

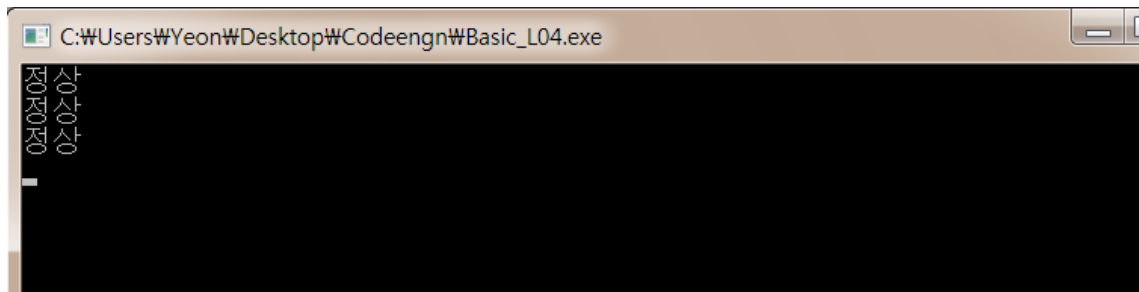
코드 엔진 Challenges: Basic 04

Author: CodeEngn

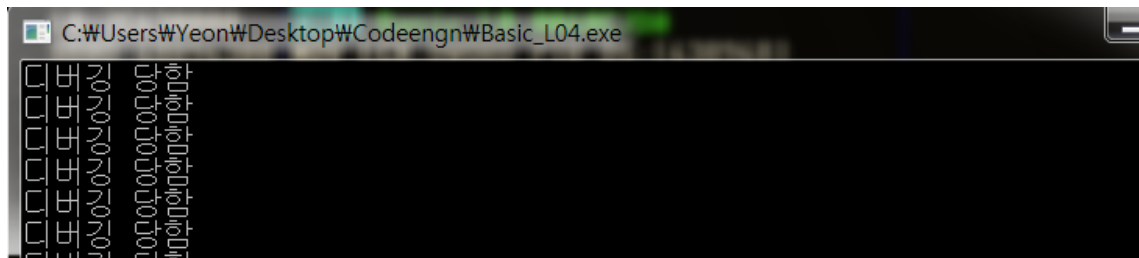
Korean: 이 프로그램은 디버거 프로그램을 탐지하는 기능을 가지고 있다. 디버거를 탐지하는 함수의 이름은 무엇인가?

문제를 읽어보면 안티 디버깅을 하는 함수의 이름을 찾으라고 하는 것 같다.

문제를 확인했으니 파일을 다운로드 받아서 실행해보자.



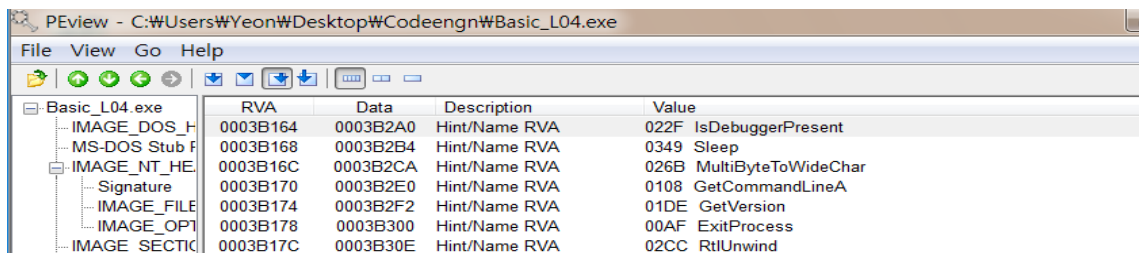
주기적으로 정상이라는 문자열을 출력하고 있다.



올리디버거를 통해 실행해보니 디버깅 당함이라는 문구를 출력한다.. 파일을 실행해보았으니 코드를 분석해 보도록 하자.

일단 PView를 통해 PE 구조를 보고 대략적으로 어떤 함수가 쓰일지를 살펴보자.

먼저, 이 파일의 Entry Point는 00408370이며



IAT를 보았을 때 IsDebuggerPresent라는 함수가 보인다. 이 함수가 안티디버깅의 역할을 할 것만 같다. 이를 확인해보기위하여 파일을 올리디버거로 실행해보자.

