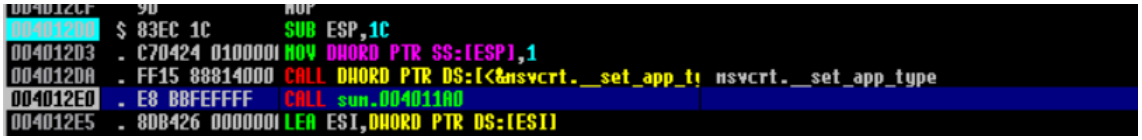
# 3주차 분석 보고서

정보보호학과 우은지

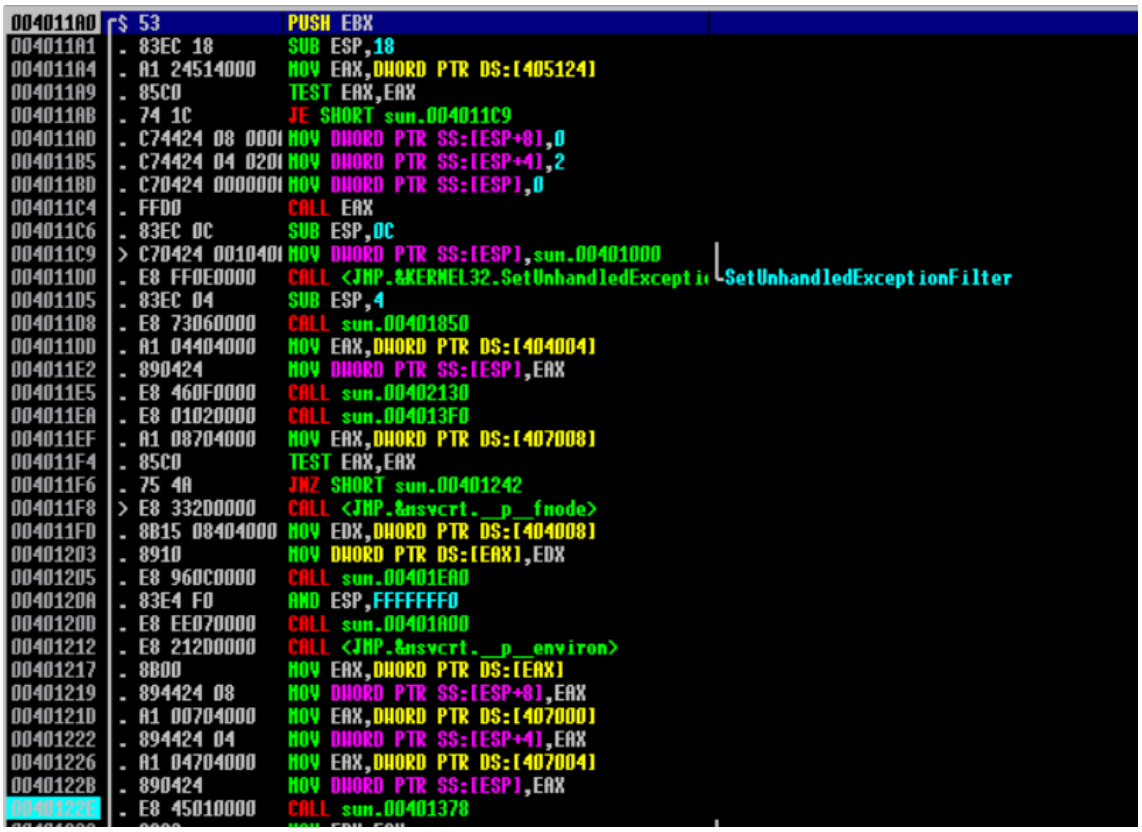
1. main 함수 찾기

디버거에 실행 파일을 넣고 F8로 실행을 해보면



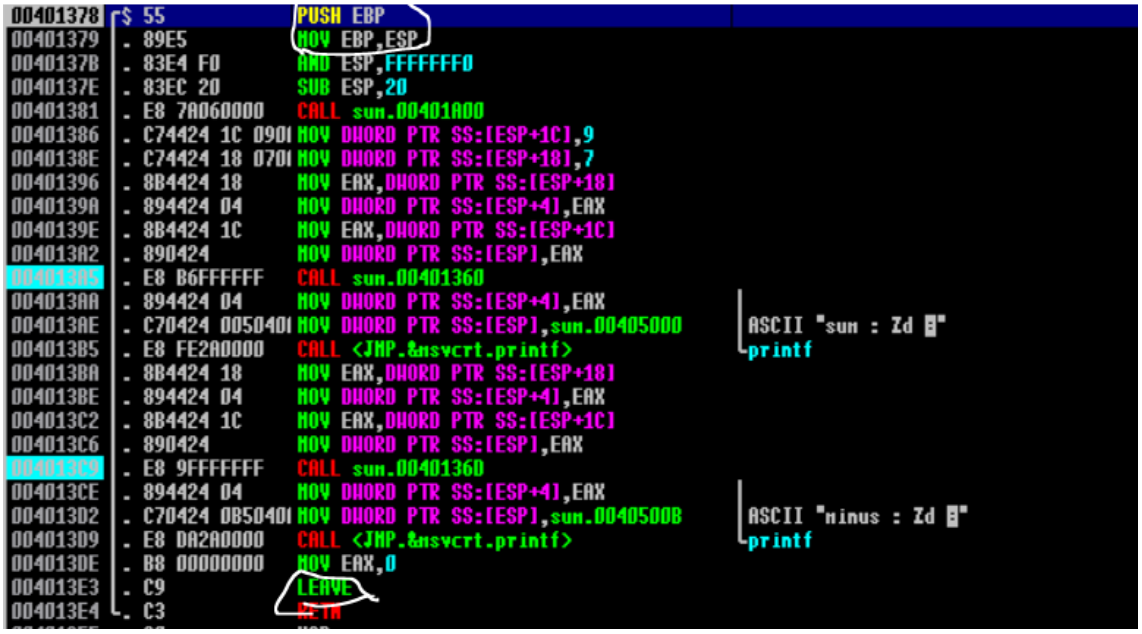

004011A0 함수 호출되는 부분을 실행하면 프로그램이 끝나는 것을 확인할 수 있다.

