**Dreamhack Quiz: x86 Assembly**

Q: end로 점프하면 프로그램이 종료된다고 가정하자. 프로그램이 종료됐을 때, 0x400000 부터 0x400019까지의 데이터를 대응되는 아스키 문자로 변환하면?

|  |
| --- |
| [Register]  rcx = 0  rdx = 0  rsi = 0x400000 |
| [Memory]  0x400000 | 0x67 0x55 0x5c 0x53 0x5f 0x5d 0x55 0x10  0x400008 | 0x44 0x5f 0x10 0x51 0x43 0x43 0x55 0x5d  0x400010 | 0x52 0x5c 0x49 0x10 0x47 0x5f 0x42 0x5c  0x400018 | 0x54 0x11 0x00 0x00 0x00 0x00 0x00 0x00 |
| [code]  1: mov dl, BYTE PTR[rsi+rcx]  2: xor dl, 0x30  3: mov BYTE PTR[rsi+rcx], dl  4: inc rcx  5: cmp rcx, 0x19  6: jg end  7: jmp 1  정답: Welcome to assembly world! |

**[코드 분석]**

mov dl, BYTE PTR [rsi+rcx]:

rcx와 rsi에 더한 주소에서 값의 1byte를 가져와 dl(register)에 저장한다. rcx = 0이고, rsi = 0x400000이므로, 0x400000 주소의 값을 dl에 가져온다.

xor dl, 0x30:

dl에 저장된 값을 0x30과 XOR 연산한다.

mov BYTE PTR [rsi+rcx], dl:

변환된 dl의 값을 다시 원래 메모리 위치에 저장한다.

inc rcx:

rcx 1증가.

cmp rcx, 0x19:

rcx의 값이 0x19(25)인지 비교한다.

jg end:

rcx가 0x19보다 크면 end로 점프하여 프로그램이 종료된다.

jmp 1:

rcx가 0x19보다 크지 않은 경우 다시 1번 명령어로 점프하여 위 과정을 반복한다.

[과정]

이 프로그램은 메모리의 25바이트를 1바이트씩 순차적으로 읽어 XOR 변환 (0x30과 XOR)한 후, 그 결과를 다시 같은 위치에 저장한다. 이후 최종적으로 변환된 데이터를 아스키 문자로 변환한다.

1. 0x67 ^ 0x30 = 0x57 -> 'W'
2. 0x55 ^ 0x30 = 0x65 -> 'e'
3. 0x5c ^ 0x30 = 0x6c -> 'l’
4. 0x53 ^ 0x30 = 0x63 -> 'c'
5. 0x5f ^ 0x30 = 0x6f -> 'o'
6. 0x5d ^ 0x30 = 0x6d -> 'm'
7. 0x55 ^ 0x30 = 0x65 -> 'e'
8. 0x10 ^ 0x30 = 0x20 -> ' '
9. 0x44 ^ 0x30 = 0x74 -> 't'
10. 0x5f ^ 0x30 = 0x6f -> 'o'
11. 0x10 ^ 0x30 = 0x20 -> ' '
12. 0x51 ^ 0x30 = 0x61 -> 'a'
13. 0x43 ^ 0x30 = 0x73 -> 's'
14. 0x43 ^ 0x30 = 0x73 -> 's'
15. 0x55 ^ 0x30 = 0x65 -> 'e'
16. 0x5d ^ 0x30 = 0x6d -> 'm'
17. 0x52 ^ 0x30 = 0x62 -> 'b'
18. 0x5c ^ 0x30 = 0x6c -> 'l'
19. 0x49 ^ 0x30 = 0x79 -> 'y'
20. 0x10 ^ 0x30 = 0x20 -> ' '
21. 0x47 ^ 0x30 = 0x77 -> 'w'
22. 0x5f ^ 0x30 = 0x6f -> 'o'
23. 0x42 ^ 0x30 = 0x72 -> 'r'
24. 0x5c ^ 0x30 = 0x6c -> 'l'
25. 0x54 ^ 0x30 = 0x64 -> 'd'
26. 0x11 ^ 0x30 = 0x21 -> '!'