00408447	. 01 40894300	MOV EAX,DWORD PTR DS:[4389401]	
0040844C	. 50	PUSH EAX	
0040844D	. 8B0D 3C894300	MOV ECX,DWORD PTR DS:[43893C1]	
00408453	. 51	PUSH ECX	
00408454	. E8 B68BFFFF	CALL Basic_L0.0040100F	
00408459	. 83C4 0C	ADD ESP,0C	

ctrl+f8을 눌러 실행해보았을 때 00408454에서 멈추면서 디버깅 당함이라는 문구를 출력하는 화면이 뜬다. 이 흐름을 더욱 자세히 분석해보기위해 00408454에 BP를 걸고 코드 안으로 들어가보자.

00401048	> 8BF4	MOV ESI,ESP	
0040104A	. 68 E8030000	PUSH 3E8	
0040104F	. FF15 68B14300	CALL DWORD PTR DS:[<&KERNEL32.Sleep>	[Timeout = 1000. ms -Sleep
00401055	. 3BF4	CMPL ESI,ESP	
00401057	. E8 B4710000	CALL Basic_L0.00408210	
0040105C	. 8BF4	MOV ESI,ESP	
0040105E	. FF15 64B14300	CALL DWORD PTR DS:[<&KERNEL32.IsDebuggerPresent>	[IsDebuggerPresent
00401064	. 3BF4	CMPL ESI,ESP	
00401066	. E8 A5710000	CALL Basic_L0.00408210	
0040106B	. 85C0	TEST EAX,EAX	
0040106D	. 74 0F	JE SHORT Basic_L0.0040107E	
0040106F	. 68 24104300	PUSH Basic_L0.00431024	[Arg1 = 00431024

코드를 계속 실행해보면 00401048~0040108B를 반복하면서 디버깅 당함을 출력하고 있다. memory dump를 같이 살펴보면 이 디버깅 당함이라는 문구는 '00431024'에 들어있는 것 같다. 즉 위의 반복 루틴이 디버깅을 탐지하는 중요한 루틴이라는 것을 알 수 있다. 여기에서 호출하는 함수가 2가지가 보이는데 sleep 함수는 1초마다 '디버깅 당함'이라는 문구를 출력하는데 쓰이는 함수라는 것은 알 수 있으니 PEView에서 추측했던 IsDebuggerPresent 함수가 디버깅을 탐지하는데 쓰이는 함수라는 것을 확인할 수 있다. 이 함수를 우회하기 위해 이 함수가 어떤 것인지 알아보고 가자.

#isDebuggerPresent 함수

//호출 프로세스가 사용자 모드 디버거에 의해 디버그되고 있는지 여부를 확인한다.

BOOL WINAPI IsDebuggerPresent(void);

-이 함수에는 매개 변수가 없다.

-현재프로세스가 디버거의 컨텍스트에서 실행중인 경우 반환값은 0이 아닙니다.

-현재프로세스가 디버거의 컨텍스트에서 실행되고 있지 않으면 반환값은 0이다.

여기서 살펴봐야할 것은 리턴값이다. 반환값이 디버깅되고있지않으면 0을 반환한다는 것이다. 즉 IsDebuggerPresent 함수가 반환하는 값을 0으로 수정하면 안티디버깅을 우회할 수 있다. 일단은 IsDebuggerPresent가 반환하는 값이 무엇인지 보자.

Registers (FPU)		
EAX	00000000	
ECX	7523189F	KERNEL
EDX	770A70B4	ntdll
EBX	7FFDF000	
ESP	0012FEFC	
EBP	0012FF48	
ESI	0012FEFC	
EDI	0012FF48	
EIP	0040105E	Basic

Registers (FPU)		
EAX	00000001	
ECX	7523189F	KERNEL
EDX	770A70B4	ntdll
EBX	7FFDF000	
ESP	0012FEFC	
EBP	0012FF48	
ESI	0012FEFC	
EDI	0012FF48	
EIP	00401064	Basic

[0040105E의 Register]

[00401064의 Register]

반환값 확인을 위해 0040105E와 00401064에 BP를 걸고 실행한 결과 EAX의 값이 0에서 1로 변하는 것을 볼 수 있다.

우회하기위해서 EAX의 값을 0으로 변환하면 안티디버깅을 우회할 수 있다.

0040105C	8BF4	MOV ESI,ESP	
0040105E	FF15 64B14300	CALL DWORD PTR DS:[<&KERNEL32.IsDebuggerPresent]	IsDebuggerPresent
00401064	B8 00000000	MOV EAX,0	
00401069	90	NOP	
0040106A	90	NOP	
0040106B	85C0	TEST EAX,EAX	

Register의 값을 직접적으로 수정해줘도 되지만 EAX에 0을 반환시키는 명령어를 삽입해 우회해보았다.

문제에 대한 답은 IsDebuggerPresent이다.