따라서 저 004011A0 함수를 F7을 이용해서 내부를 보았다.




004011A0 함수의 내부 모습인데 BREAKPOINT 걸려있는 부분을 실행하면 실행 창에 해당 프로그램 결과가 출력되는 것을 확인할 수 있다. 따라서 004011A0 함수가 main 함수인 것을 알 수 있다.

2. 서브함수 찾기

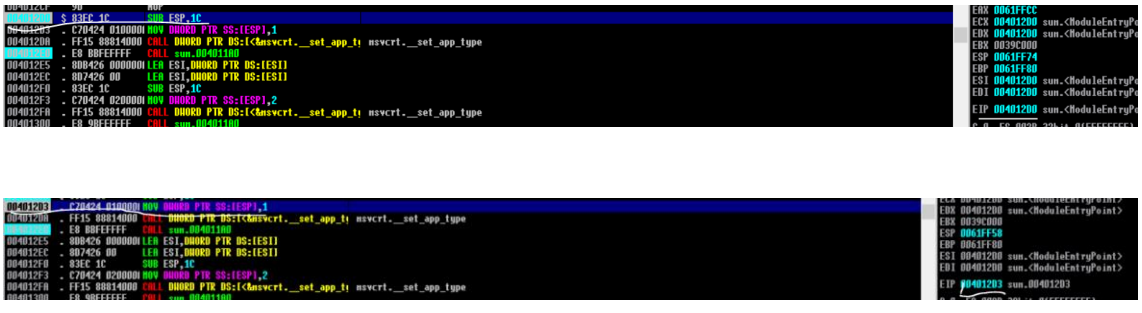



main 함수의 내부이다. call이 되는 부분에서 F7를 이용해 실행하여 보면 BREAKPOINT 걸린 부분이 sum과 minus 함수를 call하고 있는 부분임을 알 수 있다.

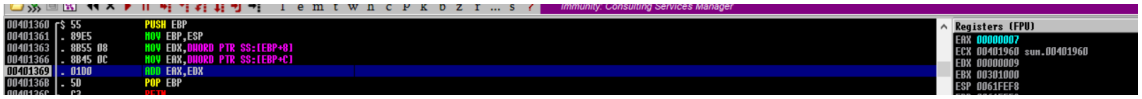
3. 레지스터 변경 확인

EAX: 산술 연산 결과 저장

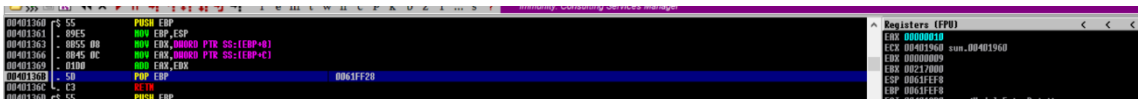
EIP: 다음 명령어 저장



F8을 이용해서 한줄한줄 실행할 때마다, EIP의 값이 다음 명령어로 바뀌는 것을 확인했고,



sum 함수에서 mov를 통해 스택에 있는 값 7을 가져와서 EAX에 저장하는 것을 확인했고,



ADD를 통해 EAX 값인 7과 EDXD에 있던 값인 9를 더한 16을 EAX에 저장한 것을 확인했다. (16진수로 저장되어 10이라고 저장됨)

4. 스택 변경 확인

여기에서 CALL로 004011A0으로 갈 때 스택을 확인해보면

스택에 004011A0 밑에 있는 명령어의 주소가 저장되어있는 것을 알 수 있다.

(004011A0 함수를 끝내고 돌아가려고)

MAIN 함수를 들어가는 부분인데 이때도 MAIN 함수를 들어가서 스택을 처음 확인해보면

다음 명령어 주소인 00401233이 스택에 저장되어 있는 것을 확인할 수 있었다,

이뿐만 아니라 CALL 명령어를 실행하면 바로 밑 명령어의 주소가 스택에 저장되는 것을 확인 할 수 있었다.

다음은 스택 관련 레지스터 EBP, ESP에 대해 서브루틴 호출 전 후를 확인해봤다.


MAIN 함수 속 SUM 함수 호출 직전 EBP, ESP에 저장된 값이다.

SUM 함수를 호출하면 어떻게 변하는지 확인해봤다.


먼저 PUSH EBP를 통해 MAIN 함수의 EBP를 스택에 저장해서 ESP의 값만 변한 것을 알 수 있다.


다음은MOV EBP, ESP를 통해 서브 루틴의 EBP의 값을 만들어주는 것을 확인했다.


POP EBP의 결과 EBP와 ESP의 값이 서브루틴 호출 전과 같아진 것을 확인할 수 있었다.