**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. 비교 결과에 따라 ZF(zero flag 값이 같을 때), SF(sign flag 결과가 음수일 때), CF(carry flag 빌림이 발생할 때) 등이 설정된다. cmp는 두 번째 피연산자를 첫 번째 피연산자에서 빼는 연산을 수행한다.

jg end

jg는 조건부 점프 명령어로 rcx가 0x19보다 크면 end로 점프하여 프로그램이 종료된다. jg 명령어는 “Jump if Greater”을 의미하며 ZF가 설정되지 않고 SF와 OF(overflow flag)가 같을 때 점프한다. 즉 두 값 중 첫 번째 값이 두 번째 값보다 클 때 만족함을 의미한다.

**rcx > 0x19: ZF=0, SF=0(rcx-0x19는 양수이므로), OF=0**

rcx < 0x19: ZF=0, SF=1(rcx-0x19는 음수이므로), OF=0

rcx > 0x19: ZF=1(결과가 0이므로), SF=0, OF=0

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(PEView 사용)**

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

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

그림 1 메인 화면

**‘mz’ 서명**

텍스트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

**PE 헤더의 시작 위치(offset)**

그림 2 DOS Header

🡪 DOS Header(IMAGE\_DOS\_HEADER): 옛날 MS-DOS와의 호환성을 위해 존재하며, PE파일임을 나타내는 ‘MZ’ 서명(0x5A4D)과 PE 헤더 위치를 가리키는 ‘e\_lfanew’ 필드를 확인할 수 있다. ‘MZ’는 DOS 실행 파일의 시그니처로, PE파일임을 나타내는 가장 기본적인 표시다. 이 값이 ‘5A4D(ASCII->’MZ’)’가 아닌 경우 PE파일이 아니거나 손상되었을 가능성이 있다. Offset to New EXE Header(0x00000100)은 PE 헤더의 시작 위치를 가리키는 오프셋으로 이 값에 따라 PE 파일의 ‘실제’헤더(PE 헤더)로 이동할 수 있다. 나머지 필드들은 PE 파일에서 사용되지 않거나, MS-DOS와의 호환성을 위해 존재하는 필드들로 참고만 하면된다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

그림 3 DOS stub program

🡪 이 섹션은 PE 파일의 초기 부분에 포함된 작은 프로그램으로 주로 ‘This program cannot be run in DOS mode’라는 메시지를 포함하고 있다. 이 부분은 PE 파일을 16비트 MS-DOS 환경에서 실행할 경우, "This program cannot be run in DOS mode"라는 메시지를 출력하는 역할을 하는데 현대의 32비트나 64비트 Windows 환경에서 이 부분은 대부분 무시되며, PE 파일 분석의 대부분에서는 신경 쓰지 않아도 된다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

그림 4 IMAGE\_NT\_Headers

🡪 PE Header(IMAGE\_NT\_HEADERS): 이 섹션은 PE파일의 핵심 정보가 있는 곳으로, PE파일의 구조와 동작 방식을 정의하는 데 매우 중요하다. 세 가지 주요 구성 요소로 나뉜다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

그림 5 IMAGE\_NT\_Headers [Signature]

Signature: PE 파일의 진짜 시작을 나타내는 ‘PE\0\0’ 서명(0x00004550)을 포함하고 있다. PE 파일임을 명확히 하는 시그니처 값으로 이 값이 올바르지 않다면 파일이 손상되거나 PE 파일이 아닐 수 있다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

그림 6 IMAGE\_NT\_Headers [File Header]

File Header(IMAGE\_FILE\_HEADER): 아키텍처, 섹션 수, 빌드 시간 등의 기본 정보를 얻을 수 있다.

