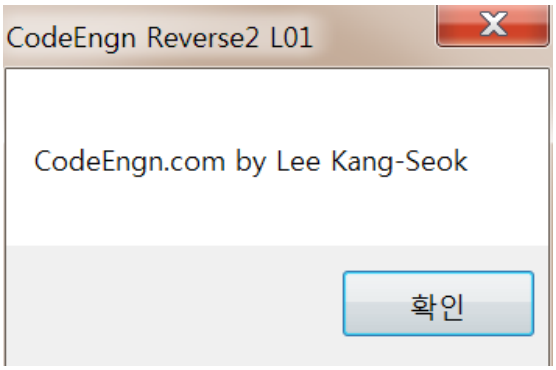
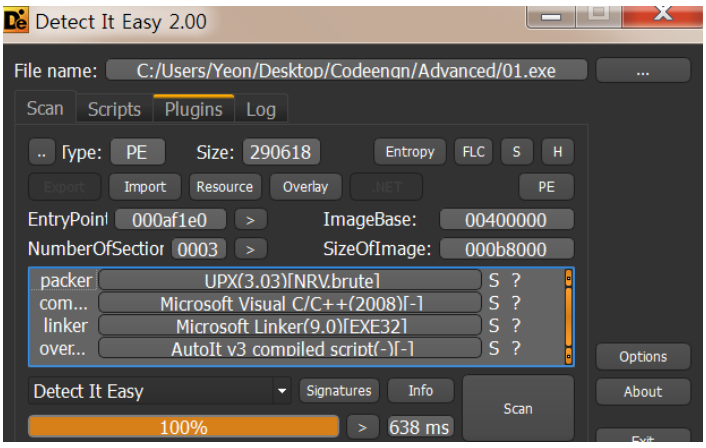
코드 엔진 Challenges: Advance 1

Author: CodeEngn
Korean: 이 프로그램은 몇 밀리 세컨드 후에 종료되는가
정답 인증은 MD5 해쉬값(대문자)변환 후 인증하시오.

몇 밀리 세컨드로 후에 종료되는가 하는 문제는 Basic19번에서 풀었던 문제와 같다. 어느 점이 다르고 어느점이 같은지 비교하면서 풀어보자 .



일단 프로그램은 이와 같다. 가만히 냅두면 몇 초후에 알아서 꺼진다. DE로 분석을 한 후 디버깅을 해보자.



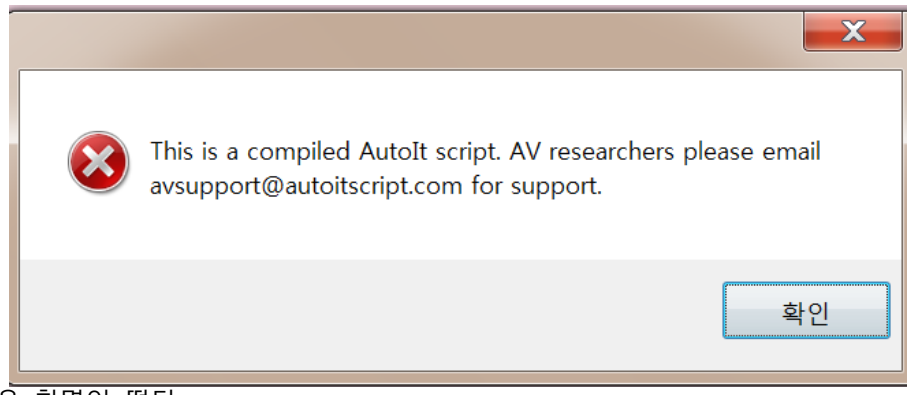
UPX로 패킹이되어 있는 것을 확인할 수 있다. 늘 그랬듯 UPX도구를 이용해 언패킹을 하자.

```
C:₩upx-3.95-win32>upx -d -o 01.unpack.exe 01.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
   --------------------   ------   -----------   -----------
    613178 <-    290618   47.40%    win32/pe     01.unpack.exe

Unpacked 1 file.
```

언패킹한 파일을 올리디버거로 분석해보자 . ms 초를 반환하는 19번과 유사하니 timeGetTime 함수를 찾아 BP를 걸고 실행해보자.



위와 같은 화면이 떴다.

```
0040E961 CALL DWORD PTR DS:[<&KERNEL32.IsDebugger kernel32.IsDebuggerPresent
004189F0 CALL DWORD PTR DS:[<&KERNEL32.IsDebugger kernel32.IsDebuggerPresent
00426997 CALL DWORD PTR DS:[<&KERNEL32.IsDebugger kernel32.IsDebuggerPresent
```

BP를 걸고 패턴을 알아보려했더니 계속 함수가 끝나버려서 혹시 안티디버깅 함수가 있나해서 찾아보니 역시 안티 디버깅함수가 들어있었다. 안티디버깅 함수가 반환하는 값을 0으로 수정하여 안티디버깅을 우회시키자.

```
0040E95B  . 57              PUSH EDI
0040E95C  . E8 1FDFFFFF     CALL 01_unpac.0040C880
0040E961  . FF15 20D34700   CALL DWORD PTR DS:[<&KERNEL32.IsDebugger IsDebuggerPresent
0040E967  . 85C0            TEST EAX,EAX
0040E969  .v0F84 6F4F0200   JE 01_unpac.004338DE
```

