the hacking & security community
ASTALAVISTA

Title:

Reverse Engineering: Smashing the Signature

Date:
August 19th 2008
Website:
http://www.astalavista.com

Author:
Nicolaou George
Mail:
ishtus@astalavista.com

Author:
Charalambous Glafkos
Mail:
glafkos@astalavista.com

Table of Contents

Introduction

Many antivirus and antispyware solutions identify malicious programs by looking for known unique signatures contained inside them. Those signatures are stored inside a database which is constantly updated. This tutorial guides you through a number of steps to encrypt the executable file code section in order to render antivirus signature checking techniques ineffective against identifying the malicious code.

Tools

The tools used in this paper are the following:

- 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(?)
```

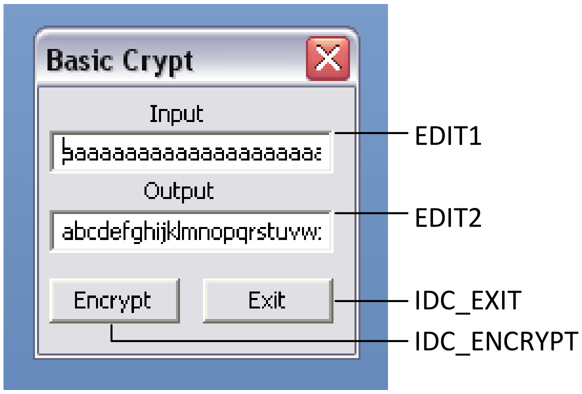
simcrypt.rc

```
;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



aaaaaaaaaaaaaaaaaaaaa

abcdefghijklmnopqrstuvw:

