Smashing the Signature

(Korean Translation V.08-01)

안티바이러스의 시그니쳐 탐색 기법을 우회하기 위해 PE파일의 헤더 및 속성을 수정하여 코드 섹션을 암호화하는 기법을 소개함.

Hacking Group "OVERTIME"

MRB00 <bsh7983@hotmail.com> 2008.09.10

Title:

Reverse Engineering: Smashing the Signature

Table of Contents

Introduction

많은 AV나 안티스파이웨어 제품들은 그들이 가지고 있는 독특한 시그니처를 탐색하여 악성 프로그램을 확인한다. 그런 시그니쳐들은 최신의 데이터베이스에 저장되어 있다. 이 문서는 안티바이러스의 시그니쳐 체크 기술이 악성코드를 확인하는 것에 대해서 효과적이지 못하게 하기 위해 실행파일의 코드섹션을 암호화하는 여러 단계를 알려준다.

Tools

이 문서에서는 아래와 같은 툴을 사용함

- OllyDBG [http://www.ollydbg.de/]
  Plugins:
  - Analyze This! Plugin v0.1 by Joe Stewart
- WinAsm Studio [http://www.winasm.net/]
- A Hex editor

Example Software

Program Name: SimpleCrypt

Md5sum: 0550212afa60066cfd7c6d4e318d2c5f

Compiler: MASM (WinAsm)

Program Analysis

Source Code

simcrypt.asm

```
.486
.model   flat, stdcall
option   casemap :none    ; case sensitive

include      simcrypt.inc

.code
start:
        invoke   GetModuleHandle, NULL
        mov   hInstance, eax
        invoke   DialogBoxParam, hInstance, 101, 0, ADDR DlgProc, 0
        invoke   ExitProcess, eax
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DlgProc   proc   hWin      :DWORD,
                 uMsg     :DWORD,
                 wParam :DWORD,
                 lParam   :DWORD

 .if   uMsg == WM_COMMAND
   .if   wParam == IDC_ENCRYPT
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        invoke GetDlgItemText,hWin,EDIT1,addr userBuffer,32      ; Get 32
characters from Input textbox
      call Convert
      .if al == 1
        invoke SetDlgItemText,hWin,EDIT2,addr userBuffer      ; Print result to
```

```
Output textbox
        .else
          invoke MessageBox,hWin,addr nullPassMsg,addr
nullPassWnd,MB_ICONERROR
        .endif
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          .elseif   wParam == IDC_EXIT
        invoke EndDialog,hWin,0
     .endif
  .elseif   uMsg == WM_CLOSE
     invoke   EndDialog,hWin,0
  .endif

  xor   eax,eax
  ret
DlgProc   endp

Convert proc
invoke lstrlen, addr userBuffer
test eax,eax
jle NULLINPUT
mov ecx,offset userBuffer
xor ebx,ebx
@@:
  .if ebx<eax
     mov dl,byte ptr [ecx+ebx]       ; dl = ascii value of character in possition ebx (counter)
     add edx,ebx             ; edx = edx + ebx (counter)
     mov byte ptr[ecx+ebx],dl       ; character in possition ebx (counter) = dl
     inc ebx
     jmp @b
  .else
     mov al,1
     ret
  .endif
NULLINPUT:
  xor eax,eax
```

```
 ret
Convert EndP
end start
```

Simcrypt.inc

```
include   windows.inc

uselib   MACRO libname
 include      libname.inc
 includelib   libname.lib
ENDM

uselib   user32
uselib   kernel32

DlgProc      PROTO :DWORD,:DWORD,:DWORD,:DWORD

EDIT1        equ   1001
EDIT2        equ 1002
IDC_ENCRYPT     equ   1005
IDC_EXIT       equ   1004

.data
nullPassMsg   db   "NULL == Bad",0
nullPassWnd    db   "Error",0

.data?
hInstance   dd     ?
userBuffer   dd     32 dup(?)
```

simcrypt.rc

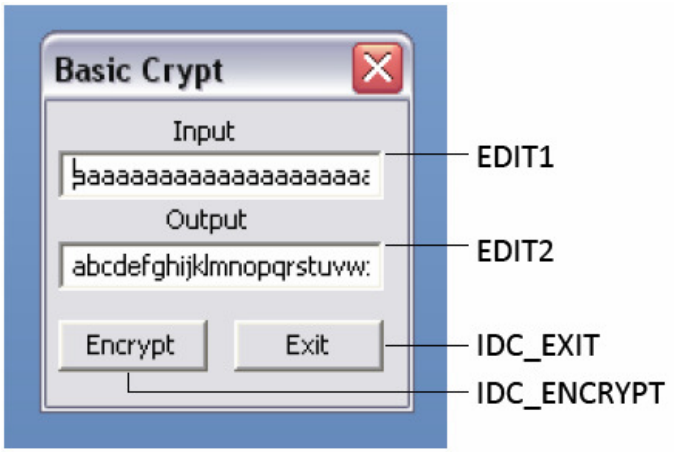
```
;This Resource Script was generated by WinAsm Studio.

#define EDIT2 1002
#define EDIT1 1001
#define IDC_STATIC1006 1006
```

```
#define IDC_STATIC1007 1007
#define IDC_ENCRYPT 1005
#define IDC_EXIT 1004


101 DIALOGEX 0,0,100,76
CAPTION "Basic Crypt"
FONT 8,"Tahoma"
STYLE 0x80c80880
EXSTYLE 0x00000000
BEGIN
  CONTROL "Exit",IDC_EXIT,"Button",0x10000000,52,55,41,13,0x00000000
  CONTROL "",EDIT1,"Edit",0x10000080,3,12,90,12,0x00000200
  CONTROL "",EDIT2,"Edit",0x10000080,3,35,90,12,0x00000200
  CONTROL "Encrypt",IDC_ENCRYPT,"Button",0x50010000,3,55,41,13,0x00000000
  CONTROL "Input",IDC_STATIC1006,"Static",0x50000000,35,3,24,8,0x00000000
  CONTROL "Output",IDC_STATIC1007,"Static",0x50000000,33,25,23,9,0x00000000
END
```