|  |  |
| --- | --- |
| Machine(0x8664) | 대상 머신 아키텍쳐를 나타내는 필드로 0x8664는 64bit 아키텍처(AMD 64)를 의미한다. |
| Number of Sections(6) | 파일에 포함된 섹션의 개수로 여기서는 6개이며, 이 값은 PE 파일에서 정의된 섹션 헤더의 수와 일치해야 한다. |
| Time Date Stamp  (5F55FCAC) | 파일이 빌드된 날짜와 시간을 나타내며 유닉스 타임 스탬프이다. 이 파일은 2020년 9월 7일에 빌드되었다. |
| Pointer to Symbol Table(0) | 심볼 테이블의 포인터를 나타낸다. 대부분의 PE 파일에서는 사용되지 않으므로 값이 0이다. |
| Number of Symbols(0) | 심볼 테이블에 있는 심볼의 개수로 대부분의 경우 0이다. |
| Size of Optional Header(0x0023) | Optional header의 크기를 바이트 단위로 나타낸다. |
| Characteristics(0x0023) | 파일의 특성을 나타내며 여러 플래그로 이루어져 있다.  0x0001: IMAGE\_FILE\_RELOCS\_STRIPPED(재배치 정보가 제거됨)  0x0002: IMAGE\_FILE\_EXECUTABLE\_IMAGE(실행 파일임)  0x0020: IMAGE\_FILE\_LARGE\_ADDRESS\_AWARE(2GB 이상의 가상 주소 공간 사용 가능) |

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

그림 7 IMAGE\_NT\_Headers [Optional Header]

Optional Header(IMAGE\_OPTIONAL\_HEADER): 프로그램의 메모리 레이아웃, 엔트리 포인트, 서브시스템 타입 등 실행과 관련된 상세 정보를 확인할 수 있다.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

그림 8 IMAGE\_SECTION\_Header .text

🡪 섹션 헤더 중 하나인 .text 섹션은 실행 가능한 코드가 포함된 섹션으로, 프로그램이 실행될 때 CPU가 가장 먼저 접근하게 되는 섹션이다.

|  |  |
| --- | --- |
| Name(.text) | 섹션의 이름이 .data임을 나타낸다. |
| Virtual Size(0x00001EC0) | 섹션이 메모리에서 차지하는 실제 크기를 나타낸다. |
| Virtual Address(RVA)(0x00001000) | 메모리에서 이 섹션이 로드될 가상 주소를 나타낸다. ImageBase와 함께 실제 메모리 주소 계산에 사용된다. 일반적으로 .text 섹션은 0x1000에서 시작한다. |
| Size of Raw Data(0x00001E00) | 파일에 저장된 이 섹션의 크기로 코드의 크기를 의미한다. |
| Pointer to Raw Data(0x0000400) | 파일 내에서 이 섹션의 데이터가 시작되는 오프셋을 나타낸다. |
| Characteristics | 섹션의 속성을 정의한다.  0x00000020(IMAGE\_SCN\_CNT\_CODE): 이 섹션이 실행 가능한 코드를 포함하고 있음을 의미  0x20000000(IMAGE\_SCN\_MEM\_EXECUTE): 이 섹션이 실행 가능함을 나타낸다.  0x40000000(IMAGE\_SCN\_MEM\_READ): 이 섹션이 메모리에서 읽을 수 있음을 나타낸다. |

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

그림 9 IMAGE\_SECTION\_Header .rdata

🡪 .rdata 섹션은 주로 읽기 전용 데이터(상수, 문자열, 함수 포인터 테이블 등)를 포함한다.

|  |  |
| --- | --- |
| Name(.rdata) | 섹션의 이름이 .rdata임을 나타낸다. |
| Vritual Size(0x000004AE) | 메모리에서 이 섹션이 차지하는 실제 크기로 메모리에 로드된 후 프로그램이 읽을 수 있는 .rdata 섹션의 데이터 전체 크기를 의미한다. |
| RVA(0x00009A60) | 메모리에서 이 섹션이 로드될 가상 주소를 의미. |
| Size of Raw Data(0x00000400) | 파일에 저장된 이 섹션의 크기 |
| Pointer to Raw Data(0x00001200) | 파일 내에서 이 섹션의 데이터가 시작되는 오프셋을 나타낸다. |
| Characteristics(0x40000040) | 섹션의 속성을 정의한다.  IMAGE\_SCN\_CNT\_INITIALIZED\_DATA(0x00000040): 이 섹션이 초기화된 데이터를 포함하고 있음을 의미  IMAGE\_SCN\_MEM\_READ(0x40000000): 이 섹션이 읽기 가능함을 나타냄. |