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
```

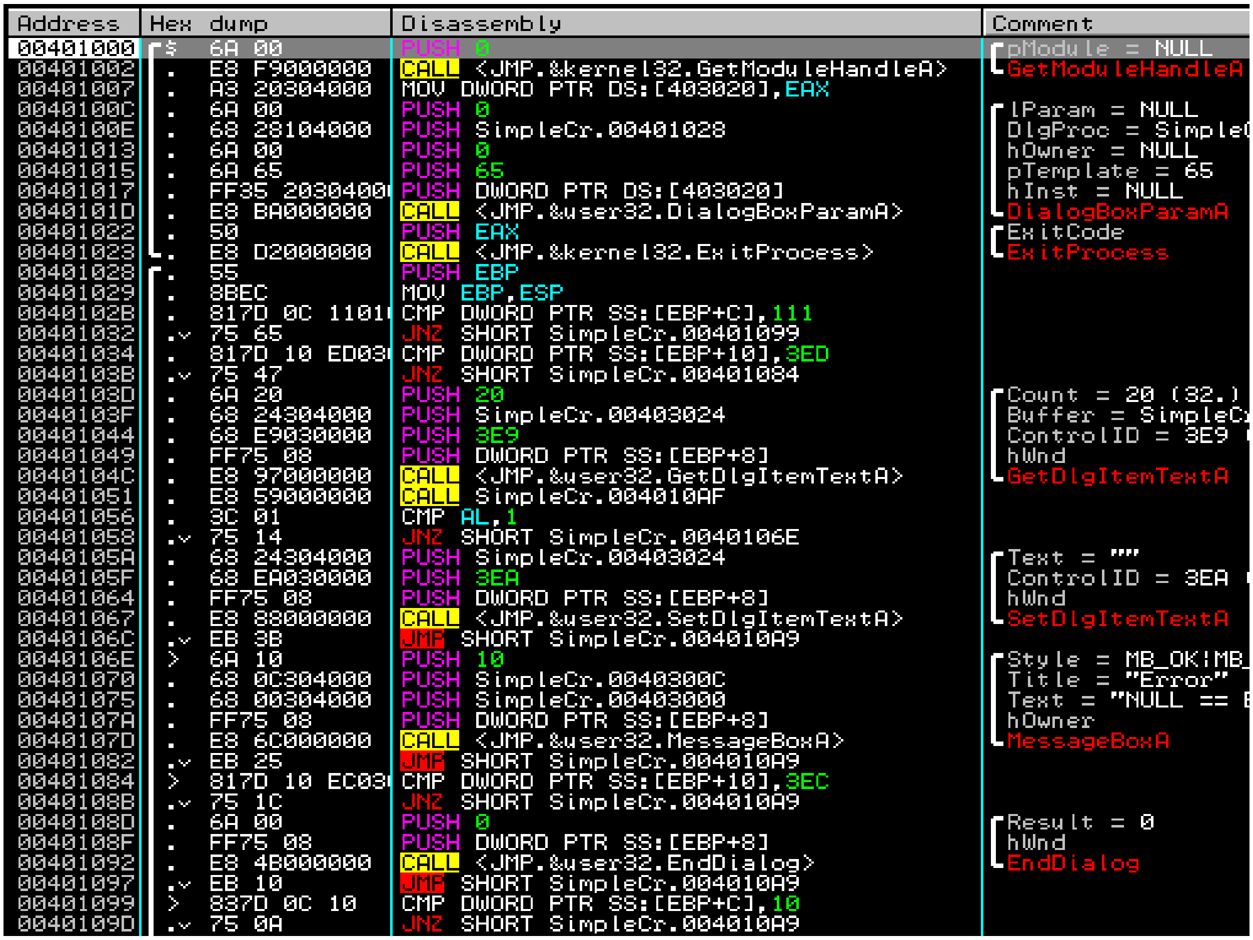
Binary Code Encryption

The idea of encrypting your binary code is simple. The binary code of your software is vulnerable towards static disassembly. In order to avoid that, your code has to be stored in an encrypted form and decrypted on runtime. Additionally, this technique is a simple way of bypassing most antivirus systems. By just changing the code section, you change the signature of your program and therefore making it undetectable.

Although the theory is quite simple, creating a working example might have a level of difficulty on understanding the techniques used. Therefore additional info will be provided.

Step 1

Fire up your olly debugger and load your target. Your ollydbg's CPU windows should look similar to this



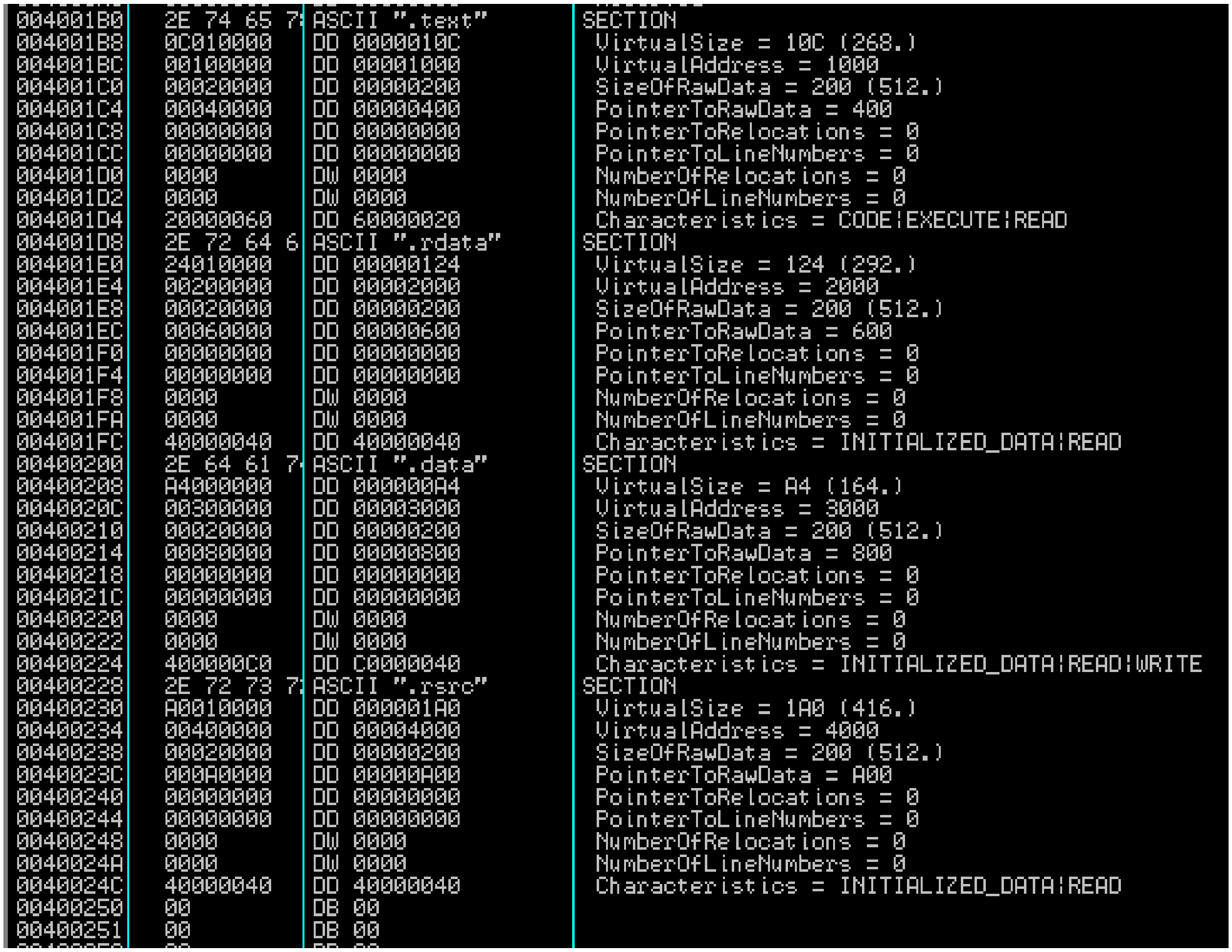
```
Address   Hex dump          Disassembly                                   Comment
00401000  $  6A 00          PUSH 0                                        pModule = NULL
00401002  .  E8 F9000000    CALL <JMP.&kernel32.GetModuleHandleA>         GetModuleHandleA
00401007  .  A3 20304000    MOV DWORD PTR DS:[403020],EAX
0040100C  .  6A 00          PUSH 0                                        lParam = NULL
0040100E  .  68 28104000    PUSH SimpleCr.00401028                        DlgProc = Simple
00401013  .  6A 00          PUSH 0                                        hOwner = NULL
00401015  .  6A 65          PUSH 65                                       pTemplate = 65
00401017  .  FF35 2030400   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 1101   CMP DWORD PTR SS:[EBP+C],111
00401032  .v 75 65          JNZ SHORT SimpleCr.00401099
00401034  .  817D 10 ED03   CMP DWORD PTR SS:[EBP+10],3ED
0040103B  .v 75 47          JNZ SHORT SimpleCr.00401084
0040103D  .  6A 20          PUSH 20                                       Count = 20 (32.)
0040103F  .  68 24304000    PUSH SimpleCr.00403024                        Buffer = SimpleC