User Interface



Assembled Code

```
00401000   /$   6A 00              PUSH 0                                   ; /pModule = NULL
00401002   |.   E8 F9000000        CALL <JMP.&kernel32.GetModuleHandleA>    ; \  GetModuleHandleA
00401007   |.   A3 20304000        MOV DWORD PTR DS:[403020],EAX
0040100C   |.   6A 00              PUSH 0                                   ; /lParam = NULL
0040100E   |.   68 28104000        PUSH SimpleCr.00401028                   ; |DlgProc = SimpleCr.00401028
```

```
00401013  |.  6A 00              PUSH 0                                   ; |hOwner = NULL
00401015  |.  6A 65              PUSH 65                                  ; |pTemplate = 65
00401017  |.  FF35 20304000      PUSH DWORD PTR DS:[403020]               ; |hInst = NULL
0040101D  |.  E8 BA000000        CALL <JMP.&user32.DialogBoxParamA>       ; \DialogBoxParamA
00401022  |.  50                 PUSH EAX                                 ; /ExitCode
00401023  \.  E8 D2000000        CALL <JMP.&kernel32.ExitProcess>         ; \ExitProcess
00401028  /.  55                 PUSH EBP
00401029  |.  8BEC               MOV EBP,ESP
0040102B  |.  817D 0C 11010>     CMP DWORD PTR SS:[EBP+C],111
00401032  |.  75 65              JNZ SHORT SimpleCr.00401099
00401034  |.  817D 10 ED030>     CMP DWORD PTR SS:[EBP+10],3ED
0040103B  |.  75 47              JNZ SHORT SimpleCr.00401084
0040103D  |.  6A 20              PUSH 20                                  ; /Count = 20 (32.)
0040103F  |.  68 24304000        PUSH SimpleCr.00403024                   ; |Buffer = SimpleCr.00403024
00401044  |.  68 E9030000        PUSH 3E9                                 ; |ControlID = 3E9 (1001.)
00401049  |.  FF75 08            PUSH DWORD PTR SS:[EBP+8]                ; |hWnd
0040104C  |.  E8 97000000        CALL <JMP.&user32.GetDlgItemTextA>       ; \GetDlgItemTextA
00401051  |.  E8 59000000        CALL SimpleCr.004010AF
00401056  |.  3C 01              CMP AL,1
00401058  |.  75 14              JNZ SHORT SimpleCr.0040106E
0040105A  |.  68 24304000        PUSH SimpleCr.00403024                   ; /Text = ""
0040105F  |.  68 EA030000        PUSH 3EA                                 ; |ControlID = 3EA (1002.)
00401064  |.  FF75 08            PUSH DWORD PTR SS:[EBP+8]                ; |hWnd
00401067  |.  E8 88000000        CALL <JMP.&user32.SetDlgItemTextA>       ; \SetDlgItemTextA
0040106C  |.  EB 3B              JMP SHORT SimpleCr.004010A9
0040106E  |>  6A 10              PUSH 10                                  ; MB_OK|MB_ICONHAND|MB_APPLMODAL
00401070  |.  68 0C304000        PUSH SimpleCr.0040300C                   ; |Title = "Error"
00401075  |.  68 00304000        PUSH SimpleCr.00403000                   ; |Text = "NULL == Bad"
0040107A  |.  FF75 08            PUSH DWORD PTR SS:[EBP+8]                ; |hOwner
0040107D  |.  E8 6C000000        CALL <JMP.&user32.MessageBoxA>           ; \MessageBoxA
00401082  |.  EB 25              JMP SHORT SimpleCr.004010A9
00401084  |>  817D 10 EC030>     CMP DWORD PTR SS:[EBP+10],3EC
0040108B  |.  75 1C              JNZ SHORT SimpleCr.004010A9
0040108D  |.  6A 00              PUSH 0                                   ; /Result = 0
0040108F  |.  FF75 08            PUSH DWORD PTR SS:[EBP+8]                ; |hWnd
00401092  |.  E8 4B000000        CALL <JMP.&user32.EndDialog>             ; \EndDialog
```

```
00401097   |.   EB 10                JMP SHORT SimpleCr.004010A9
00401099   |>   837D 0C 10           CMP DWORD PTR SS:[EBP+C],10
0040109D   |.   75 0A                JNZ SHORT SimpleCr.004010A9
0040109F   |.   6A 00                PUSH 0                                   ; /Result = 0
004010A1   |.   FF75 08              PUSH DWORD PTR SS:[EBP+8]                ; |hWnd
004010A4   |.   E8 39000000          CALL <JMP.&user32.EndDialog>             ; \EndDialog
004010A9   |>   33C0                 XOR EAX,EAX
004010AB   |.   C9                   LEAVE
004010AC   \.   C2 1000              RETN 10
004010AF   $   68 24304000          PUSH SimpleCr.00403024                   ; /String = ""
004010B4   .   E8 4D000000          CALL <JMP.&kernel32.lstrlenA>            ; \lstrlenA
004010B9   .   85C0                 TEST EAX,EAX
004010BB   .   7E 1B                JLE SHORT SimpleCr.004010D8
004010BD   .   B9 24304000          MOV ECX,SimpleCr.00403024
004010C2   .   33DB                 XOR EBX,EBX
004010C4   >   3BD8                 CMP EBX,EAX
004010C6   .   73 0D                JNB SHORT SimpleCr.004010D5
004010C8   .   8A140B               MOV DL,BYTE PTR DS:[EBX+ECX]
004010CB   .   03D3                 ADD EDX,EBX
004010CD   .   88140B               MOV BYTE PTR DS:[EBX+ECX],DL
004010D0   .   43                   INC EBX
004010D1   .^  EB F1                JMP SHORT SimpleCr.004010C4
004010D3   .   EB 03                JMP SHORT SimpleCr.004010D8
004010D5   >   B0 01                MOV AL,1
004010D7   .   C3                   RETN
004010D8   >   33C0                 XOR EAX,EAX
004010DA   .   C3                   RETN
004010DB       CC                   INT3
004010DC   $-  FF25 20204000        JMP DWORD PTR DS:[<&user32.DialogBoxPara>;  user32.DialogBoxParamA
004010E2   $-  FF25 14204000        JMP DWORD PTR DS:[<&user32.EndDialog>]      ;  user32.EndDialog
004010E8   $-  FF25 10204000        JMP DWORD PTR DS:[<&user32.GetDlgItemTex>   ;
user32.GetDlgItemTextA
004010EE   $-  FF25 1C204000        JMP DWORD PTR DS:[<&user32.MessageBoxA>]    ;  user32.MessageBoxA
004010F4   $-  FF25 18204000        JMP DWORD PTR DS:[<&user32.SetDlgItemTex>   ;  user32.SetDlgItemTextA
004010FA   .-  FF25 04204000        JMP DWORD PTR DS:[<&kernel32.ExitProcess>   ;  kernel32.ExitProcess
00401100   $-  FF25 00204000        JMP DWORD PTR DS:[<&kernel32.GetModuleHa>;
```

kernel32.GetModuleHandleA					
00401106	$-	FF25 08204000	JMP DWORD PTR DS:[<&kernel32.lstrlenA>]	;	kernel32.lstrlenA

