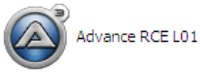
Advance RCE L01

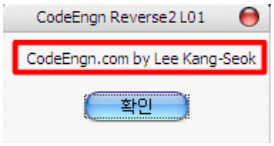
2010년 9월 3일 금요일

오후 5:09

파일 확인

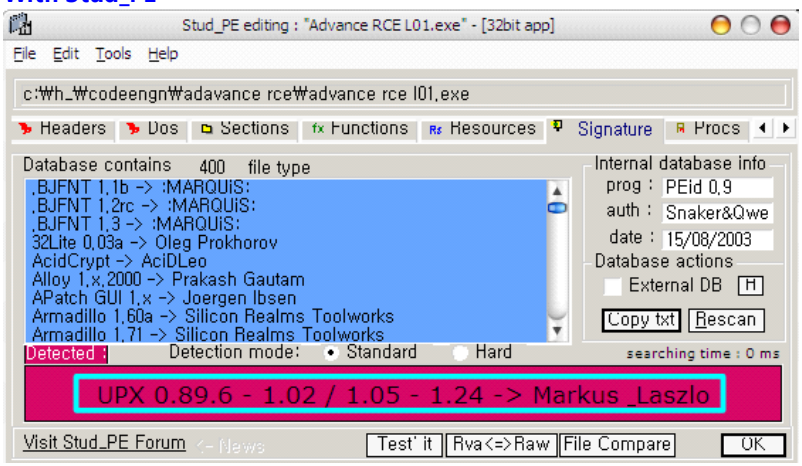


프로그램 실행



프로그램을 실행시키면 **몇 초 있다가 자동으로 꺼져버린다.**

With Stud_PE



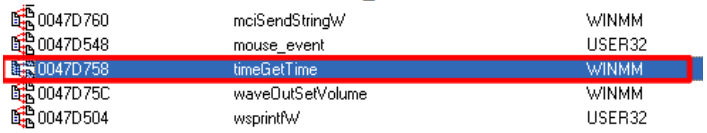
Program 을 확인한 결과 **UPX 로 Packing** 되어 있었다.

- **UPX 정도는 Tool 로 푸는게 빠르고 편하다.**

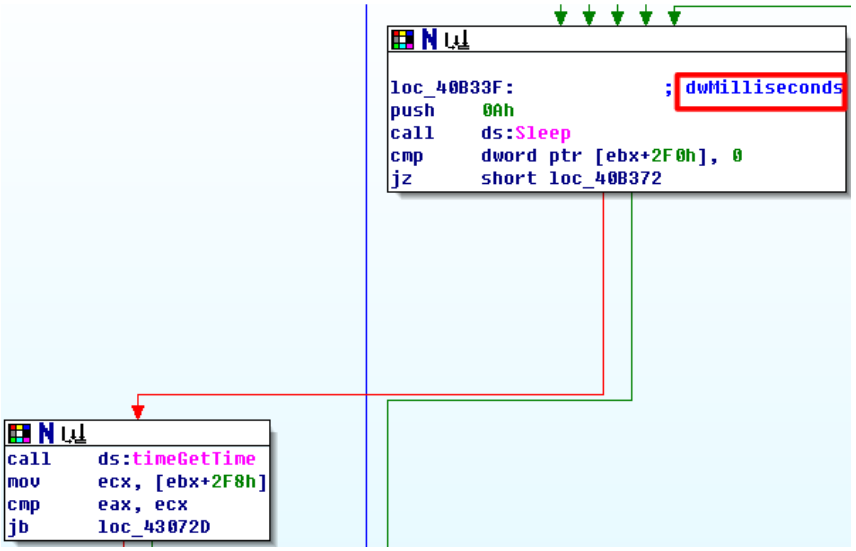
```
C:₩H_Util₩upx305w>upx -d unpack.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2010
UPX 3.05w       Markus Oberhumer, Laszlo Molnar & John Reiser   Apr 27th 2010

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

Unpacked 1 file.
```

With IDA

문제에서 몇 millisecond 후에 종료되는지를 물어 봤으므로, IDA 로 Time 관련 함수를 찾아보았다.