00401044  .  68 E9030000    PUSH 3E9                                      ControlID = 3E9
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  .v 75 14          JNZ SHORT SimpleCr.0040106E
0040105A  .  68 24304000    PUSH SimpleCr.00403024                        Text = ""
0040105F  .  68 EA030000    PUSH 3EA                                      ControlID = 3EA
00401064  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]                     hWnd
00401067  .  E8 88000000    CALL <JMP.&user32.SetDlgItemTextA>            SetDlgItemTextA
0040106C  .v EB 3B          JMP SHORT SimpleCr.004010A9
0040106E  >  6A 10          PUSH 10                                       Style = MB_OK|MB_
00401070  .  68 0C304000    PUSH SimpleCr.0040300C                        Title = "Error"
00401075  .  68 00304000    PUSH SimpleCr.00403000                        Text = "NULL == 
0040107A  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]                     hOwner
0040107D  .  E8 6C000000    CALL <JMP.&user32.MessageBoxA>                MessageBoxA
00401082  .v EB 25          JMP SHORT SimpleCr.004010A9
00401084  >  817D 10 EC03   CMP DWORD PTR SS:[EBP+10],3EC
0040108B  .v 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  .v EB 10          JMP SHORT SimpleCr.004010A9
00401099  >  837D 0C 10     CMP DWORD PTR SS:[EBP+C],10
0040109D  .v 75 0A          JNZ SHORT SimpleCr.004010A9
```

Step 2

In the case where the size of the code you intend to patch is greater than the raw size of the data section you are patching at, or because it is wiser, you will need to modify the PE header in order to make some room to work with. That room we will be creating is referred as a “cove cave”.

Every Windows executable file contains a PE header. That header contains information like:

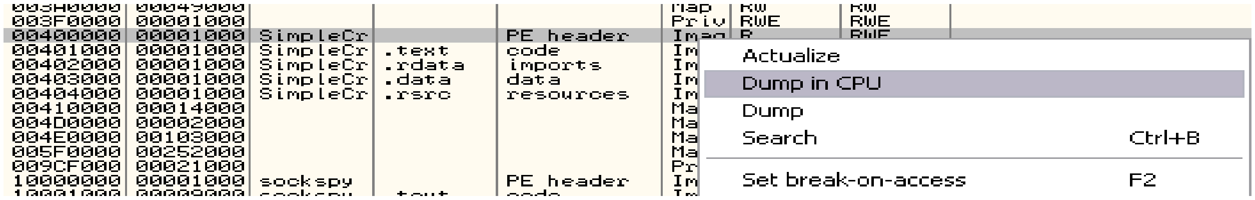
- Time and Date Stamp
- Checksum
- The address of the executable entry point (EP). In our case this is the Original Entry Point of our code (OEP) since we will overwrite this address later on.
- Section Headers (see image below)



