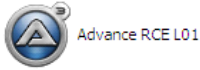


Advance RCE L01

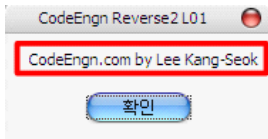
2010년 9월 3일 금요일
오후 5:09

파일 확인



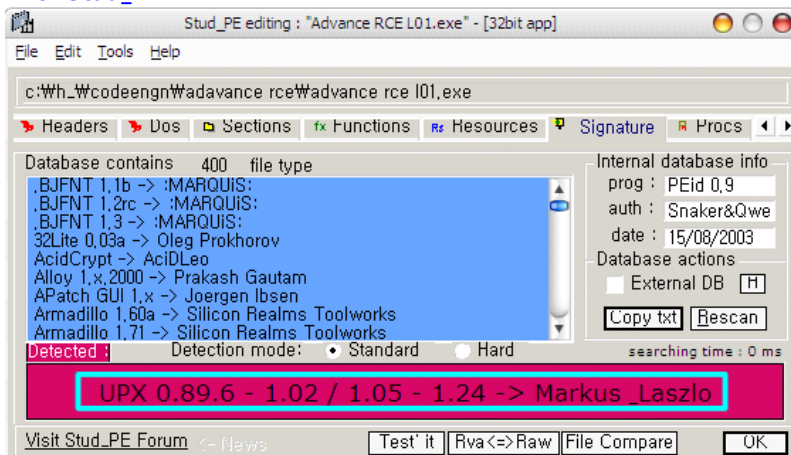
Advance RCE L01

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:\WH_Util\Wupx305w>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