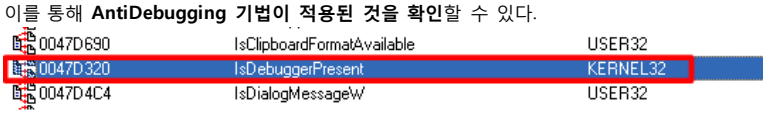
Graph Mode 로 확인



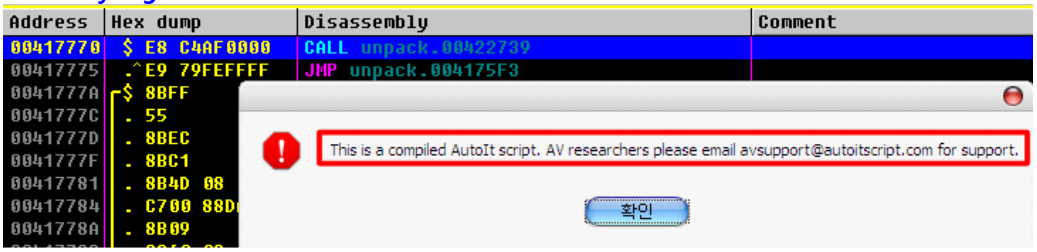
우리가 찾던 **Milliseconds 라는 글자가 보인다.**

Import 함수를 살펴 보던중 **Basic RCE L04 에서 보았던 IsDebuggerPresent 함수도 볼 수 있었다.**

- 이를 통해 **AntiDebugging 기법이 적용된 것을 확인**할 수 있다.



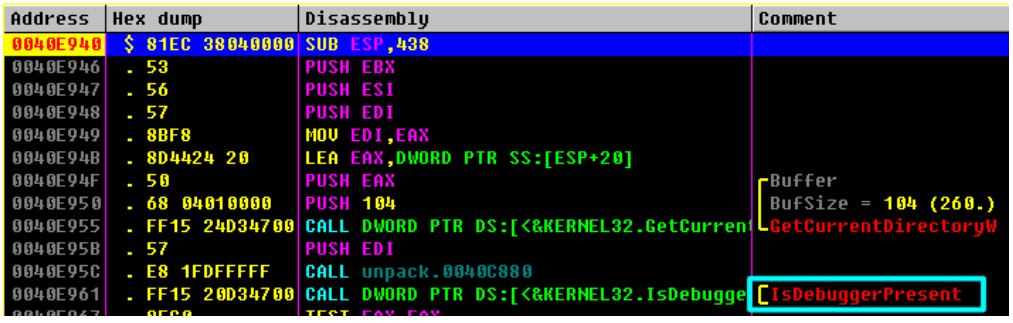
With Ollydbg



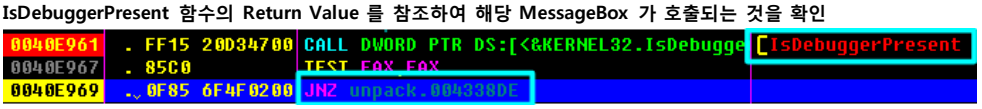
Unpacking 된 파일을 Ollydbg 로 실행을 시키면 위와 같은 창이 뜨게 된다.

- 확인을 누르니 프로그램이 종료 되었다.
- 이는 Anti Debugging 기법으로써, 위의 **MessageBox 가 호출되는 지점**을 찾아보았다.

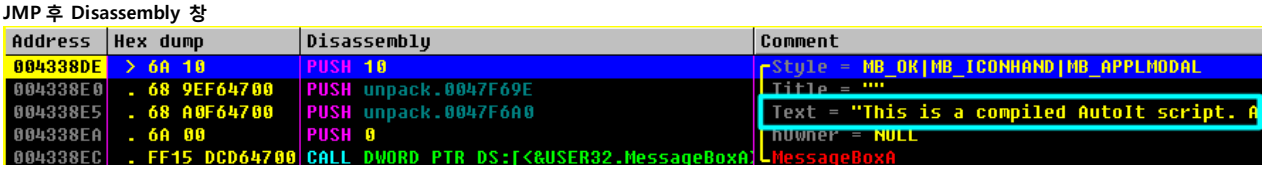
Step Over 와 Step Into 를 활용하며 MessageBox 의 Call 을 찾는 도중 IsDebuggerPresent 함수를 발견