텍스트, 폰트, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 번호, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

Helloworld 정적 분석 IDA

문자열 검색

텍스트, 스크린샷, 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

Shift + F12를 누르면 바이너리에 포함된 문자열이 열거된 strings 창이 나타난다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

밑으로 내리다 보면 Hello, world!\n라는 문자열이 보인다. 이 문자를 더블클릭하면 다음과 같은 창이 나온다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

상호참조(Cross Reference, XRef) 기능을 통해 수상한 값이나 함수를 참조하는 함수를 분석할 수 있다. 앞선 Hello, world!\n라는 문자열이 어디서 사용되는지 추적해보자

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

aHelloWorld를 클릭하고 상호참조 단축키(x)를 누르면 xrefs창이 나타난다. 이 창에는 해당 변수를 참조하는 모든 주소가 출력된다. 첫번째 항목을 더블클릭하여 이를 따라가면 main 함수를 찾을 수 있다.

텍스트, 전자제품, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

main 함수를 찾았으므로 F5키를 눌러 디컴파일 한다.

텍스트, 소프트웨어, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

여기서는 argc, argv, envp로 3개의 인자를 받는다고 해석했다.

[동작]

1. sleep함수를 호출하여 1초 대기한다,
2. qword\_14001DBE0에 Hello, world!\n 문자열의 주소를 넣는다.
3. sub\_140001060에 Hello,world!\n를 인자로 전달해서 호출한다.
4. 0을 반환한다.

여기서 qword\_14001DBE0 변수는 값이 변경될 수 있는 전역 변수이므로 data 섹션에 존재한다. 이는 변수를 더블클릭해서도 확인할 수 있다.



Hello, world!\n 문자열은 실행 도중 값이 변경될 일이 없는 상수이므로 rodata 섹션에 존재할 것이다.

sub\_140001060는 printf()함수이다.

텍스트, 소프트웨어, 디스플레이, 폰트이(가) 표시된 사진

자동 생성된 설명

sub\_140001060 함수를 디컴파일(F5) 해본 결과 va\_start 함수를 통해 가변 인자를 처리하는 함수임을 알 수 있다. \_\_acrt\_iob\_func 함수는 스트림을 가져올 때 사용되는 함수인데, 인자로 들어가는 1은 stdout을 의미한다. 따라서 문자열 인자를 받고 stdout 스트림을 내부적으로 사용하는 가변 함수임을 알 수 있다. 따라서 이런 정황들을 통해 printf 함수로 추정할 수 있다.

Hello world.exe 동적 분석 IDA

대부분의 디버거에는 중단점(Break point, F2)과 실행(Run, F9)이라는 기능이 있다. 중단점을 특정 주소에 설정하고, 실행 명령을 내리면 프로그램은 중단점까지 멈추지 않고 실행된다.

1. main 함수에 중단점을 설정한다. (F2)

텍스트, 폰트, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

1. 디버깅을 시작해서 main 함수까지 실행한다. (F9)

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

한 단계 실행(step over, F8)

텍스트, 소프트웨어, 웹 페이지, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

sub rsp, 38 🡪 main 함수가 사용할 스택 영역 확보

mov [rsp+38h+dwMilliseconds], 3E8h 🡪 rsp+0x20에 4바이트 값인 0x000003e8 저장

mov ecx, [rsp+38h+dwMilliseconds] ; dwMilliseconds 🡪 rsp+0x20에 저장된 값을 ecx에 옮김. 함수의 첫번째 인자를 설정하는 것

call cs:Sleep Sleep 함수 호출. ecx가 0x3e8이므로, Sleep(100)이 실행되어 1초간 실행이 멈춤

lea rax, aHelloWorld ; “Hello, world!\n” “Hello, world!\n”문자열의 주소를 rax에 옮김

아래의 메모리 덤프 창을 이용하여 0x14001a140의 데이터를 보면 실제로 해당 문자열이 저장되어 있음을 확인할 수 있다.

mov cs:qword\_14001DBE0, rax 🡪 rax의 값을 data 세그먼트의 주소인 0x14001a140에 저장한다.

mov rcx, cs:qword\_14001DBE0 🡪 0x14001DBE0에 저장된 값을 rcx에 옮긴다. 이는 다음 호출할 함수의 첫번째 인자로 사용된다.

call sub\_140001060 🡪 0x140001060 함수를 호출한다. 우리는 이를 정적 분석을 통해 printf 함수로 추측했었다.

