C 함수 호출 규약

\_\_cdecl

함수를 call한 후 add esp, n 식으로 스택을 정리한다. 함수 밖에서. 이를 통해 파라미터의 개수 또한 파악 가능하다.

\_\_stdcall

함수 내에서 retn n 의 형식으로 스택을 정리한다. win32 api는 stdcall을 사용한다.

\_\_fastcall

fastcall은 파라미터가 2개이하일 경우 인자를 ecx와 edx 레지로 이용. 메모리보다 빠름. 함수 호출 전 edx, ecx에 넣는 모습이 보인다면 fastcall

\_\_thiscall

c++ 클래스에서 주로 사용. 현재 객체의 포인터를 ecx에 전달. 이것이 this. 그래서 멤버 변수는 ecx포인터에 오프셋 및 번지를 더하는 식으로 사용한다. 나머지 방법은 stdcall과 동일.

구조체와 API call

변수가 레지스터로 충당이 안된다면 스택으로 공간을 확보한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

ida를 통해 예제 코드를 짜고 어셈블리 코드를 분석해보았다.

구조체의 크기만큼 스택을 sub하여 공간을 확보한다.

교재에서는 구조체의 크기만큼만 스택을 확보했으나 IDA를 통해 코드를 분석해보니 교재와는 다른 크기만큼 스택을 확보했다.

코드를 분석하며 차이점을 살펴보았다.

교재에서는 push를 통해 파라미터를 집어넣었지만 IDA에선 mov를 통해 esp-n만큼의 위치에 파라미터를 집어넣었다.

이를 통해 IDA에서 보여준 어셈블리 코드는 구조체의 파라미터가 push되는 것까지 미리 계산하여 스택을 확보했음을 알 수 있었다.

그 후 호출되는 코드에서 구조체의 포인터를 가져올 때는 구조체의 크기를 계산한 만큼의 위치를 스택에서 불러오고, 0x44 바이트와 0x 10바이트의 크기를 가진 구조체라는 것을 인지하며 어셈블리 코드를 분석할 시 훨씬 분석이 수월하였다. 또한 함수는 원형이 존재하며 보이는 파라미터가 아니라 추가적인 파라미터를 push할 시 다른 함수를 통해 계산을 하는 것을 생각해야한다.