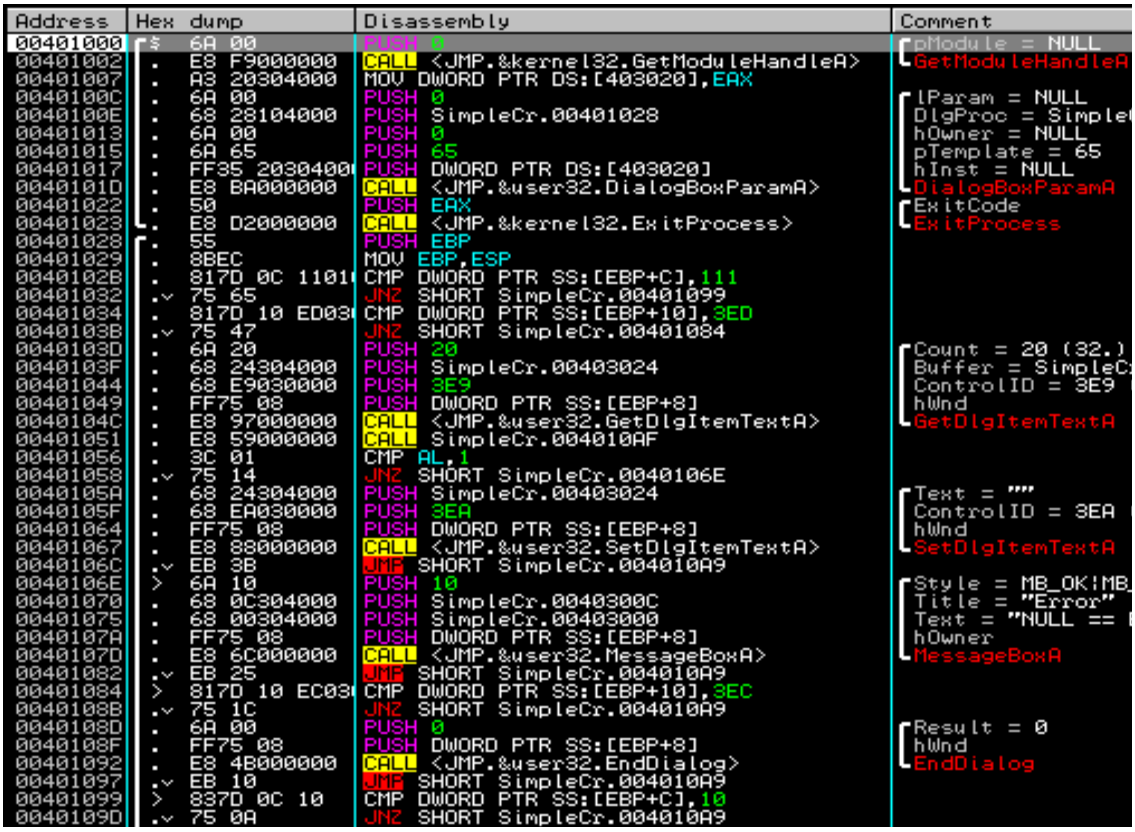
Binary Code Enctyption

바이너리코드를 암호화하는 방법은 간단하다. 소프트웨어의 바이너리는 static disassembly 에 취약할수 있다. 이 점을 피하기 위해서 코드를 암호화 해야 하며 실행 시 복호화 해야 한다. 추가로 이 기술은 대부분의 안티바이러스 시스템을 우회하는 간단한 방법이기도 하다. 단지 코드섹션만을 바꿈으로써 프로그램의 시그니쳐를 바꾸며 결국 탐지를 어렵게 하는 것이다.

비록 이론은 간단하나 예제를 만드는데 있어 사용할 기술의 이해에 대한 어려움이 따를 수 있다. 그러므로 추가정보를 제공할 것이다.

Step 1

올리디버거를 기동하여 타겟프로그램을 로드하라. CPU윈도우가 아래와 비슷하게 나올것이다.

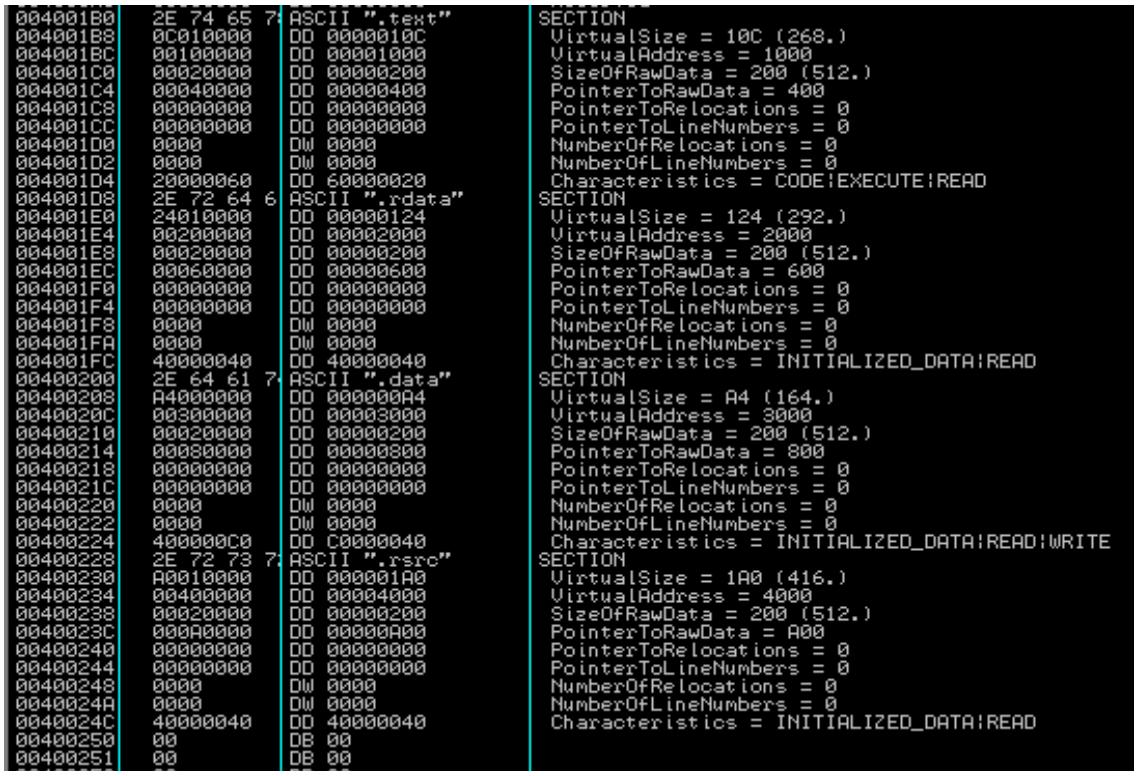


Step 2

만약 패치하고 싶은 코드 크기가 패치할 곳의 데이터섹션의 실제 크기보다 클 경우, 작업할 공간을 만들기 위해 PE헤더를 수정할 필요가 있을 것이다. 이 작업공간을 "**code cave**"라고 불린다.

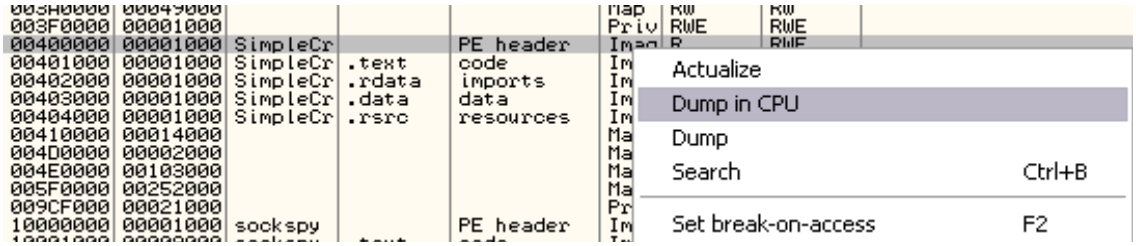
모든 윈도우 실행파일은 PE헤더를 포함한다. 헤더에 포함되는 정보는:

- Time and Date Stamp (시간, 날짜 스탬프)
- Checksum (무결성의 검사합)
- The address of the executable entry point (EP). 나중에 덮어씌울 곳이기 때문에 이 같은 경우 Original Entry Point of our code (OEP)이다.
- Section Headers (아래 참조)



위의 각 섹션 헤더들은 섹션의 특징을 정의하고 있다. 최대한 간단히 하기 위해 다른 섹션들사이에 존재하는 섹션의 크기가 증가되는 것을 피할 것이다. 그러므로 우리는 파일의 끝에 위치해 있는 .rsrc 섹션의 크기를 증가시키도록 하겠다.