```
004001B0   2E 74 65 7  ASCII ".text"       SECTION
004001B8   0C010000    DD 0000010C          VirtualSize = 10C (268.)
004001BC   00100000    DD 00001000          VirtualAddress = 1000
004001C0   00020000    DD 00000200          SizeOfRawData = 200 (512.)
004001C4   00040000    DD 00000400          PointerToRawData = 400
004001C8   00000000    DD 00000000          PointerToRelocations = 0
004001CC   00000000    DD 00000000          PointerToLineNumbers = 0
004001D0   0000        DW 0000              NumberOfRelocations = 0
004001D2   0000        DW 0000              NumberOfLineNumbers = 0
004001D4   20000060    DD 60000020          Characteristics = CODE|EXECUTE|READ
004001D8   2E 72 64 6  ASCII ".rdata"      SECTION
004001E0   24010000    DD 00000124          VirtualSize = 124 (292.)
004001E4   00200000    DD 00002000          VirtualAddress = 2000
004001E8   00020000    DD 00000200          SizeOfRawData = 200 (512.)
004001EC   00060000    DD 00000600          PointerToRawData = 600
004001F0   00000000    DD 00000000          PointerToRelocations = 0
004001F4   00000000    DD 00000000          PointerToLineNumbers = 0
004001F8   0000        DW 0000              NumberOfRelocations = 0
004001FA   0000        DW 0000              NumberOfLineNumbers = 0
004001FC   40000040    DD 40000040          Characteristics = INITIALIZED_DATA|READ
00400200   2E 64 61 7  ASCII ".data"       SECTION
00400208   A4000000    DD 000000A4          VirtualSize = A4 (164.)
0040020C   00300000    DD 00003000          VirtualAddress = 3000
00400210   00020000    DD 00000200          SizeOfRawData = 200 (512.)
00400214   00080000    DD 00000800          PointerToRawData = 800
00400218   00000000    DD 00000000          PointerToRelocations = 0
0040021C   00000000    DD 00000000          PointerToLineNumbers = 0
00400220   0000        DW 0000              NumberOfRelocations = 0
00400222   0000        DW 0000              NumberOfLineNumbers = 0
00400224   400000C0    DD C0000040          Characteristics = INITIALIZED_DATA|READ|WRITE
00400228   2E 72 73 7  ASCII ".rsrc"       SECTION
00400230   A0010000    DD 000001A0          VirtualSize = 1A0 (416.)
00400234   00400000    DD 00004000          VirtualAddress = 4000
00400238   00020000    DD 00000200          SizeOfRawData = 200 (512.)
0040023C   000A0000    DD 00000A00          PointerToRawData = A00
00400240   00000000    DD 00000000          PointerToRelocations = 0
00400244   00000000    DD 00000000          PointerToLineNumbers = 0
00400248   0000        DW 0000              NumberOfRelocations = 0
0040024A   0000        DW 0000              NumberOfLineNumbers = 0
0040024C   40000040    DD 40000040          Characteristics = INITIALIZED_DATA|READ
00400250   00          DB 00
00400251   00          DB 00
```

Each section header above defines the properties of a section. In order to keep things as simple as possible, we avoid increasing the size of sections that reside between other sections. Therefore we will be increasing the size of the .rsrc section, which is located at the end of the file.

Go to the Memory window (Alt+M) > Right Click on the PE header > Select Dump in CPU

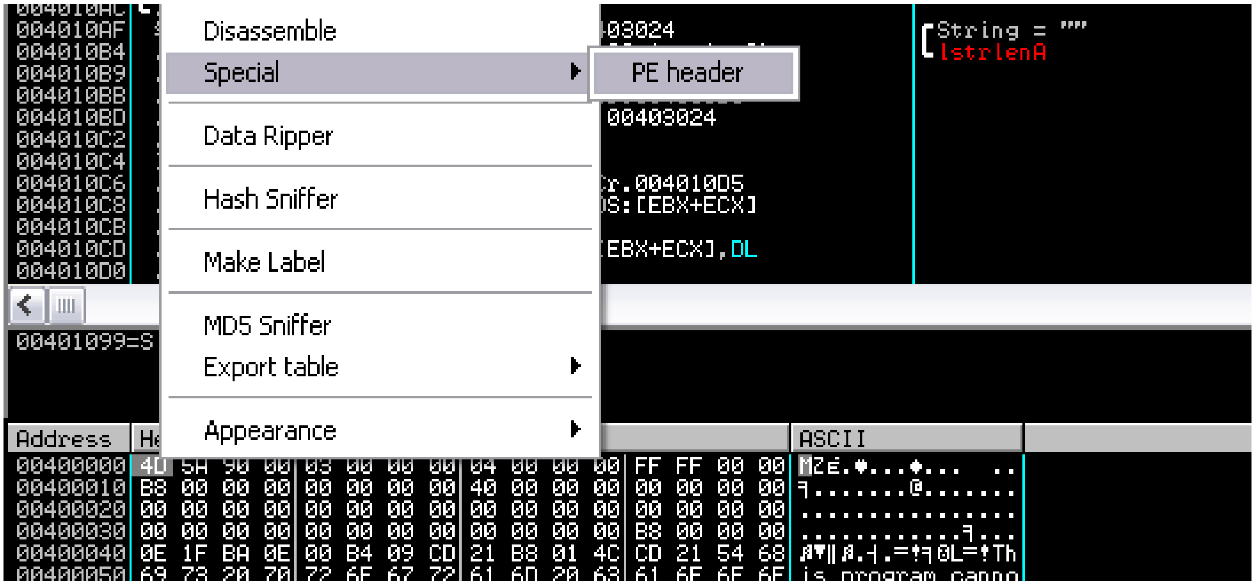


