리버싱 쪽에 집중되서 설명

nt라는 만드는 과정에서 만들어짐

com이라는 파일 포맷은 한계가 있음 64비트를 넘어버리면 코드세그먼트와 데이터세그먼트를 나눔

원래 같은 세그먼트를 가르키고 있는데 off이 동일한 위치에서 움직임

//cs : B36, ds : B36

결국 1메가가 넘는 것을 다른 확장자로 만들고 cs와 ds를 멀리 떨어뜨려놈

dos에서는 1메가 넘는 파일이 거의 없어서 메모리에서 잠깐 잠깐 쓰는 형태에 유형으로 개발됨

이 포맷을 유닉스에 coff 포맷에서 살짝 빌려왔음

확장자는 exe지만 대체로 coff 포맷과 비슷함 (기본개념 흡사, 기본개념을 기반으로해서 PE파일이 만들어짐)

원래 PE가아니라 COM->EXE(DOS)->NE(Win31)->PE(W95) //NE와 PE가 거의 유사 (NE:16비트, PE:32비트) -> PE오 비슷한 LE라고도 불림 (악성코드 분석하다 나올 수 있음)

dos시절

C:\> cd win3.1

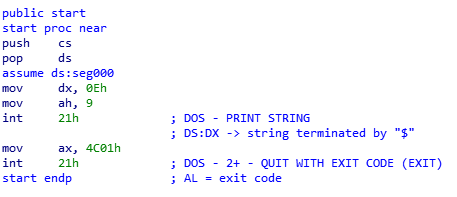
C:\win3.1 > win.com이라는

cale.exe 이런식으로 프로그램 실행

진짜 옛날 dos와 windows 환경이 공존하고 있을 때



1. Mz로 시작되는 일반적신 PE 파일
2. Dos용으로 실행하면



단순히 옛날 유물 하는 일 없음

Notepad가 64비트로 컴파일 됐음에 불구하고 16비트가 헤더위에 붙어있음

dos시절에는 정적라이브러리 지금은 동적라이브러리 사용함

컴파일할 때 메모리는 몇번지에 매핑하는지 (컴파일 옵션으로 설정 가능)

기본 메모장으로 로딩하면 0x400000에 매핑 따라서 2개 프로그램 매핑하면 충돌이 남

첫번째 실행한놈은 400000에 들어올 수 있음 2번째 실행한놈은 위치를 옮겨야함

1000000으로 만약 옮겨서 로딩하면 바로 점프해서 실행할 수 없음 콜이나 메모리 참조들이

꼬여버리고 다 틀어져 버림 만약 메모장이 40만번지 계산기가 100만번지에 매핑했을 때 로딩하면 참조들이 다 하드코딩해서 들어가는데 이렇게 되면 참조가 다 틀어져버림

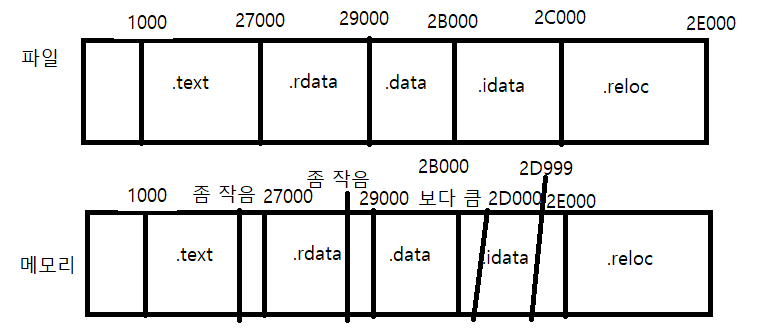
틀어져도 방법이있는데 Win7부터 나온기법으로 재배치를 한다.

PE파일 위에는 왜 dos실행파일이 있을까?

실전으로 가면 말도 안되는 운영체제를 사용하는 것을 볼 수 있음

Ms가 운영체제에 통계를 내는데 네트웤에 물리지 않고 돌리고 있음

아마 그래서? PE파일 구조 위에 dos에서 실행할 수 있는 것이 존재함



메모리와 파일의 위치가 딱 맞지 않은 이유는 섹터에 크기가 정해져 있기 때문이다.

파일은 섹터 단위로 나눠져있어서 딱딱 나눠짐

하지만 메모리에 올라올 때는 기계어만 올라오기 때문에 이상한 숫자가 나옴 (컴파일 된 크기)

뒷 공간은 쓰레기 값이다.

뒷 공간에 바이러스를 붙여서 메모리의 바이러스 코드도 올릴 수 있음

파일은 어쩔수 없게 0x200으로 끊을 수 밖에 없음

// c언어로 코딩할 때 배열에 값을 채우지 않음 (대부분 0이라고 가장 하기 때문에)

컴파일하고 메모리에 올라갈 때 배열의 메모리가 올라가기 때문에 메모리가 커질 수 있음

디버거란?

프로그램이 실행 중에 필요한 부분에서 멈추려고 만듦

버그를 없애려는 목적으로 만들어짐

원래는 리버싱 하려는 목적으로 만ㄷ르어진 것은 아님

개발자의 실수를 고치려고 함

원래 소스코드가 실제 보이는데 우리가 보려는 것은 소스코드가 없기 때문에 소스코드 창을 별도로 보여주지 않음 WinDBG는 ms에서 만든 디버거 이것은 윈도우를 디버거할 수 있음 (커널부터)

사용법이 정말 어렵고 명령어가 매우 복잡해서 작업이 까다로움

명령어는 훨씬더 많음, 윈도우에 내부에 특정위치에 프로세스위치가 어디에 있는지 보려면 WIndbg를 잘 다뤄야함 우리나라에서 Windbg를 잘 분석할줄 알면 고급인력

보통적인 함수 인자 전달방식은 스택, 다른 방법은 레지스터에 인자를 넣어서 보내는 것

직접적으로(어셈)코딩할 때는 복잡하기 때문에 보통 스택 이용

브레이크포인트 : f2로 걸 수 있음

하지만 브레이크포인트를 걸 수 없는 상황이나 브레이크 포인트를 걸 개수가 적을 떄는

하드웨어 브레이크 포인트를 걸 수 있다

Cpu에 디버거 레지스터에 걸 수 있는데 이 중4개에 걸 수 있다.

메모리 브레이크 포인트는 사실 없는 말이 맞는데 디버거 자체가 그냥 이 공간을 보호를 걸어서 보호영역이기 때문에 읽기 쓰기가 안되 에러가 발생하여 프로그램이 멈추면서 프로그램이 디버거에게 제어가 들어온다. 그래서 실행될때는 그 보호된걸 풀어줘서 브레이크 포인트를 건 것 같은 현상을 볼 수 있게 설게함

32비트일땐 Immunity Debugger

64비트일땐 x64 디버거

Immunity Debugger은 python으로 확장할 수 있다는 큰 메리트가 있음

흰색 영역에는 코드를 넣을 수 있음

Imm..log”문자열” : 문자열이 log에 찍힘

Imm.setBreakpoint(주소) : 주소에 브포검

Imm.run : 실행

변수 = imm.readMemory(주소, 바이트) : 해당 주소에 해당 바이트만큼 읽어서 변수에 저장

Imm.writeMemory(주소, 값) : 해당 주소에 값을 덮어 씀

주소에 공백이 들어오는 구간에는 랜덤값이 들어오는 구간이기 때문에 정확히 주소 지정할 수 없다.