IsDebuggerPresent 함수의 Return Value 를 참조하여 해당 MessageBox 가 호출되는 것을 확인



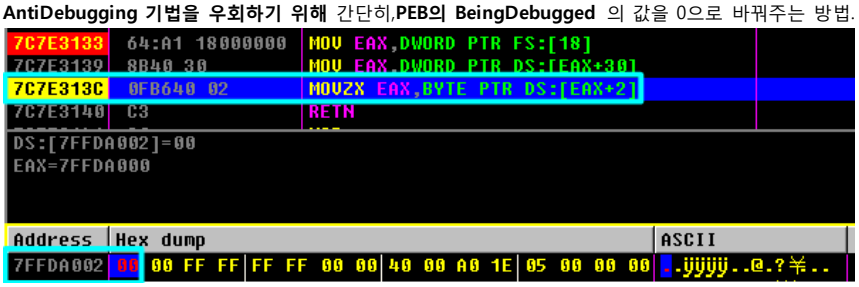
JMP 후 Disassembly 창



IsDebuggerPresent 함수를 우회 해야 함을 알 수 있다.

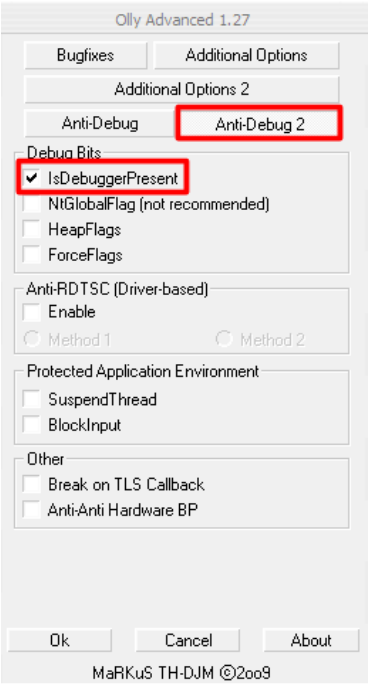
AntiDebugging 우회

AntiDebugging 기법을 우회하기 위해 간단히,**PEB의 BeingDebugged** 의 값을 0으로 바꿔주는 방법.



바꿔 준 후 실행 시키면 **Debugger 를 탐지 하지 못하고, 사용자가 Debugging 을 가능하게 해 준다.**

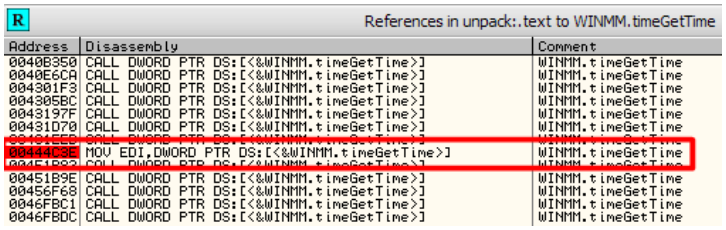
OllyAdvanced Plugin 을 이용하여 IsDebuggerPresent 함수를 우회 하는 방법



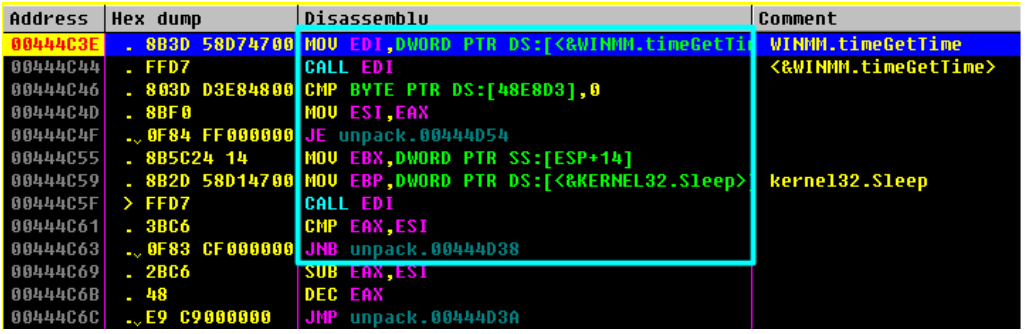
문제 해결

Ctrl + N 을 눌러 timeGettime 함수에 Breakpoint 를 걸어 놓았다.