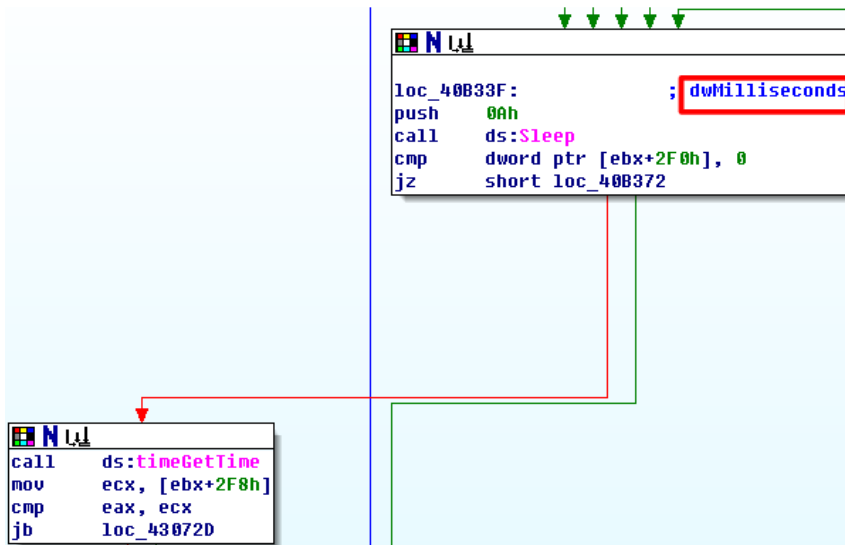
Unpacked 1 file.
```

With IDA

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

0047D760	mciSendStringW	WINMM
0047D548	mouse_event	USER32
0047D758	timeGetTime	WINMM
0047D75C	waveOutSetVolume	WINMM
0047D504	wsprintfW	USER32

Graph Mode 로 확인



우리가 찾던 **Milliseconds** 라는 글자가 보인다.

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

- 이를 통해 **AntiDebugging** 기법이 적용된 것을 확인할 수 있다.

0047D690	IsClipboardFormatAvailable	USER32
0047D320	IsDebuggerPresent	KERNEL32
0047D4C4	IsDialogMessageW	USER32

With Ollydbg

Address	Hex dump	Disassembly	Comment
00417770	\$ E8 C4AF0000	CALL unpack.00422739	
00417775	^ E9 79FEFFFF	JMP unpack.004175F3	
0041777A	\$ 8BFF		
0041777C	. 55		
0041777D	. 8BEC		
0041777F	. 8BC1		
00417781	. 8B4D 08		
00417784	. C700 88D		
0041778A	. 8B09		

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 지점을 찾아보았다.

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

Address	Hex dump	Disassembly	Comment
0040E940	\$ 81EC 38040000	SUB ESP,438	
0040E946	. 53	PUSH EBX	
0040E947	. 56	PUSH ESI	
0040E948	. 57	PUSH EDI	
0040E949	. 8BF8	MOV EDI,EAX	
0040E94B	. 8D4424 20	LEA EAX,DWORD PTR SS:[ESP+20]	
0040E94F	. 50	PUSH EAX	
0040E950	. 68 04010000	PUSH 104	Buffer BufSize = 104 (260.)
0040E955	. FF15 24034700	CALL DWORD PTR DS:[<&KERNEL32.GetCurrentDirectoryW	GetCurrentDirectoryW
0040E95B	. 57	PUSH EDI	
0040E95C	. E8 1FDFFFFF	CALL unpack.0040C880	
0040E961	. FF15 20034700	CALL DWORD PTR DS:[<&KERNEL32.IsDebuggerPresent	IsDebuggerPresent

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

0040E961	. FF15 20034700	CALL DWORD PTR DS:[<&KERNEL32.IsDebuggerPresent	IsDebuggerPresent
0040E967	. 85C0	TEST EAX,EAX	
0040E969	. 0F85 6F4F0200	JNZ unpack.004338DE	

JMP 후 Disassembly 창

Address	Hex dump	Disassembly	Comment
004338DE	> 6A 10	PUSH 10	Style = MB_OK MB_ICONHAND MB_APPLMODAL
004338E0	. 68 9EF64700	PUSH unpack.0047F69E	Title = ""
004338E5	. 68 A0F64700	PUSH unpack.0047F6A0	Text = "This is a compiled AutoIt script. A
004338EA	. 6A 00	PUSH 0	hOwner = NULL
004338EC	. FF15 DCD64700	CALL DWORD PTR DS:[<&USER32.MessageBoxA	MessageBoxA

IsDebuggerPresent 함수를 우회 해야 함을 알 수 있다.

AntiDebugging 우회

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

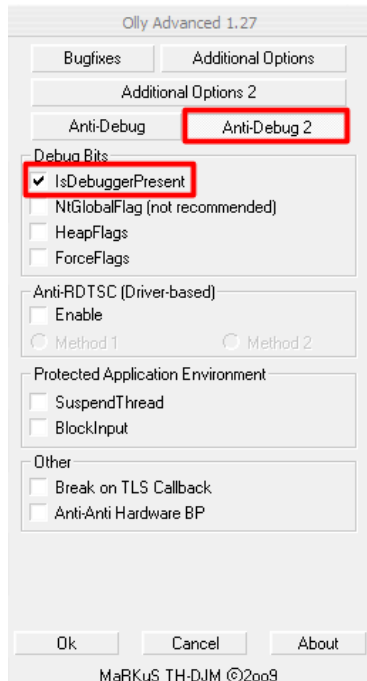
7C7E3133	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
7C7E3139	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]
7C7E313C	8FB640 02	MOVZX EAX,BYTE PTR DS:[EAX+2]
7C7E3140	C3	RETN

DS:[7FFDA002]=00
EAX=7FFDA000

Address	Hex	dump	ASCII
7FFDA002	00	FF FF FF FF 00 00 40 00 A0 1E 05 00 00 00	.jjjjj..e.?..