```
003F0000 00001000                                       Priv RWE     RWE
00400000 00001000 SimpleCr           PE header          Imag R       RWE
00401000 00001000 SimpleCr .text     code               Im
00402000 00001000 SimpleCr .rdata    imports            Im
00403000 00001000 SimpleCr .data     data               Im
00404000 00001000 SimpleCr .rsrc     resources          Im
00410000 00014000                                       Ma
004D0000 00002000                                       Ma
004E0000 00103000                                       Ma
005F0000 00252000                                       Ma
009CF000 00021000                                       Pr
10000000 00001000 sockspy            PE header          Im
```

Step 3

Modify the dump to treat this section as a PE Header

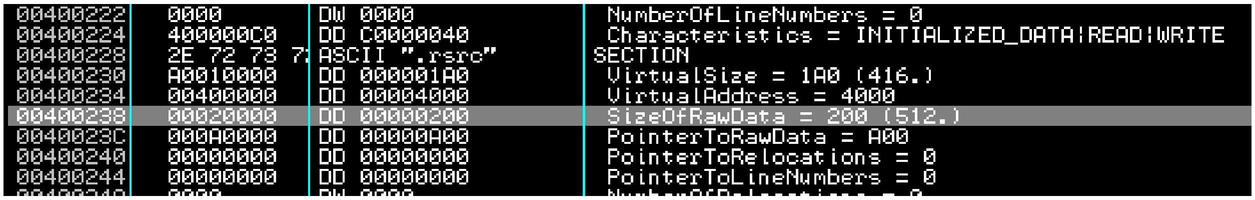
Right Click at the dump window > Special > PE Header