- 단순히 호출 하는 부분이 아니라 MOV 연산을 수행하는 부분에 설정



Break 가 걸린 화면에서 Code 확인



- **timeGetTime 함수를 호출 한 후, 48E8D3 주소의 1Byte 값 (1) 과 0을 비교**한다.
- **ESI 에 timeGetTime 함수로 부터 얻어온 현재 시간을 넣는다.**
- **위의 비교 코드에서 값이 같으면 00444D54 로 JMP 를 한다.**
- **값이 다르므로, ESP+14 포인터가 가리키는 값 (008AF8C8) 을 EBX 에 넣고, Sleep 함수를 EBP 에 넣는다.**
- **timeGetTime 함수를 한번 더 호출하여, 좀 전에 받았던 시간 값과 비교를 하여 크거나 같으면 00444D38 지점으로 JMP 를 한다**

00444D38 Code

```
Address   Hex dump          Disassembly
00444D38  > 2BC6            SUB EAX,ESI
00444D3A  > 3B43 04         CMP EAX,DWORD PTR DS:[EBX+4]
00444D3D  .^0F83 2EFFFFFF   JNB unpack.00444C71
00444D43  . 6A 0A           PUSH 0A
00444D45  . FFD5            CALL EBP
00444D47  . 803D D3E84800   CMP BYTE PTR DS:[48E8D3],0
00444D4E  .^0F85 0BFFFFFF   JNZ unpack.00444C5F
```

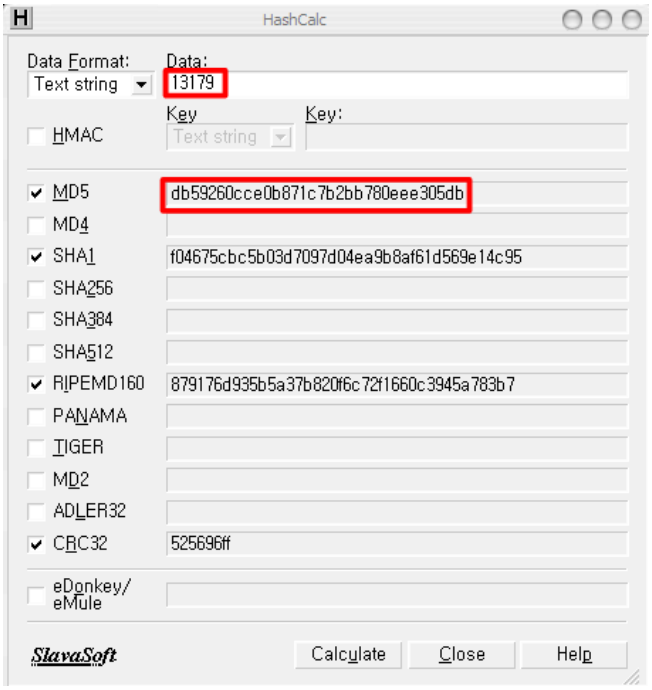
2번째 timeGetTime 함수의 반환 값에서 1번째 timeGetTime 함수의 반환값을 뺀다.

timeGetTIme 함수 반환의 차이값 (EAX) 과 EBX + 4 (008AF8CC) 주소가 가리키는 값 (337B)을 비교하여 분기

- **차이값이 더 작으면 다시 시간을 구하는 부분으로 돌아가 이 과정을 반복**하게 된다.
- **차이값이 더 크면 다른곳으로 JMP 후 프로그램을 종료** 시킨다.

결론적으로 16진수 **337B** = 13179 **millisecond 후에 프로그램을 종료** 시키게 된다.

MD5 Hash 값을 구하라고 했으므로, HashCalc Tool 을 사용하여 확인



보충 설명

timeGetTime

지금까지 흐른 시간을 **1/1000초 단위로 DWORD형을 Return**하는 함수

반환하는 값이 의미가 있는 함수로, **일정시간까지 동작을 하기 위해 사용하려면 계속 비교를 해서 맞지 않는 경우 반복** 수행

```
DWORD timeGetTime(VOID);
```

Parameters

This function does not take parameters.