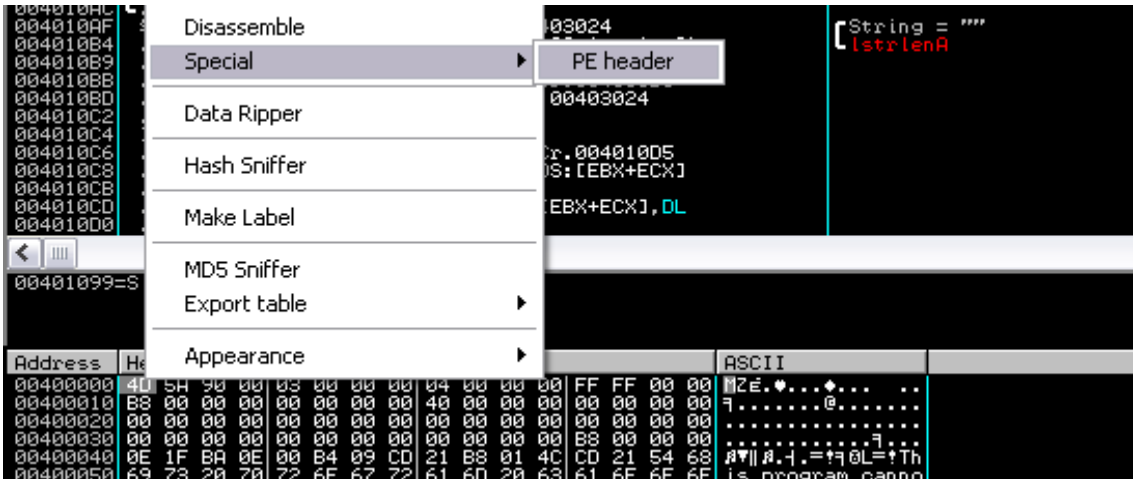
Memory 윈도우를 띄우고(Alt+M) > PE header에서 마우스 우클릭 > Dump in CPU선택



Step 3

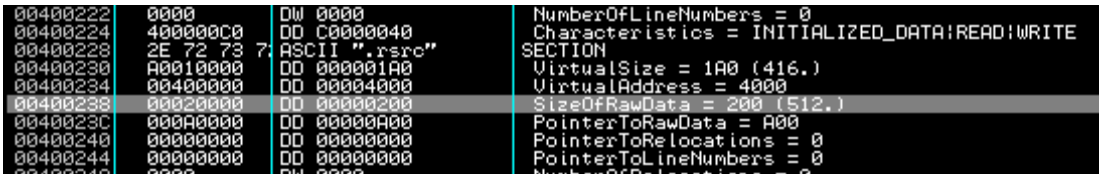
이곳을 PE Header로써 다루기 위해 덤프를 수정해라.

dump 윈도우에서 마우스 우클릭 > Special > PE Header 클릭



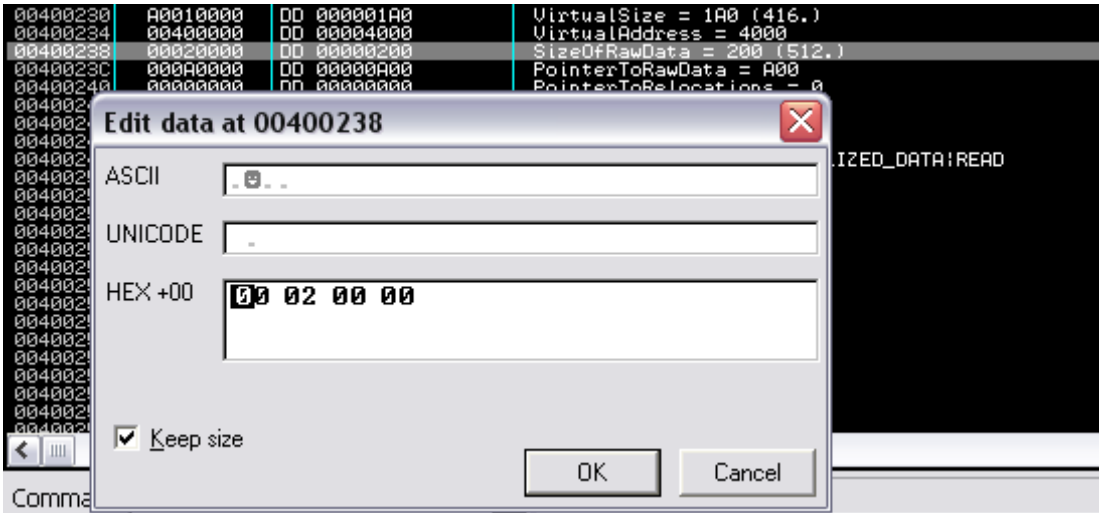
Step 4

.rsrc섹션에서 "SizeOfRawData"옵션을 찾을때까지 스크롤 다운.



Step 5

Ctrl+E 또는 우클릭 > Binary > .rsrc섹션의 크기를 수정하기 위해 Edit 선택



주의:

Intel *아키텍쳐에서 데이터는* ***"little Endian"*** *형식으로 표현된다. 이것은 아래 테이블에서 보여주는 것처럼* ***CPU****에 의해 역방향으로 읽혀짐을 뜻한다****.(1-4)***

4	***3***	***2***	***1***
00	***02***	***00***	***00***

=

1	***2***	***3***	***4***
00	***00***	***20***	***00***

=

0x200(16진수)는 512(10진수)와 같다.

Step 6

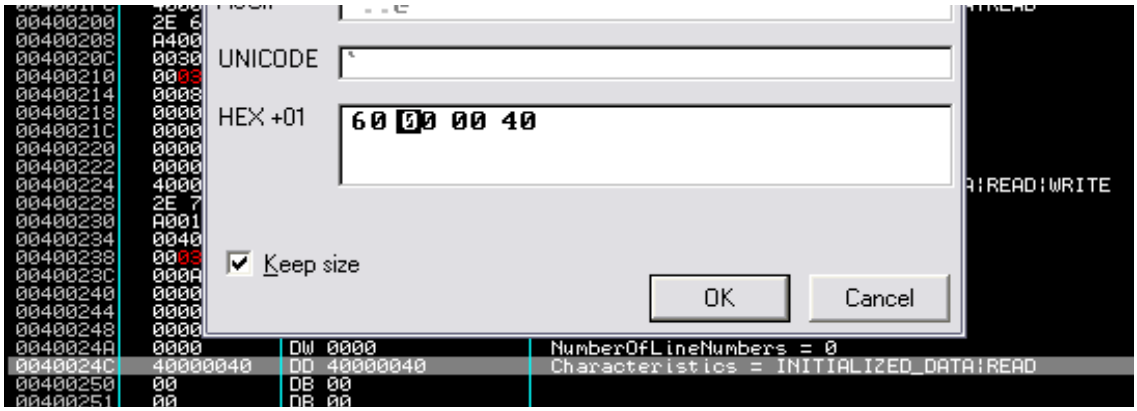
0x100(십진수 256)바이트를 섹션의 크기에 더해라 (0x200+0x100=0x300)

