESP,1C
00401203 . C70424 010000 MOV DHORD PTR SS:[ESP],1
0040120A . FF15 A0814000 CALL DHORD PTR DS:[<&nsvcrtdll.__set_app_type nsvcrtdll.__set_app_type
004012E0 . E8 BBFEFFFF CALL hellonor.004011A0
004012E5 . 8DB426 000000 LEA ESI,DWORD PTR DS:[ESI]

```

가장 먼저 보이는 함수를 f7를 통해 내부를 보았다.

```

0040122E . E8 2D010000 CALL hellonor.00401360
00401233 . 89C3 MOV EBX,EAX
00401235 . E8 9E2C0000 CALL <JMP.&nsvcrtdll._cexit> nsvcrtdll._cexit
0040123A . 891C24 MOV DHORD PTR SS:[ESI],EBX
0040123D . E8 820E0000 CALL <JMP.&KERNEL32.ExitProcess> ExitProcess

```

이번 실습도 실습1과 동일하게 exitprocess 위에 있는 call 함수에서 호출하는 부분이 main 함수였다.

```

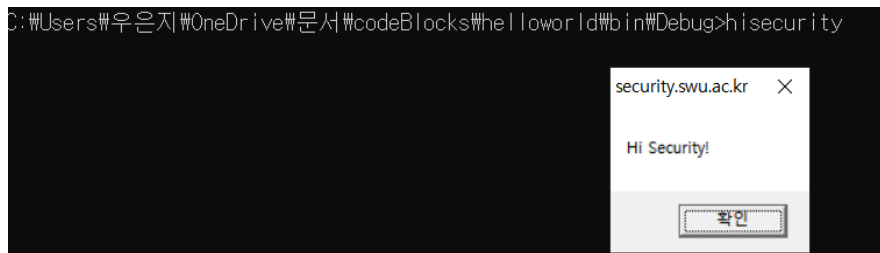
00401360 $ 8D4C24 04 LEA ECX,DWORD PTR SS:[ESP+4]
00401364 . 83E4 F0 AND ESP,FFFFFFF0
00401367 . FF71 FC PUSH DHORD PTR DS:[ECX-4]
0040136A . 55 PUSH EBP
0040136B . 89E5 MOV EBP,ESP
0040136D . 51 PUSH ECX
0040136E . 83EC 14 SUB ESP,14
00401371 . E8 4A060000 CALL hellonor.004019C0
00401376 . C74424 0C 0001 MOV DHORD PTR SS:[ESP+C],0
0040137E . C74424 08 0051 MOV DHORD PTR SS:[ESP+8],hellonor.004051 ASCII "security.swu.ac.kr"
00401386 . C74424 04 1351 MOV DHORD PTR SS:[ESP+4],hellonor.004051 ASCII "Hello World!"
0040138E . C70424 000000 MOV DHORD PTR SS:[ESI],0
00401395 . E8 7E2B0000 CALL <JMP.&USER32.MessageBoxA> MessageBoxA
0040139A . 83EC 10 SUB ESP,10
0040139D . B8 00000000 MOV EAX,0
004013A2 . 8B4D FC MOV ECX,DWORD PTR SS:[EBP-4]

```

Security.swu.ac.kr과 Hello World를 스택에 저장하고 messagebox를 call하는 것을 알 수 있다.

Address	Hex dump	ASCII
00405013	48 65 6C 6C 6F 20 57 6F	Hello Wo
0040501B	72 6C 64 21 00 04 17 40	rl!.
00405023	00 6A 14 40 00 6A 14 40	.j.je.je
0040502B	00 6A 14 40 00 6A 14 40	.j.je.je
00405033	00 6A 14 40 00 6A 14 40	.j.je.je

Hello world가 저장되어 있는 곳에 ctrl+g 키를 통해 와서 문자열을 변경하기 위해 컨트롤+e로 Hi Security로 수정하였다. 그 후 copy to executable 후 save file을 하여 복사본을 저장한 후 그 파일을 실행해본 결과



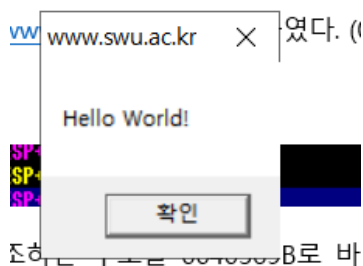
다음과 같이 원하는 결과가 출력되었다.



다음으로 덤프에서 문자열이 저장된 곳 아래에 0000으로 패딩되어 있는 곳에 가서 칸을 선택한 후에 컨트롤+e 키로 security.swu.ac.kr을 [www.swu.ac.kr](http://www.swu.ac.kr)로 변경하였다. (0040569B의 주소에)



다음은 0040137E 부분을 두 번 클릭해 참조하는 주소를 0040569B로 바꿔서 실행해보았다.



다음과 같이 원하는 결과를 얻을 수 있었다.