**리버싱 도구**

**디버깅 도구 (Ollydbg, IDA pro, Immunity Debugger, Windbg)**

파일을 어셈블리어로 디스어셈블해서 코드의 실행 흐름을 추적하고, 실시간 분석에 사용된다. 주로 코드의 로직을 이해하고, 오류를 찾는 데 중점을 둔다.

|  |
| --- |
| IDA Pro: 디스어셈블러와 디버깅 기능을 모두 제공하는 강력한 도구로, 복잡한 바이너리 분석에 적합. |
| 🡪 x64dbg: 사용자 친화적인 인터페이스를 제공하며, 실시간으로 프로그램의 동작을 분석하는 데 유용. |

**정적 분석 도구 (Detours**, **Eceinfo, PEiD**, **PEView, Stud\_PE, LoardPE**, **strings, Bintext**)

바이너리를 실행하지 않고, 파일의 구조와 내부 구성 요소를 분석하는데 사용

|  |
| --- |
| PEView: PE 파일의 구조(예: 헤더, 섹션, 임포트/익스포트 테이블 등)를 분석하는 도구로, PE 파일의 내부를 상세히 파악하는 데 유용. |
| Strings: 바이너리 파일 내부에서 ASCII나 Unicode 문자열을 추출하여, 파일 내에 포함된 텍스트나 API 호출을 분석하는 데 사용. |

**동적 분석 도구-유저레벨 (Process Monitor**, **Process Explorer**, **Wireshark**.)

프로그램이 실행되는 동안 발생하는 다양한 시스템 활동을 모니터링하고 분석하는 데 사용된다. 프로그램이 실제로 어떻게 동작하는지, 어떤 시스템 리소스를 사용하는지 등을 확인하는 데 초점을 맞춘다.

|  |
| --- |
| Process Monitor: 파일 시스템, 레지스트리, 프로세스, 스레드 활동 등을 실시간으로 모니터링할 수 있는 도구로, 시스템에서 발생하는 다양한 이벤트를 추적할 수 있음. |
| Wireshark: 네트워크 트래픽을 캡처하고 분석하는 도구로, 네트워크 상에서 발생하는 모든 패킷을 확인하고 분석하는 데 사용. |

**동적 분석 툴-커널레벨 (GMER**., **PC Hunter**)

시스템의 심층적인 동작을 분석하고, 특히 루트킷이나 기타 고급 악성코드를 탐지하는데 유용하다.

|  |
| --- |
| GMER: 루트킷 탐지에 특화된 도구로, 시스템 콜 훅킹이나 숨겨진 프로세스를 탐지하는 데 매우 효과적. |
| PC Hunter: 심층적인 커널 모드 분석이 필요한 경우 사용되며, 루트킷 탐지에 활용. |

**기타 리버싱 도구(패커/언패커 도구, 디컴파일러, 메모리 분석 도구, 레지스트리 분석 도구 e.t.c…)**

|  |
| --- |
| UPX/Unpacker: 패킹된 파일을 원래의 상태로 복원하여 분석할 수 있는 도구.  Ghidra: 디컴파일러로, 바이너리 코드를 고수준 언어로 변환하는 도구.  Volatility: 메모리 덤프를 분석하여 시스템 메모리에 존재하는 악성코드, 루트킷 등을 분석하는 데 사용되는 도구. |

이 중에서 동적 분석(디스어셈블러 및 디버거), 그리고 정적 분석(PE 파일 분석)은 가장 핵심적인 역할을 한다. 리버싱을 수행할 때는 일반적으로 정적 분석 도구을 통해 파일의 기본적인 구조와 메타데이터를 먼저 파악하고, 이후 동적 분석 도구를 사용해 코드의 동작을 심층적으로 분석한다.

**Helloworld.exe 파일의 정적 분석**

드림핵 리버싱 강의의 helloworld 파일을 사용했다. 🡪 [Helloworld.exe](https://drive.google.com/file/d/1XJObiFQFb_GWM_8zqElq2zUllKro5Wd7/view?usp=sharing)