```
Address  Hex                                              ASCII
00400000 4D 5A 90 00|03 00 00 00|04 00 00 00|FF FF 00 00| MZÉ.♥...♦...  ..
00400010 B8 00 00 00|00 00 00 00|40 00 00 00|00 00 00 00| ╕.......@.......
00400020 00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00| ................
00400030 00 00 00 00|00 00 00 00|00 00 00 00|B8 00 00 00| ............╕...
00400040 0E 1F BA 0E|00 B4 09 CD|21 B8 01 4C|CD 21 54 68| ♬▼║♬.┤.═!╕☺L═!Th
```

Step 4

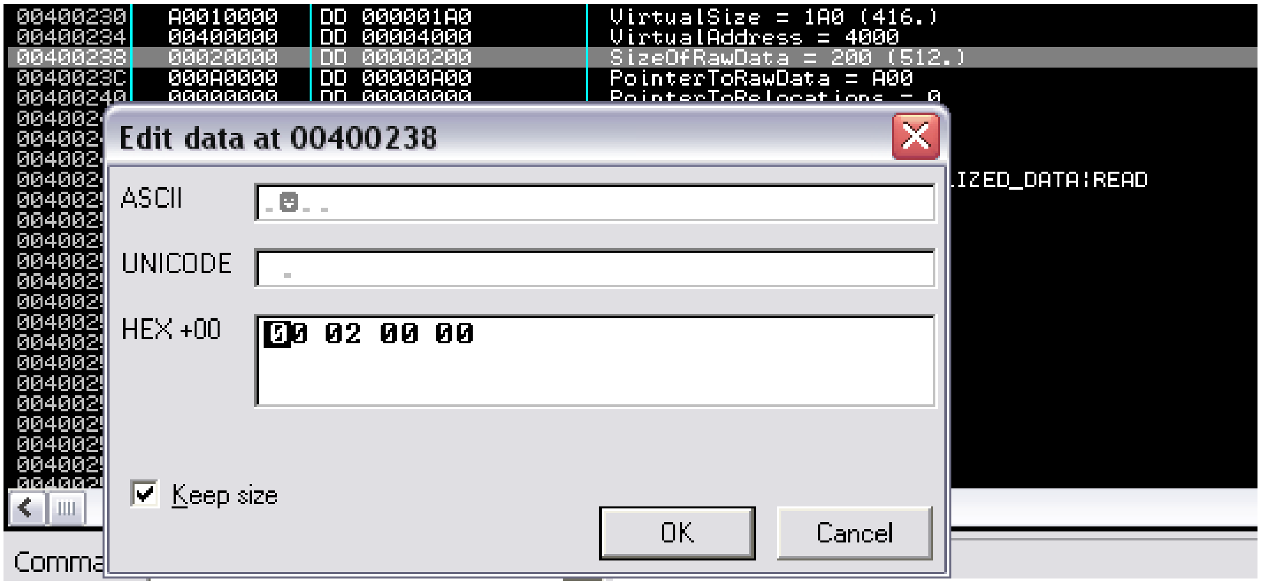
Scroll down until you find the “SizeOfRawData” option inside the .rsrc section.



```
00400222   0000         DW 0000            NumberOfLineNumbers = 0
00400224   400000C0     DD C0000040        Characteristics = INITIALIZED_DATA|READ|WRITE
00400228   2E 72 73 7:  ASCII ".rsrc"     SECTION
00400230   A0010000     DD 000001A0        VirtualSize = 1A0 (416.)
00400234   00400000     DD 00004000        VirtualAddress = 4000
00400238   00020000     DD 00000200        SizeOfRawData = 200 (512.)
0040023C   000A0000     DD 00000A00        PointerToRawData = A00
00400240   00000000     DD 00000000        PointerToRelocations = 0
00400244   00000000     DD 00000000        PointerToLineNumbers = 0
```

Step 5

Press Ctrl+E or Right Click > Binary > Edit, to binary edit the size of the .rsrc section



```
00400230   A0010000   DD 000001A0   VirtualSize = 1A0 (416.)
00400234   00400000   DD 00004000   VirtualAddress = 4000
00400238   00020000   DD 00000200   SizeOfRawData = 200 (512.)
0040023C   000A0000   DD 00000A00   PointerToRawData = A00
00400240   00000000   DD 00000000   PointerToRelocations = 0
```

Edit data at 00400238

ASCII: .☻..

UNICODE: .

HEX +00: 00 02 00 00

☑ Keep size

OK Cancel

Note:

Data in the Intel architecture is presented in "little Endian" form this means it is read by the CPU in a reverse order as shown in the table below (1 – 4)

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

=

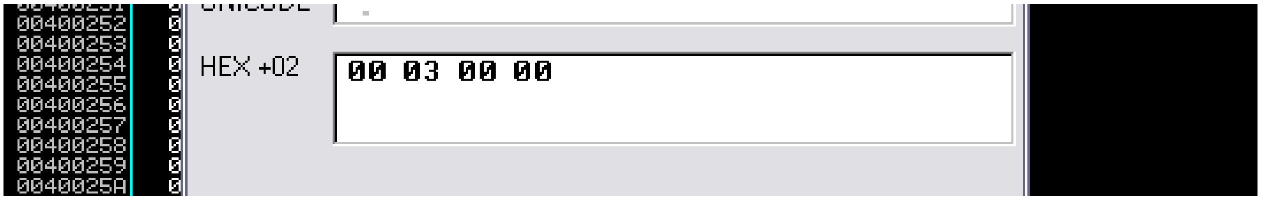
1	***2***	***3***	***4***
00	*00*	*02*	*00*

=

0x200 in hexadecimal (base 16) is equal to 512 decimal (base 10).

Step 6

Add 0x100 (256) bytes to the size of the section (0x200 + 0x100 = 0x300).

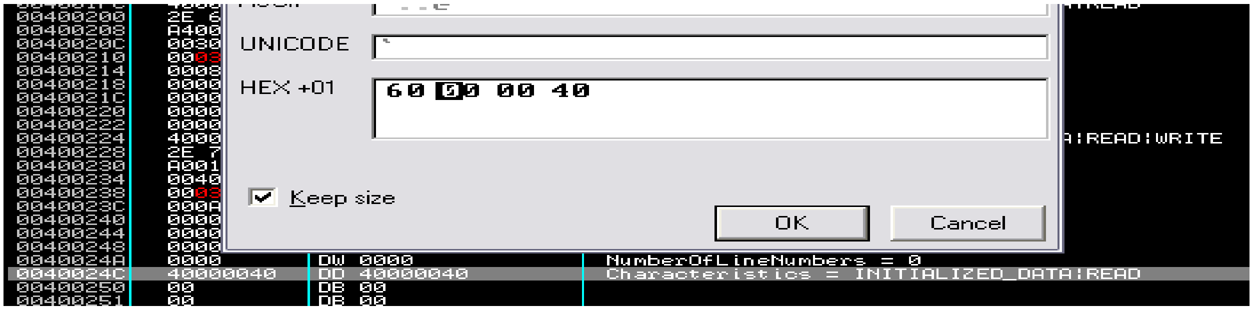


