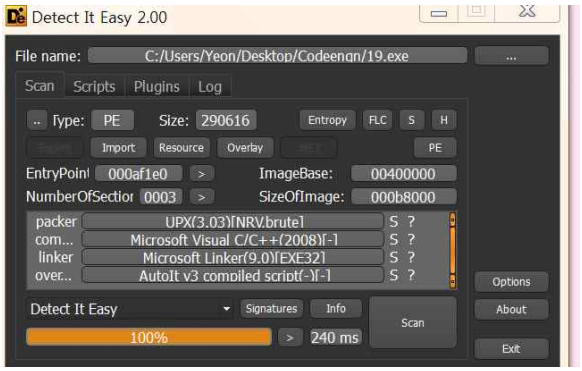
코드 엔진 Challenges: Basic 19

Author: CodeEngn
Korean: 이 프로그램은 몇 밀리 세컨드 후에 종료되는가

파일을 실행하면 위와 같은 프로그램이 나온다. 확인을 누르면 바로 종료되고 가만히 냅두면 몇 초 후에 알아서 종료된다. 이 프로그램이 몇 밀리 세컨드 후에 종료되는지 알아보기 위해서 파일을 분석해보자. 먼저 패킹 여부부터 알아보자.



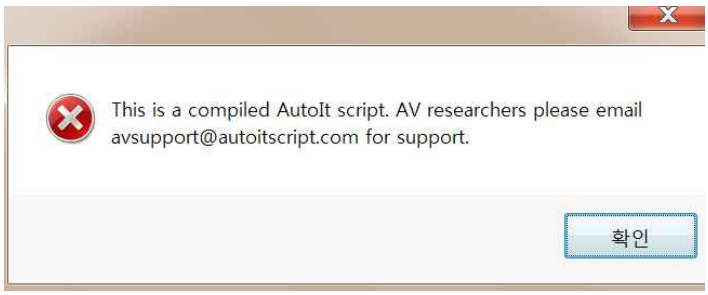

```
C:₩upx-3.95-win32>upx -d -o Basic_19_unpack.exe 19.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2018
UPX 3.95w       Markus Oberhumer, Laszlo Molnar & John Reiser   Aug 26th 2018

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    613176 <-    290616   47.40%    win32/pe     Basic_19_unpack.exe

Unpacked 1 file.
```

DE로 파일을 여니 UPX로 패킹되어 있는 것을 알 수있다. 바로 UPX 툴을 이용해 언패킹을 진행하였다.

언패킹을 진행했으니 다른 문제들과 비슷하게 올리디버거로 아스키 값과 함수 목록을 확인

하여 파악해보려 했지만 데이터가 너무 많아 파악하기 힘들다 . 그래서 많은 양의 데이터 중 필요한 데이터만 찾아내기 위해 문제에서 주어진 "밀리센컨드"라는 단어를 이용해 시간과 관련된 API를 찾아본 결과 timeGetTime() 함수가 리턴값이 밀리초임을 알았고 문제와 관련있음을 유추할 수 있었다. search for에서 timeGetTime함수를 찾아 이 함수가 있는 모든 곳에 BP를 걸어 분석해보자.

```
004443DC CALL DWORD PTR DS:[<&KERNEL32.Terminate kernel32.TerminateThread
0040B350 CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
0040E6CA CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
004301F3 CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
004305BC CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
0043197F CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
00431D70 CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
00431EED CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
00444C44 CALL EDI                                WINMM.timeGetTime
00451B82 CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
00451B9E CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
00456F68 CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
0046FBC1 CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
0046FBDC CALL DWORD PTR DS:[<&WINMM.timeGetTime> WINMM.timeGetTime
004183AA CALL DWORD PTR DS:[<&KERNEL32.TlsAlloc> kernel32.TlsAlloc
00418820 CALL DWORD PTR DS:[<&KERNEL32.TlsAlloc> kernel32.TlsAlloc
```



실행을 하자 위와 같은 메시지가 출력되었다. 확인을 누르니 종료되었다. 무엇 때문에 계속 종료되는지 고민하다 코드엔진 Basic4에서 확인했던 안티디버깅 함수인 IsDebuggerPresent() 함수가 떠올랐고 이 함수를 찾아보았다.

```
004187D0 CALL ESI                                kernel32.GetProcAddress
00418820 CALL DWORD PTR DS:[<&KERNEL32.TlsAlloc> kernel32.TlsAlloc
004188EA CALL DWORD PTR DS:[<&KERNEL32.GetCurren kernel32.GetCurrentThreadId
004189F0 CALL DWORD PTR DS:[<&KERNEL32.IsDebugge kernel32.IsDebuggerPresent
004189FA CALL DWORD PTR DS:[<&KERNEL32.SetUnhand kernel32.SetUnhandledExceptionFilter
00418A07 CALL DWORD PTR DS:[<&KERNEL32.UnhandledE kernel32.UnhandledExceptionFilter
00418A22 CALL DWORD PTR DS:[<&KERNEL32.GetCurren KERNELBA.GetCurrentProcess
00418A29 CALL DWORD PTR DS:[<&KERNEL32.Terminate kernel32.TerminateProcess
```

안티디버깅함수가 존재하고 있고 이 곳에 BP를 걸고 IsDubuggerPresent 함수가 반환하는 값을 0으로 수정하여 안티디버깅을 우회할 수 있게 하기위해서 . 일단은 IsDubuggererPresent가 반환하는 값이 무엇인지 찾아보았다. 0040E961을 실행했을 때 EAX의 값은 1이 나온다.