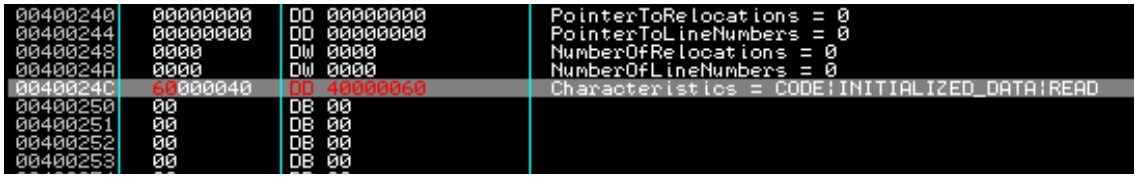
Step 7

이 섹션의 플래그("Characteristics")를 exeutable code를 포함하게 수정해라

DWORD 40000040의 첫바이트에 0x20을 더한다.(0x40+0x20=0x60)

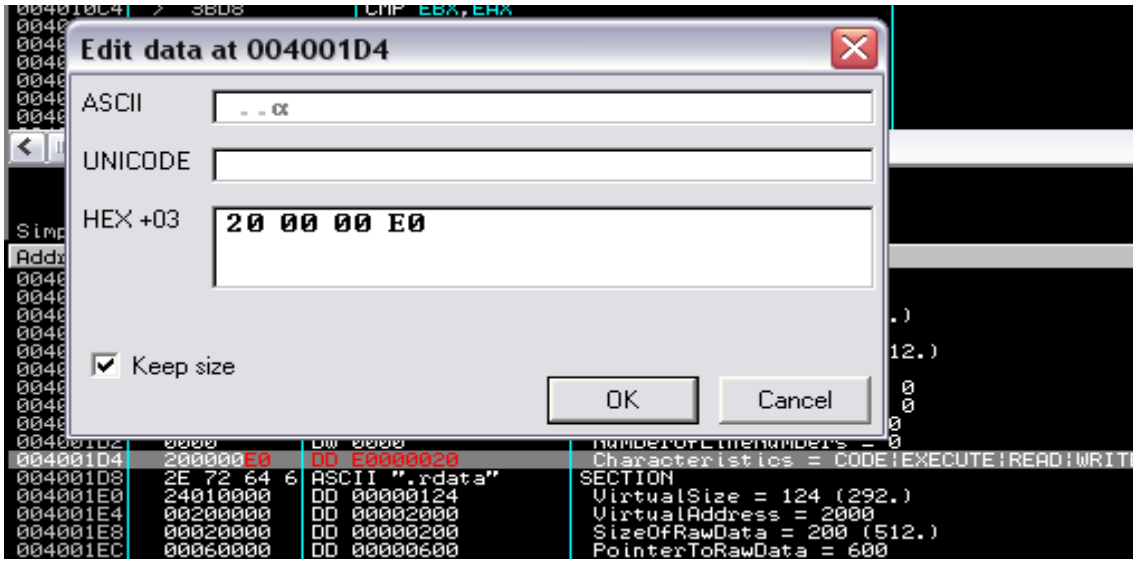
결과 DWORD 40000060이 되었다.(INITIALIZED_DATA| READ - > CODE| INITIALIZED_DATA| READ로 수정)





Step 8

추가로 .text(code) 섹션에서의 수정을 위해 writable flag를 추가시킬 필요가 있다. .text섹션의 characteristics의 위치가 나올때까지 스크롤업하고 > 60000020을 E0000020으로 수정해라. (CODE| EXECUTE| READ - > CODE| EXECUTE| READ| WRITE)



Step 9

실행파일의 original entry point (OEP) 를 패치하고자 하는 코드가 있는 곳으로 바꾸어라

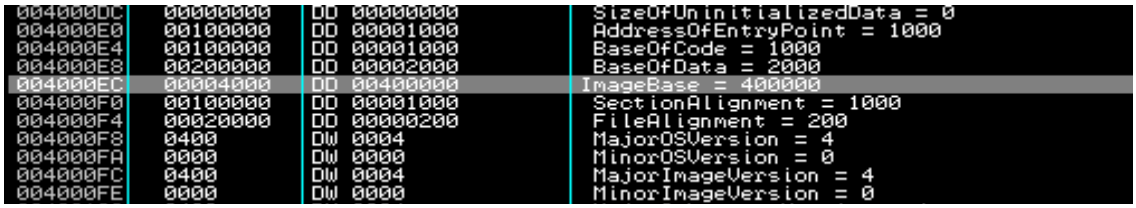
우리의 경우 가상 오프셋(vitual offset)은 .rsrc섹션의 시작지점에서 0x200바이트 떨어진 곳에 위치하며 그 다음이 작업공간을 만들기 위해 0x100바이트를 추가한 곳이다. 우리 코드(**code cave**)의 시작 주소를 아래의 덧셈으로 계산할수 있다.

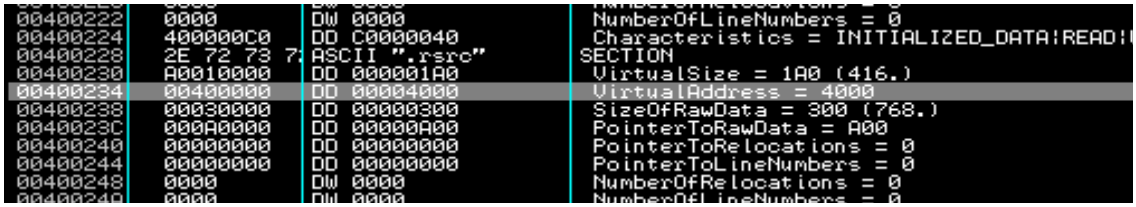
```
Image Base offset + Vitual address of the .rsrc section + 0x200
```