```
00400252
00400253
00400254
00400255
00400256
00400257
00400258
00400259
0040025A
```

HEX +02: 00 03 00 00

Step 7

Edit the flags of this section ("characteristics") to define that it contains executable code. Add to the first byte value of the DWORD 40000040 the byte 0x20 (0x40+ 0x20 = 0x60). The resulting DWORD should be 40000060.



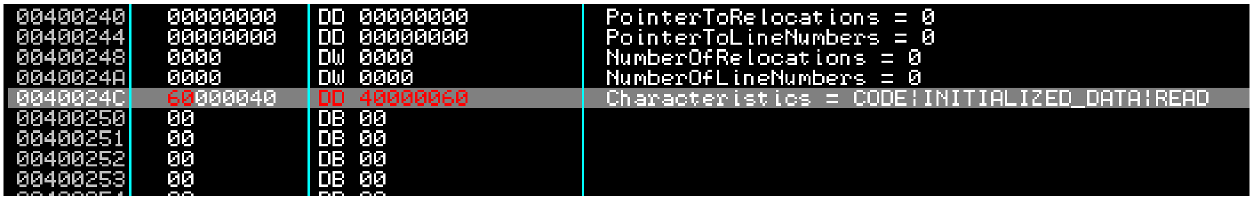
UNICODE

HEX +01

```
60 00 00 40
```

☑ Keep size

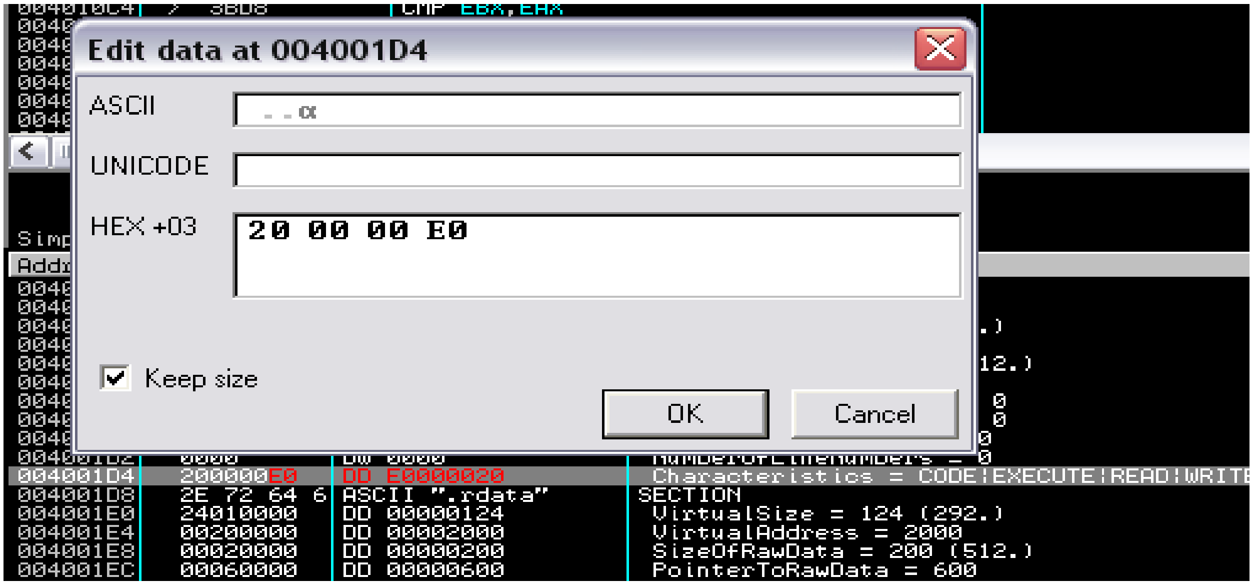
```
00400200  2E 6
00400208  A400
0040020C  0030
00400210  0003
00400214  0008
00400218  0000
0040021C  0000
00400220  0000
00400222  0000
00400224  4000                                              A|READ|WRITE
00400228  2E 7
00400230  A001
00400234  0040
00400238  0003
0040023C  000A
00400240  0000
00400244  0000
00400248  0000
0040024A  0000       DW 0000       NumberOfLineNumbers = 0
0040024C  40000040   DD 40000040   Characteristics = INITIALIZED_DATA|READ
00400250  00         DB 00
```



```
00400240  00000000   DD 00000000   PointerToRelocations = 0
00400244  00000000   DD 00000000   PointerToLineNumbers = 0
00400248  0000       DW 0000       NumberOfRelocations = 0
0040024A  0000       DW 0000       NumberOfLineNumbers = 0
0040024C  60000040   DD 40000060   Characteristics = CODE|INITIALIZED_DATA|READ
00400250  00         DB 00
00400251  00         DB 00
00400252  00         DB 00
00400253  00         DB 00
```

Step 8

Additionally we need to add the writable flag at the .text (code) section, since we intend to modify the bytes in that section. Scroll up and locate the .text section's characteristics > Modify 60000020 to E0000020



Edit data at 004001D4

ASCII

UNICODE

HEX +03