프로그램을 확인하면, Hello, world!가 출력되어 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

xor eax, eax

add rsp, 38h 🡪 시작할 때 확장한 스택 영역을 다시 축소시킨다.

retn 🡪 ret으로 원래 실행 흐름으로 돌아간다.

main endp

함수 내부로 진입하기(step into, F7)

함수를 분석하다 보면 그 함수가 호출하는 다른 함수까지 정밀하게 분석해야 할 때가 있다.

1. 디버깅을 중단하고(Ctrl+F2), printf를 호출하는 0x14000110b에 중단점을 설정한다.

텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 디버깅을 다시 시작하고 continue(F9)를 클릭해서 printf 함수에 도달한다.

텍스트, 전자제품, 디스플레이, 스크린샷이(가) 표시된 사진

자동 생성된 설명

printf() 함수 호출 직전 중단된 모습

1. F7 단축키를 통해 함수 내부로 들어간다. 함수 내부로 rip가 ㅣ동한 것을 확인할 수 있다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

printf() 함수 내부에 진입한 상태

Appendix, 실행 중인 프로세스 조작하기

IDA를 이용하면 실행 중인 프로세스의 메모리를 조작할 수 있다. 기존 코드에서는 Sleep(delay=100)을 호출해서 1초 동안 프로세스를 정지시켰는데 delay 값을 1000000으로 조작해서 1000초 동안 프로세스를 정지시킨다.

1. delay를 Sleep 함수의 인자로 전달하는 부분에 중단점을 설정하고, 프로세스를 재시작한다.

텍스트, 전자제품, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

1. 스택을 보면 rsp+0x20에 delay 값인 0x3e8이 저장되어 있다.

텍스트, 전자제품, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

스택에 저장된 delay 변수 값

1. 해당 값을 클릭하고, F2를 누른 뒤 0xf4240(=100000)을 입력한다. 그리고 다시 F2를 눌러서 값을 저장한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. F9를 눌러서 Sleep 함수를 호출한다. 아까와 다르게 한참을 기다려도 프로세스가 재개되지 않는다.

텍스트, 전자제품, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

BreakPoint(F2): 중단점 설정. 프로그램이 해당 지점에 도달하는 순간 정지한다.

Restart(Ctrl+F2): 디버깅 중단

Run(F9): 프로그램 계속 실행, 또는 디버깅 시작

Step Into(F7): 어셈블리 코드를 한 줄 실행. 함수 호출의 경우 함수 내부로 들어감

Step Over(F8): 어셈블리 코드 한 줄 실행

**Reversing Basic Challenge #0**

이 문제는 사용자에게 문자열 입력을 받아 정해진 방법으로 입력값을 검증하여 correct 또는 wrong을 출력하는 프로그램이 주어집니다. 해당 바이너리를 분석하여 correct를 출력하는 입력값을 찾으세요!

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

일단 main 함수를 찾았다. main 함수에서 사용자 입력과 미리 설정된 값을 비교하는 조건문을 통해 correct를 출력한다고 추론했다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

F5키를 눌러 디컴파일을 했다.

어떤 입력이 들어왔을 때 correct를 출력하는지 확인하기 위해 sub\_140001000(v4)을 더블클릭한다.

텍스트, 폰트, 소프트웨어, 라인이(가) 표시된 사진

자동 생성된 설명

이 프로그램은 사용자가 입력한 a1과 Compar3\_the\_string 문자열을 비교해서 두 문자열이 완전히 동일할 때 correct를 출력함을 알 수 있다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

디버깅을 통해 Compar3\_the\_str1ng을 입력하고 Correct가 나오는지 확인한다.