이것은 아래와 동일하다

```
00400000 + 00004000 + 00000200 = 00404200
```

프로그램의 PE헤더로부터 위 변수의 값들을 찾을 수 있다. 아래를 봐라

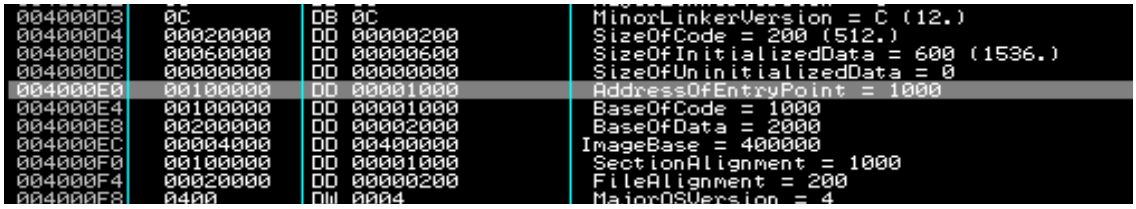


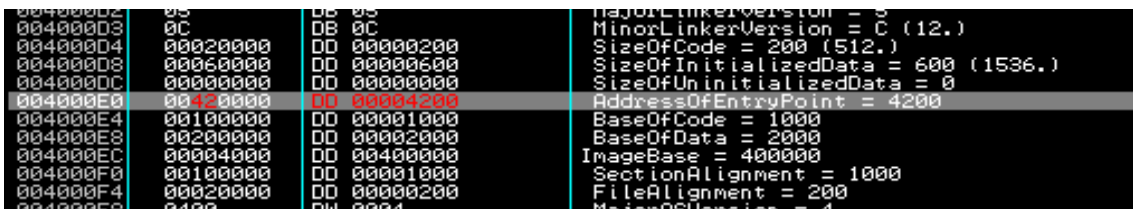


이제 PE헤더의 "AddressOfEntryPoint"의 값을 **code cave**의 오프셋으로 바꾸어라

주의할 점은 이것은 실제의 파일 포인터이며, 즉 Image Base를 포함하지 않는다는 것을 의미한다. 그러므로 Image Base를 우리 **code cave**의 Vitual offset으로부터 뺄 것이고 그 결과를 실제 오프셋에 패치할것이다.

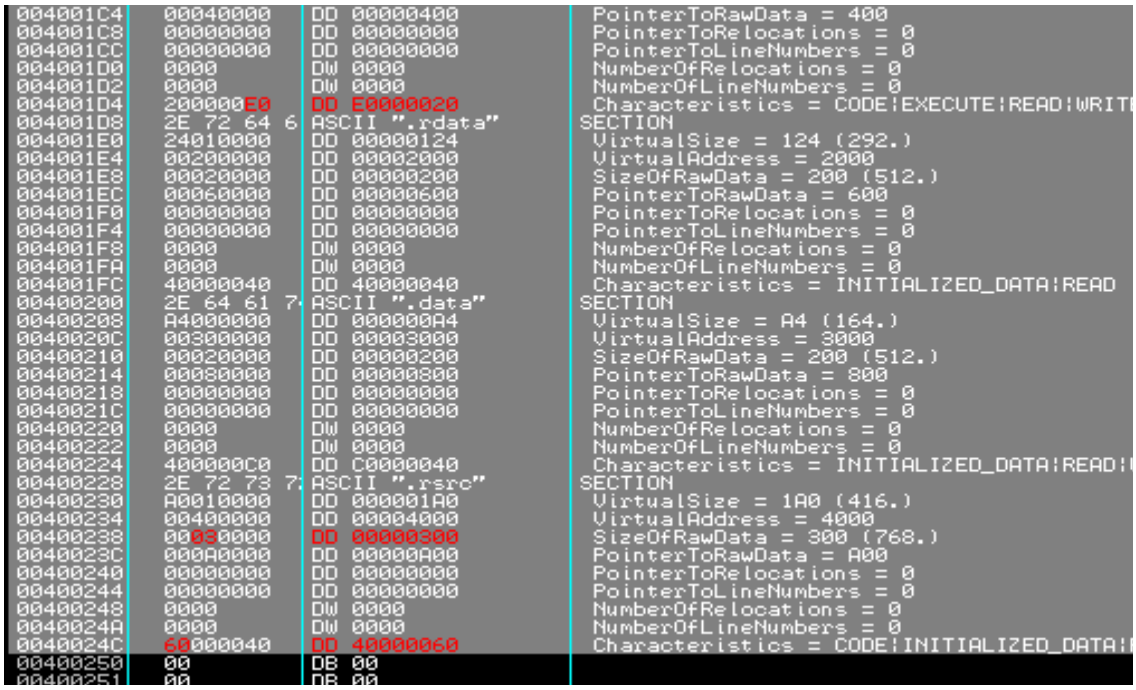
```
404200 – 400000 = 4200
```





step 10

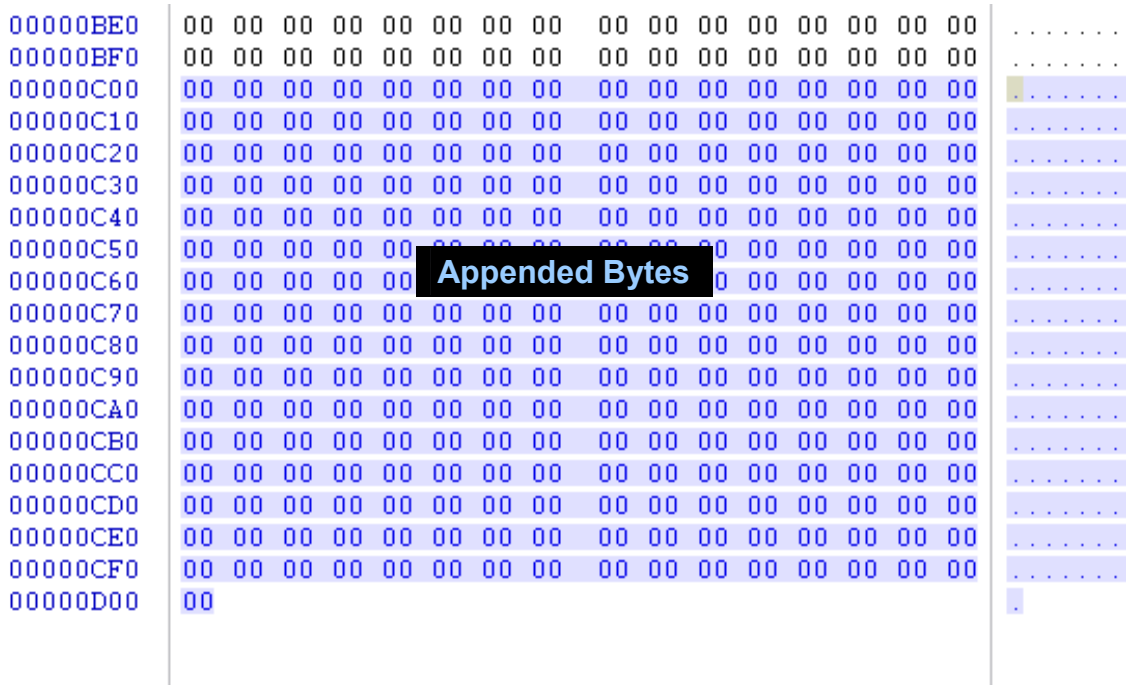
현재까지 수정한 모든것을 선택하고 > 우클릭 > Copy to executable file > 우클릭 > Save file



Step 11

즐겨 사용하는 Hex Editor로 실행파일을 읽고, 0x100(십진수 256)을 더해보라.