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

References in unpack:.text to WINMM.timeGetTime		
Address	Disassembly	Comment
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
0043137F	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
00431D70	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
00444C3E	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
00444C3E	MOV EDI,DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
00444C3E	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
0046FBC1	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime

Break 가 걸린 화면에서 Code 확인

Address	Hex	dump	Disassembly	Comment
00444C3E	. 8B3D 58D74700		MOV EDI,DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
00444C44	. FFD7		CALL EDI	<&WINMM.timeGetTime>
00444C46	. 803D D3E84800		CMP BYTE PTR DS:[48E8D3],0	
00444C4D	. 8BF0		MOV ESI,EAX	
00444C4F	. 0F84 FF000000		JE unpack.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	. 0F83 CF000000		JNB unpack.00444D38	
00444C69	. 2BC6		SUB EAX,ESI	
00444C6B	. 48		DEC EAX	
00444C6C	. E9 C9000000		JMP unpack.00444D3A	

- timeGetTime 함수를 호출 한 후, 48E8D3 주소의 1Byte 값 (1) 과 0을 비교한다.
- 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

0044D38 Code

Address	Hex dump	Disassembly
0044D38	> 2BC6	SUB EAX,ESI
0044D3A	> 3B43 04	CMP EAX,DWORD PTR DS:[EBX+4]
0044D3D	^ 0F83 2FFFFFFF	JNB unpack.0044C71
0044D43	. 6A 0A	PUSH 0A
0044D45	. FFD5	CALL EBP
0044D47	. 803D D3E84800	CMP BYTE PTR DS:[48E8D3],0
0044D4E	^ 0F85 0BFFFFFF	JNZ unpack.0044C5F

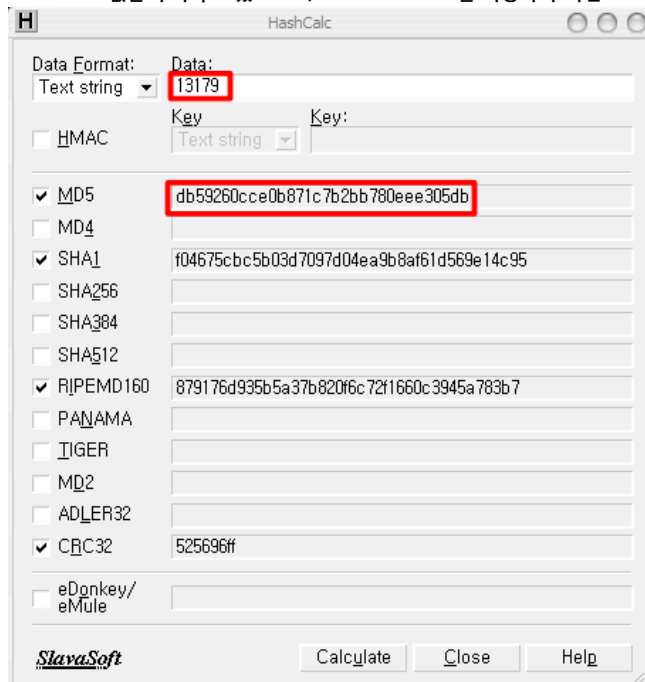
2번째 timeGetTime 함수의 반환 값에서 1번째 timeGetTime 함수의 반환값을 뺀다.

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 반복하게 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 Return하는 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

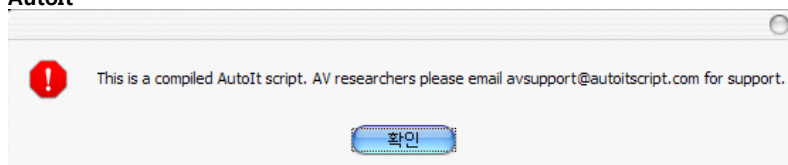
Returns the system time, in milliseconds.

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 스크립트로 조작이 불가능하다.

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

IsDebuggerPresent

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

7C7E3133	64A1 18000000	MOV EAX,DWORD PTR FS:[18h]
7C7E3139	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]
7C7E313C	0FB640 02	MOVBZ 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/symbols
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 주소이다.

```
> 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 실행을 수행한다.

선언 예)

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

사용 : If IsDebuggerPresent Then

```
MsgBox "Debugger Found ", vbCritical, "Program will be exited"
End
Endif
```

답

DB59260CCE0B871C7B2BB780EEE305DB