JNZ 분을 JE분으로 바꾸어 점프를 하지않게 했다. 그럼 다시 TimeGetTime() 함수에 BP를 걸어준 후 실행을 해보자 .

```
00444C44  . FFD7            CALL EDI                          WINMM.timeGetTime; <&WINMM.timeGetTime>
00444C46  . 803D D3E84800   CMP BYTE PTR DS:[48E8D3],0
00444C4D  . 8BF0            MOV ESI,EAX
00444C4F  .v0F84 FF000000   JE 01_unpac.00444D54
00444C55  . 8B5C24 14       MOV EBX,DWORD PTR SS:[ESP+14]
00444C59  . 8B2D 58D14700   MOV EBP,DWORD PTR DS:[<&KERNEL32.Sleep>] kernel32.Sleep
00444C5F  > FFD7            CALL EDI
00444C61  . 3BC6            CMP EAX,ESI
00444C63  .v0F83 CF000000   JNB 01_unpac.00444D38
00444C69  . 2BC6            SUB EAX,ESI
00444C6B  . 48              DEC EAX
00444C6C  .vE9 C9000000     JMP 01_unpac.00444D3A
00444C71  > 8B03            MOV EAX,DWORD PTR DS:[EBX]
00444C73  . 6A 00           PUSH 0                            lParam = 0
00444C75  . 68 FC864300     PUSH 01_unpac.004386FC            Callback = 01_unpac.004386FC
00444C7A  . 50              PUSH EAX                          ThreadID
00444C7B  . C705 28E94900   MOV DWORD PTR DS:[49E928],0
00444C85  . FF15 58D54700   CALL DWORD PTR DS:[<&USER32.EnumThreadWi EnumThreadWindows
00444C8B  . A1 28E94900     MOV EAX,DWORD PTR DS:[49E928]
00444C90  . 85C0            TEST EAX,EAX
00444C92  .v0F84 BC000000   JE 01_unpac.00444D54
```

계속 RETN 함수로 돌아가고 꺼져버리던것과 다르게 00444C44로 이동한 것을 알 수 있다. 이제 다시 찬찬히 분석을 해보도록 하자.

-00444C44 TimeGetTime 함수를 호출

-00444C46 0과 0040E8D3의 1바이트 값인 1과 비교

-00444C4D EAX의 값을 ESI에 저장

-00444C4F 00444C46 값이 참이면 분기함 => 그러나 같지않아서 분기하지않음

-00444C55 EXP+14의 4바이트 값을 EBX에 저장 =>값 008AF890가 EBX에 저장됨

-00444C59 kernel.sleep

-00444C5F TimegetTime() 함수 호출

-00444C61 EAX값과 ESI값을 비교 =>처음 시간과 두 번쨰 시간을 비교

※처음 00444C44에서 TimegetTime()함수를 호출할 때 systemtime 값을 EAX로 받아왔다가 ESI에 저장하고 두 번째로 00444C5F에서 두 번째 TimeGetTime()함수를 불러온 것은 EAX에 그대로 뒀다가 처음 시간과 두 번째 시간을 비교한다.
-00444C63 JNB 01.unpack
//비교한 결과가 크거나 같으면 점프 =>EAX값이 크므로 분기한다.

※JNB
Jump if Not Below (CF=0)
왼쪽 인자의 값이 오른쪽 인자의 값보다 작지 않으면(크거나 같으면) 점프 (부호없는)

-00444D38 EAX-ESI 의 값을 EAX에 저장 =>000027C1을 저장
-00444D3A EBX+4의 값(0000337B)과 EAX의 값을 비교

-00444D3D EAX가 EBX+4의 값보다 크거나 같다면 분기=> 작으므로 분기하지않는다.

로 분석 가능하다 .위의 과정은 해당문제의 핵심 부분이였다 결국 TimeGetTime()함수를 처음 받아왔던 시간가 두 번쨔로 받아왔던 시간을 비교하여 크거나 같으면 분기하여 2번째 호출함수의 반환값에서 1번쨔 함수의 반환값을 뺀다 .TimeGetTime 함수 반호나의 차이값인 EAX와 EBX+4의 값을 비교하여 분기할지말지 결정하는데

-차이값이 작으면 분기하지 않고 처음 다시 시간을 구하는 과정을 반복하고 그렇지 않으면 다른 곳으로 분기 후에 프로그램을 종료한다.

따라서 EAX와 비교하는 EBX+4 수가 가리키는 값을 확인하면 해당 문제를 해결 할 수 있다.
EBX+4sms 0x337B인데 이 값을 10진수(13179)로 고쳐 MD5로 변환하면 그게 답이다.