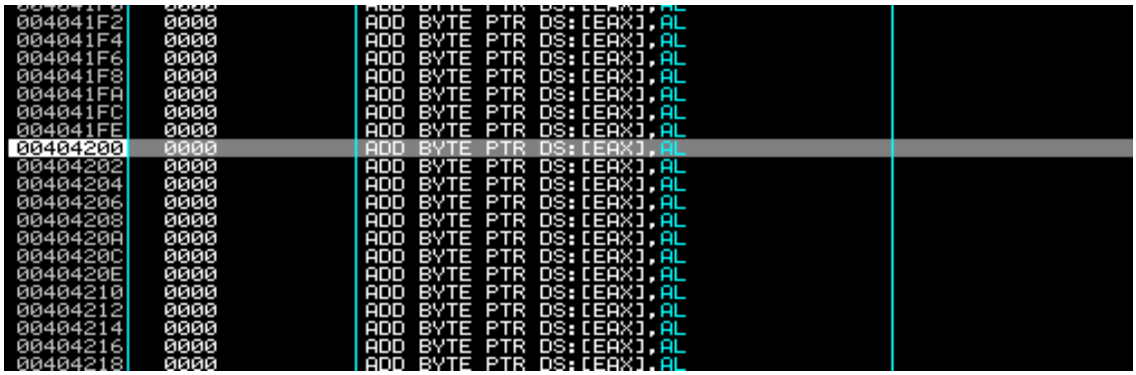
바이트가 정확히 256(0x100)임을 확인하고 아니라면 PE헤더는 유효한게 아닐 것이다.(0xD00 - 0xC00 = 0x100) 새로운 파일로 저장하기 위해 올리디버거를 언로드 해야 한다.



Step 12

만약 올리디버거로 로드했을때 실행파일에 잘못된 바이트를 패치해서 에러를 받든지, 올리엔진의 익숙한 에러를 받았다면 "%ollydir%\ udd"위치에 있는 udd파일을 삭제함으로 고칠 수 있다.

모든게 정상적으로 완료됐으면 CPU윈도우의 엔트리포인트는 아래와 같이 보일것이다.



Step 13

프로그램의 .text(code) 섹션의 암호화를 시키는 코드를 패치 해라. 예를 들면

```
00404200      PUSHAD                                    ;  Backup extended registers to stack
00404201      PUSHFD                                    ;  Backup EFlags to stack
00404202      MOV EAX,OFFSET SimpleCr.<ModuleEntryPoin>  ;  EAX = entry point address
00404207      MOV ECX,SimpleCr.0040110C                 ;  ECX = last address with code
0040420C      XOR EBX,EBX                               ;  EBX xor EBX = 0
0040420E   >  MOV BL,BYTE PTR DS:[ EAX]                 ;  BL = byte pointed by EAX
00404210      ADD BL,10                                 ; Add 10 to the current pointed byte value
00404213      XOR BL,AL                                 ; XOR result with AL
00404215      MOV BYTE PTR DS:[ EAX] ,BL                ;  Store BL into the byte pointed by eax
00404217      INC EAX                                   ;  EAX++
00404218      CMP EAX,ECX
0040421A   ^ JNZ SHORT SimpleCr.0040420E                ;  Jump until EAX = ECX
0040421C      POPFD                                     ;  Restore flags
0040421D      POPAD                                     ;  Restore registers
0040421E      PUSH OFFSET SimpleCr.<ModuleEntryPoint>   ;  Push return address
00404223      RETN                                      ;  Return to initial offset
```

위 코드는 EAX에 .text(code) 섹션의 시작주소를 저장하고(오리지널 엔트리), 실행코드의 마지막 바이트+1의 주소 사이에서 한 바이트씩 모든걸 암호화한다.

*보충 : **EAX**가 가리키는 값을 **0**으로 초기화된 **BL**에 넣고 **10**을 더한거와 **XOR**을 하여 다시 **EAX**가 가리키는곳(시작주소의 바이트)에 복사를 하면서 암호화 시킴. 그리고 **EAX**(시작주소)를 하나씩 증가를 시켜서 실행코드의 마지막주소를 가리킬때까지 **CODE**의 시작주소와 마지막주소의 모든 바이트를 암호화시키는 코드임*

Step 14

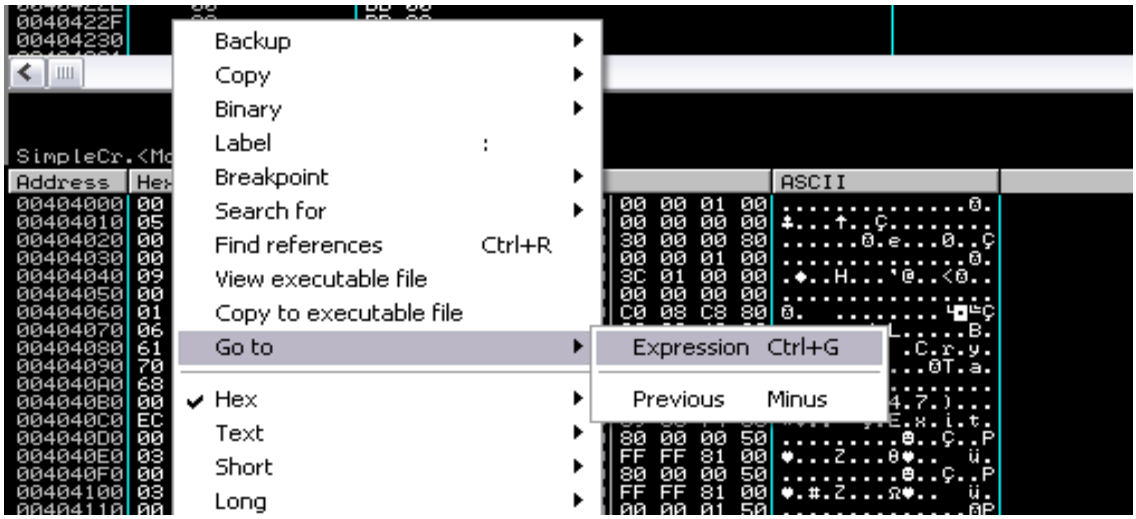
loop문 뒤에 브레이크포인트를 설정하고 프로그램을 실행(F9)

Step 15

브레이크포인트에 성공적으로 다다랐다면 모든게 계획대로 됐다는걸 의미한다. 만약 그러지 않다면 모든 것을 조금씩 다시 체크해야 될것이다.

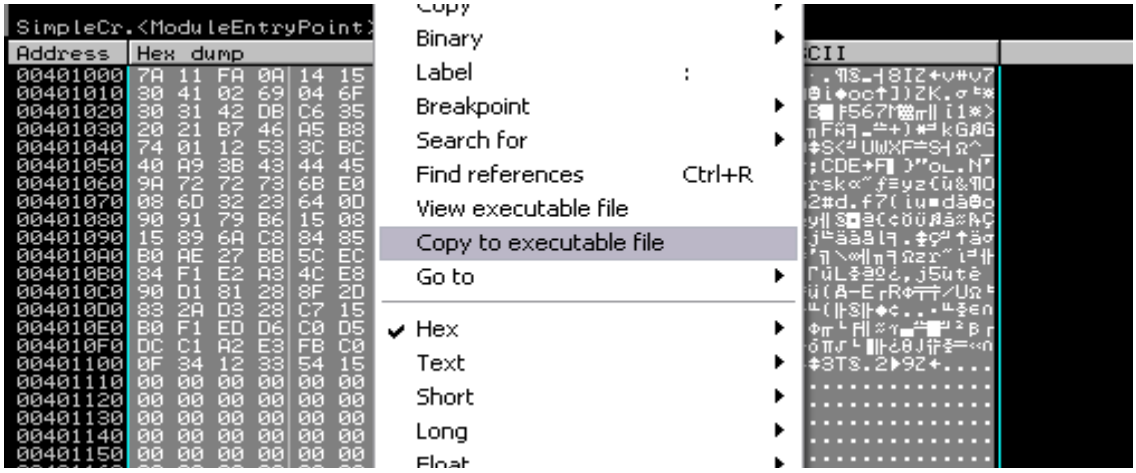
이제 암호화된 .text 섹션을 파일로 저장하려 한다.