**Petya 랜섬웨어의 초기 액세스 방법 분석**

1. 개요

Petya 랜섬웨어는 2016년에 처음 등장한 악성 프로그램으로, 컴퓨터의 부팅 섹터를 암호화하여 시스템 접근을 차단하고 금전을 요구하는 악성 소프트웨어입니다. 이 랜섬웨어는 초기에는 주로 이메일을 통한 피싱 공격으로 유포되었으며, 이후 여러 가지 다른 공격 기법들이 결합되어 더욱 복잡한 형태로 발전해왔습니다. 본 보고서는 Petya 랜섬웨어의 초기 액세스 방법을 분석하고, MITRE ATT&CK 매트릭스에서 관련된 기술을 매핑하여 공격 흐름을 이해하는 것을 목적으로 합니다.

2. Petya 랜섬웨어의 초기 액세스 방법 분석

초기 Petya 랜섬웨어는 주로 다음과 같은 방법으로 목표 시스템에 접근했습니다.

피싱 이메일 (Phishing Emails):

Petya 랜섬웨어는 악성 첨부파일을 포함한 이메일을 통해 배포되었습니다. 이 이메일은 사용자에게 중요한 문서인 것처럼 위장되어 있으며, 사용자가 첨부파일을 열도록 유도합니다. 첨부파일은 일반적으로 실행 가능한 파일(.exe) 또는 매크로가 포함된 문서(.docx, .xls 등)로, 사용자가 이를 실행하면 악성 코드가 시스템에 다운로드됩니다.

악성 첨부파일 (Malicious Attachments):

Petya 랜섬웨어는 주로 이메일에 첨부된 악성 파일을 통해 전파됩니다. 이러한 파일은 실행 가능한 형태로 첨부되며, 사용자에 의해 실행될 경우 시스템에 악성 코드를 설치하게 됩니다.

워드 매크로 악용 (Exploitation of Word Macros):

첨부된 문서 파일은 워드 매크로 기능을 악용하여 사용자 시스템에 악성 코드를 설치합니다. 사용자가 매크로 실행을 허용하면, 매크로 스크립트가 자동으로 실행되며, 이를 통해 Petya 랜섬웨어가 설치됩니다.

원격 데스크톱 프로토콜(RDP) 크리덴셜 탈취 (RDP Credential Theft):

일부 변종에서는 취약한 RDP 세션을 악용해 원격으로 시스템에 접근하고, 관리자 권한을 획득하여 랜섬웨어를 설치하는 방식도 확인되었습니다.

3. MITRE ATT&CK 매트릭스 기술 매핑

Petya 랜섬웨어의 초기 액세스 방법은 MITRE ATT&CK 매트릭스의 다음 기술들과 매핑될 수 있습니다:

T1566.001: Spearphishing Attachment

설명: 공격자는 피싱 이메일을 통해 악성 파일을 전달하여 목표 시스템에 액세스합니다. Petya 랜섬웨어의 경우, 악성 첨부파일을 통해 시스템에 접근하고 악성 코드를 실행하는 전형적인 기법입니다.

T1204.002: User Execution - Malicious File

설명: 공격자는 사용자가 악성 파일을 직접 실행하도록 유도합니다. Petya 랜섬웨어의 첨부파일 실행 방식이 이에 해당하며, 사용자가 실행한 파일이 시스템에 악성 코드를 설치합니다.

T1203: Exploitation for Client Execution

설명: 공격자는 클라이언트 측 소프트웨어의 취약점을 악용하여 악성 코드를 실행합니다. Petya 랜섬웨어는 일부 변종에서 이러한 기법을 활용하여 설치를 진행합니다.

T1078: Valid Accounts

설명: 합법적인 계정 정보를 사용하여 시스템에 접근합니다. Petya 랜섬웨어는 RDP 크리덴셜 탈취를 통해 시스템에 접근하는 방법을 사용한 사례가 있습니다.

T1566.002: Spearphishing Link

설명: 악성 링크가 포함된 이메일을 통해 사용자가 악성 웹사이트를 방문하거나 악성 파일을 다운로드하도록 유도합니다. 일부 변종에서는 악성 링크를 통해 랜섬웨어를 유포하기도 했습니다.