#TEST

TEST 인수1,인수2

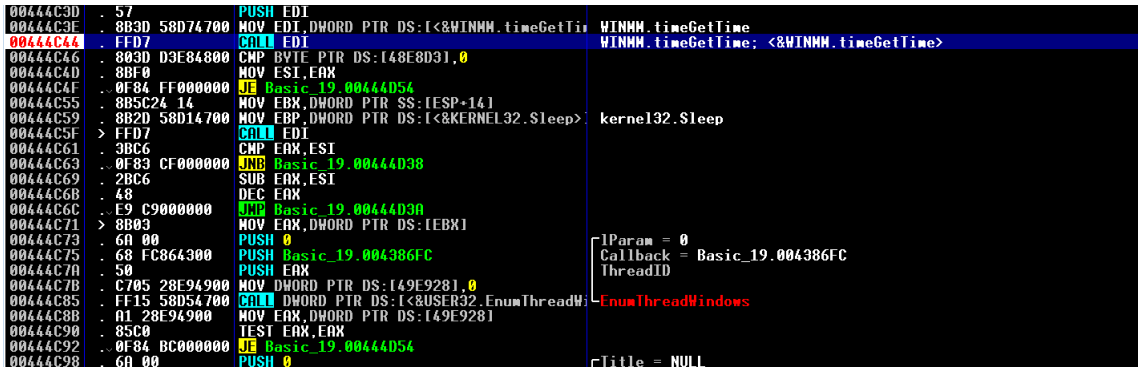
인수1과 인수2의 내용을 AND 연산하여 결과가 0이면 ZF를 1로 연산

=>위에서 TEST EAX,EAX가 의 연산결과 0이었으므로 ZF는 0인 상태이다.

#JNZ(Jump not Zero)

ZF가 0이거나 앞의 연산결과가 0이 아니면 점프한다.

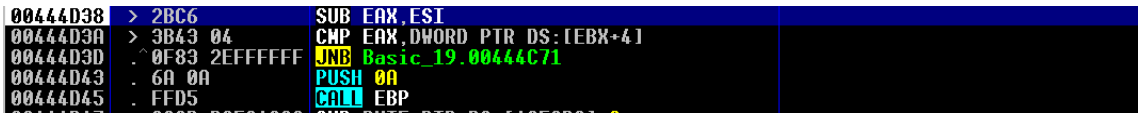
=>앞선 ZF가 0이기에 004338DE로 점프하게 됨
우리는 우회를 하기위해 JZ 004338DE로 바꿔줘야한다
#JZ (Jump Zero)
ZF가 1이거나 앞의 연산 결과가 0이면 점프한다.
=>앞선 ZF가 0이기에 점프하지 않게 된다.



우회를 한 후에 다시 TimeGetTime() 함수에 BP를 걸어준 후 실행을 하면 00444c44에서 멈추는 것을 확인할 수 있다. 이 부분을 분석해보도록하자.

00444C3D 57 PUSH EDI
//스택에 EDI 값 적재
00444C3E 8B3D 58D74700 MOV EDI,DWORD PTR DS:[<&WINMM.timeGetTim>;
//TimeGetTime 함수의 주소값을 EDI에 넣음
00444C44 FFD7 CALL EDI
//TimeGetTime함수 호출
00444C46 803D D3E84800 > CMP BYTE PTR DS:[48E8D3],0
//0048E8D3의 1 바이트 값과 0을 비교
00444C4D 8BF0 MOV ESI,EAX
//EAX값을 ESI에 복사 => 둘이 같은 값으로 만들어줌
00444C4F 0F84 FF000000 JE Basic_19.00444D54
//00444C46의 값이 참이면 분기 =>0048E8D3의 1 바이트 값이 0과 같으면 분기
00444C55 8B5C24 14 MOV EBX,DWORD PTR SS:[ESP+14]
//ESP+14의 4바이트 값을 EBX에 넣음
00444C59 8B2D 58D14700 MOV EBP,DWORD PTR DS:[<&KERNEL32.Sleep>] ;
//kernel32.sleep
00444C5F > FFD7 CALL EDI
//TimeGetTime 함수 재호출
00444C61 3BC6 CMP EAX,ESI
//결과 시간인 EAX값과 처음 시간인 ESI을 비교
00444C63 0F83 CF000000 JNB Basic_19.00444D38

//cmp문이 참이면 분기 00444C4D의 명령때문에 무조건 00444D38로 분기



00444D38 > 2BC6 SUB EAX,ESI
//EAX값에서 ESI값 만큼 뺀 값을 EAX에 저장
00444D3A > 3B43 04 CMP EAX,DWORD PTR DS:[EBX+4]
//EBP+4의 4바이트를 EAX와 비교
00444D3D .^0F83 2EFFFFFF JNB Basic_19.00444C71
//EAX가 EBX+4보다 크거나 같으면 분기
#JB 뒤의 값이 크면 분기

로 분석할 수있다 .위의 과정은 해당문제의 핵심이 되는 부분이고 00444C5F에서 TimeGetTime함수르 재호출하여 00444C44에서 받아왔던 처음 시간 값과 비교하여 크거나 같으면 00444D38지점으로 분기하여 2번째 TimeGetTime함수의 반환값에서 1번째 TimeGetTime함수의 반환값을 뺀다. TimeGetTime 함수 반환의 차이값인 EAX와 EBX+4 주소가 가리키는 값을 비교하여 분기한다.

-차이값이 작으면 시간을 구하는 부분으로 돌아가 이 과정을 반복
-차이값이 크면 다른곳으로 분기 후에 프로그램을 종료

따라서 EAX와 비교하는 EBX+4의 줏가 가르키는 값을 확인하면 해당 문제를 해결할 수 있다.

EBX+4가 가리키는 값은 00 00 2B 70이다.
이 값을 10진수로 변환해 계산해보면 11.12초가 나온다.