덤프 윈도우창에서 우클릭 > Go to > Expression > 00401000(Original Entry Point) 엔터



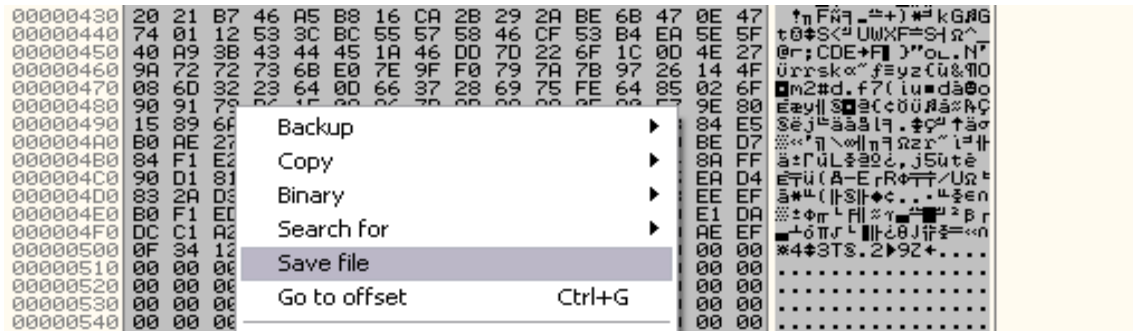
Step 16

덤프윈도우에서 암호화된 바이트를 모두 선택 > 우클릭 > Copy to executable file



Step 17

우클릭 > Save file

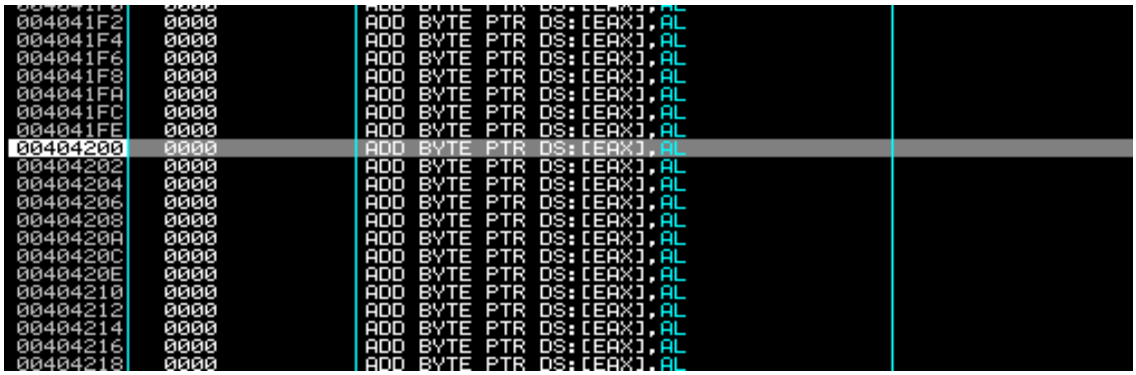


원하는 곳에 파일을 저장해라. 그리고 올리디버거로 로드를 해라(또는 현재 파일이 패치되었다면 reload를 해라)

Step 18

다시 한번 엔트리포인트가 아래와 같이 보일것이다.

(보충 : 암호화하는 코드는 선택을 하지 않았기에 ***404200****에는* ***0000****모두 널값이 들어가 있을 것이다.)*



다음, 우리는 .text(code) 섹션을 복호화해줄 decrypting 코드를 패치해야 한다.

암호화시키는 코드의 중요부분이다(twick ??). 우리가 해야할것은 이 두 opcode를 바꾸어야 하는 것이다.

Encrypt:
ADD BL,10 XOR BL,AL

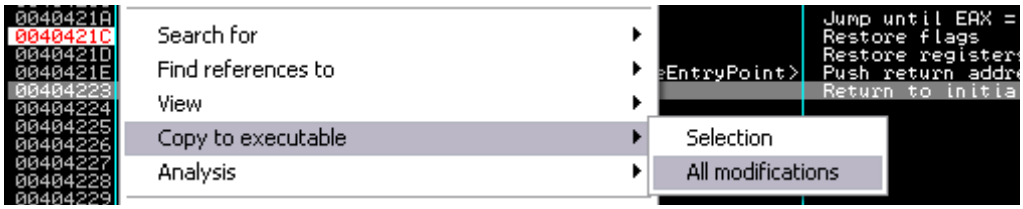
Decrypt:
XOR BL,AL SUB BL,10

우리의 decrypting code는 아래와 같다.

```
00404200      PUSHAD                                          ;  확장레지스터를 스택에 백업
00404201      PUSHFD                                          ;  EFlags를 스택에 백업
00404202      MOV EAX,OFFSET SimpleCr.<ModuleEntryPoin>       ;  EAX = entry point address
00404207      MOV ECX,SimpleCr.0040110C                       ;  ECX = last address with code
0040420C      XOR EBX,EBX                                     ;  EBX xor EBX = 0
0040420E  >   MOV BL,BYTE PTR DS:[ EAX]                      ;  BL = byte pointed by EAX
00404210      XOR BL,AL                                       ; XOR current pointed byte value with AL
00404212      SUB BL,10                                       ; Subtract 10 from the result
00404215      MOV BYTE PTR DS:[ EAX] ,BL                      ;  Store BL into the byte pointed by eax
00404217      INC EAX                                         ;  EAX++
00404218      CMP EAX,ECX
0040421A  ^ JNZ SHORT SimpleCr.0040420E                       ;  Jump until EAX = ECX
0040421C      POPFD                                           ;  Restore flags
0040421D      POPAD                                           ;  Restore registers
0040421E      PUSH OFFSET SimpleCr.<ModuleEntryPoint>     ;  Push return address
00404223      RETN                                            ;  Return to initial offset
```

Step 19

모든 변화를 파일에 저장해라, 우클릭 > Analyze This > 우클릭 > Copy to executable > All modifications > Copy all



원하는 곳에 저장

Step 20

암호화된 파일을 실행

Final Words

실행 파일의 코드섹션을 암호화하는 방법에 대해 문서화했다.

단지 교육목적으로 사용되길 권한다. 안티바이러스의 시그니쳐 검사를 우회하기 위한 기본적인 방법을 보여준다. 비록, 안티바이러스제품이 데이터섹션과 같은 PE파일의 다른 섹션 역시 체크할수도 있기에 탐지를 피하기 위해서는 목표가 될 섹션의 범위를 넓힐 필요가 있다. 마지막으로 뭔가 빠지거나, 설명을 원하거나, 개인적인 목적으로 이 문서의 내용을 사용할 필요가 있다면 편히 생각하고 메일을 보내주기 바란다.