Return Values

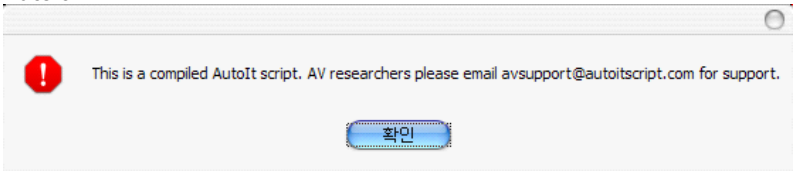
Returns the system time, in milliseconds.

Remarks

The only difference between this function and the **timeGetSystemTime** function is that **timeGetSystemTime** uses the **MMTIME** structure to return the system time. The **timeGetTime** function has less overhead than **timeGetSystemTime**.

Note that the value returned by the **timeGetTime** function is a **DWORD** value. The return value wraps around to 0 every 2^32 milliseconds, which is about 49.71 days. This can cause problems in code that directly uses the **timeGetTime** return value in computations, particularly where the value is used to control code execution. You should always use the difference between two **timeGetTime** return values in computations.

AutoIt



GUI 프로그램을 자동으로 실행시켜주는 스크립트 언어

- 일반 DOS 용 프로그램은 *.BAT 파일을 사용해서 스크립트 수행이 가능한데, GUI 프로그램은 *.BAT 스크립트로는 조작이 불가능하다.

- 실행까지는 가능한데 그 후에는 전혀 해볼수 있는게 없다.
  - **AutoIt 을 사용하면 키보드 입력이나 윈도우 핸들을 사용한 검색등이 가능**하다.
  - 약간의 Windows API 지식이 필요하다.

IsDebuggerPresent

TIB 블럭 **(FS : [18h])** 에서 PEB 구조체 주소값을 구한 뒤, **PEB의 BeingDebugged 필드를 반환하는 함수**이다.

```
7C7E3133   64:A1 18000000   MOV EAX,DWORD PTR FS:[18]
7C7E3139   8B40 30          MOV EAX,DWORD PTR DS:[EAX+30]
7C7E313C   0FB640 02        MOVZX EAX,BYTE PTR DS:[EAX+2]
```

Windbg 로 확인

> dt _teb

```
0:000> .symfix .
0:000> .sympath
Symbol search path is: srv*
Expanded Symbol search path is: cache*.;SRV*http://msdl.microsoft.com/download/symb
0:000> .reload
Reloading current modules
.....
0:000> dt _teb
ntdll!_TEB
   +0x000 NtTib            : _NT_TIB
   +0x01c EnvironmentPointer : Ptr32 Void
   +0x020 ClientId         : _CLIENT_ID
   +0x028 ActiveRpcHandle  : Ptr32 Void
   +0x02c ThreadLocalStoragePointer : Ptr32 Void
   +0x030 ProcessEnvironmentBlock : Ptr32 _PEB
   +0x034 LastErrorValue   : Uint4B
   +0x038 CountOfOwnedCriticalSections : Uint4B
   +0x03c CsrClientThread  : Ptr32 Void
```

0x030은 **PEB 구조체의 시작 주소**이다.

> dt _peb

```
0:000> dt _peb
ntdll!_PEB
   +0x000 InheritedAddressSpace : UChar
   +0x001 ReadImageFileExecOptions : UChar
   +0x002 BeingDebugged    : UChar
   +0x003 SpareBool        : UChar
   +0x004 Mutant           : Ptr32 Void
   +0x008 ImageBaseAddress : Ptr32 Void
   +0x00c Ldr              : Ptr32 _PEB_LDR_DATA
   +0x010 ProcessParameters : Ptr32 _RTL_USER_PROCESS_PARAMETERS
   +0x014 SubSystemData    : Ptr32 Void
```

BeingDebugged라는 멤버 변수의 값을 읽어서 **1이면 debugger 탐지, 0이면 실행**을 수행한다.

선언 예)

선언 : public declare funciton InDebuggerPresent Lib "kernel32.dll" () As Long

사용 :
```
If IsDebuggerPresent Then
        MsgBox "Debugger Found ", vbCritical,  "Program will be exited"
        End
Endif
```

답

DB59260CCE0B871C7B2BB780EEE305DB