```
20 00 00 E0
```

☑ Keep size

```
004001D4  200000E0      DD E0000020      Characteristics = CODE|EXECUTE|READ|WRITE
004001D8  2E 72 64 6    ASCII ".rdata"   SECTION
004001E0  24010000      DD 00000124       VirtualSize = 124 (292.)
004001E4  00200000      DD 00002000       VirtualAddress = 2000
004001E8  00020000      DD 00000200       SizeOfRawData = 200 (512.)
004001EC  00060000      DD 00000600       PointerToRawData = 600
```

Step 9

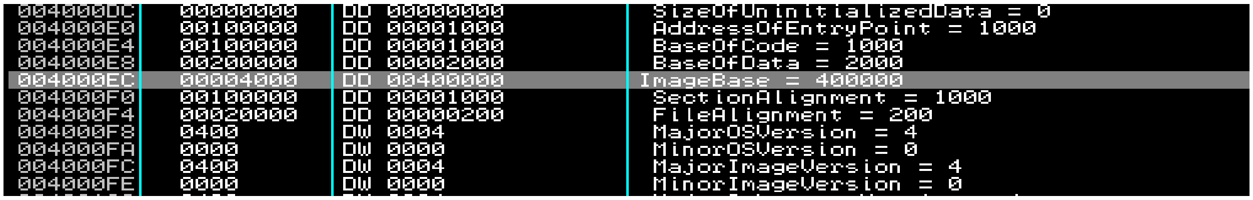
Change the original entry point (OEP) of the executable file with the one we intend to patch our code at. In our case the ***virtual offset*** *is located at the 0x200th byte from the start of the .rsrc section, since we appended 0x100 bytes to that offset in an attempt to create more space to work with. We can calculate the starting point of our code cave by adding together:*

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

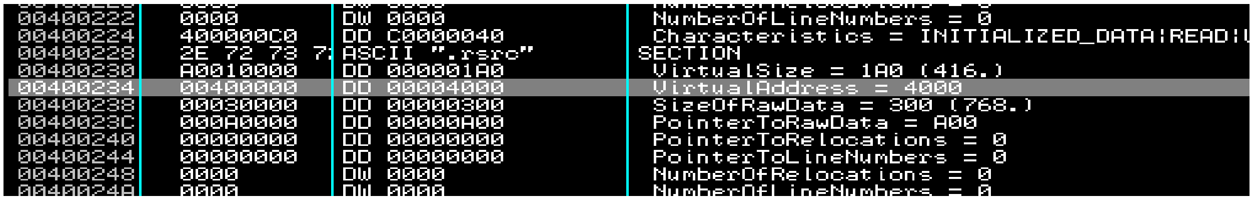
which is equal to:

00400000 + 00004000 + 00000200 = 404200

You can retrieve the value of these variables from the PE Header of your program. As shown below:



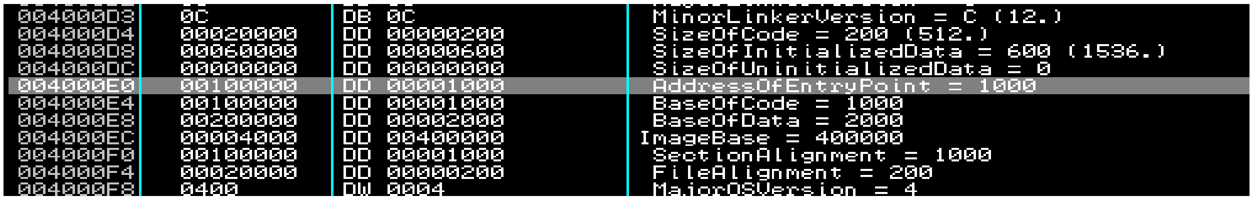
```
004000DC  00000000  DD 00000000   SizeOfUninitializedData = 0
004000E0  00100000  DD 00001000   AddressOfEntryPoint = 1000
004000E4  00100000  DD 00001000   BaseOfCode = 1000
004000E8  00200000  DD 00002000   BaseOfData = 2000
004000EC  00004000  DD 00400000  ImageBase = 400000
004000F0  00100000  DD 00001000   SectionAlignment = 1000
004000F4  00020000  DD 00000200   FileAlignment = 200
004000F8  0400      DW 0004       MajorOSVersion = 4
004000FA  0000      DW 0000       MinorOSVersion = 0
004000FC  0400      DW 0004       MajorImageVersion = 4
004000FE  0000      DW 0000       MinorImageVersion = 0
```



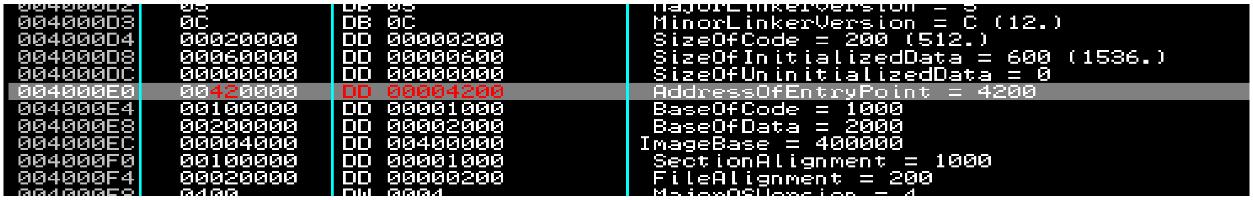
```
00400222  0000       DW 0000        NumberOfLineNumbers = 0
00400224  400000C0   DD C0000040    Characteristics = INITIALIZED_DATA|READ|U
00400228  2E 72 73 7: ASCII ".rsrc" SECTION
00400230  A0010000   DD 000001A0    VirtualSize = 1A0 (416.)
00400234  00400000   DD 00004000    VirtualAddress = 4000
00400238  00030000   DD 00000300    SizeOfRawData = 300 (768.)
0040023C  000A0000   DD 00000A00    PointerToRawData = A00
00400240  00000000   DD 00000000    PointerToRelocations = 0
00400244  00000000   DD 00000000    PointerToLineNumbers = 0
00400248  0000       DW 0000        NumberOfRelocations = 0
0040024A  0000       DW 0000        NumberOfLineNumbers = 0
```

Now replace the "AddressOfEntryPoint" value in the PE Header with the offset of the code cave. Note that this is a raw file pointer value, meaning that it does not include the ImageBase. Therefore we subtract that from the ***virtual offset*** *of our code cave and patch the resulting raw offset.*

404200 – 400000 = 4200



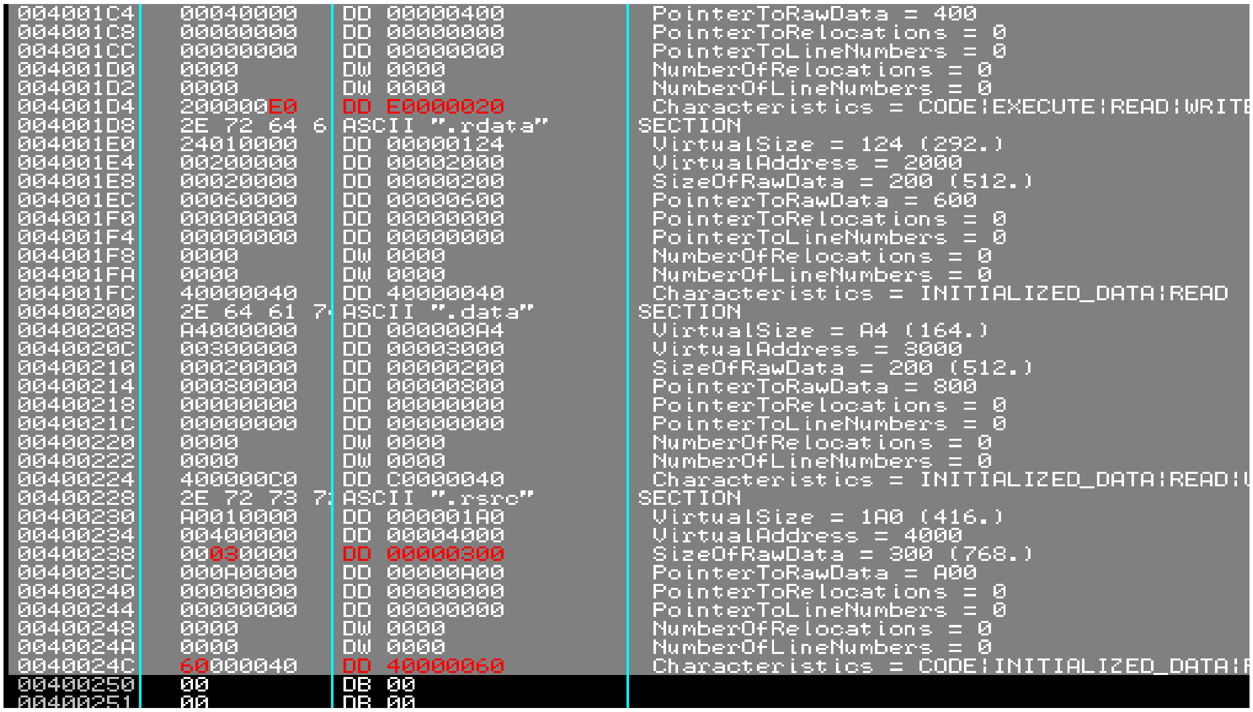
```
004000D3  0C        DB 0C         MinorLinkerVersion = C (12.)
004000D4  00020000  DD 00000200   SizeOfCode = 200 (512.)
004000D8  00060000  DD 00000600   SizeOfInitializedData = 600 (1536.)
004000DC  00000000  DD 00000000   SizeOfUninitializedData = 0
004000E0  00100000  DD 00001000   AddressOfEntryPoint = 1000
004000E4  00100000  DD 00001000   BaseOfCode = 1000
004000E8  00200000  DD 00002000   BaseOfData = 2000
004000EC  00004000  DD 00400000  ImageBase = 400000
004000F0  00100000  DD 00001000   SectionAlignment = 1000
004000F4  00020000  DD 00000200   FileAlignment = 200
004000F8  0400      DW 0004       MajorOSVersion = 4
```



```
004000D2  05        DB 05         MajorLinkerVersion = 5
004000D3  0C        DB 0C         MinorLinkerVersion = C (12.)
004000D4  00020000  DD 00000200   SizeOfCode = 200 (512.)
004000D8  00060000  DD 00000600   SizeOfInitializedData = 600 (1536.)
004000DC  00000000  DD 00000000   SizeOfUninitializedData = 0
004000E0  00420000  DD 00004200   AddressOfEntryPoint = 4200
004000E4  00100000  DD 00001000   BaseOfCode = 1000
004000E8  00200000  DD 00002000   BaseOfData = 2000
004000EC  00004000  DD 00400000  ImageBase = 400000
004000F0  00100000  DD 00001000   SectionAlignment = 1000
004000F4  00020000  DD 00000200   FileAlignment = 200
```

Step 10

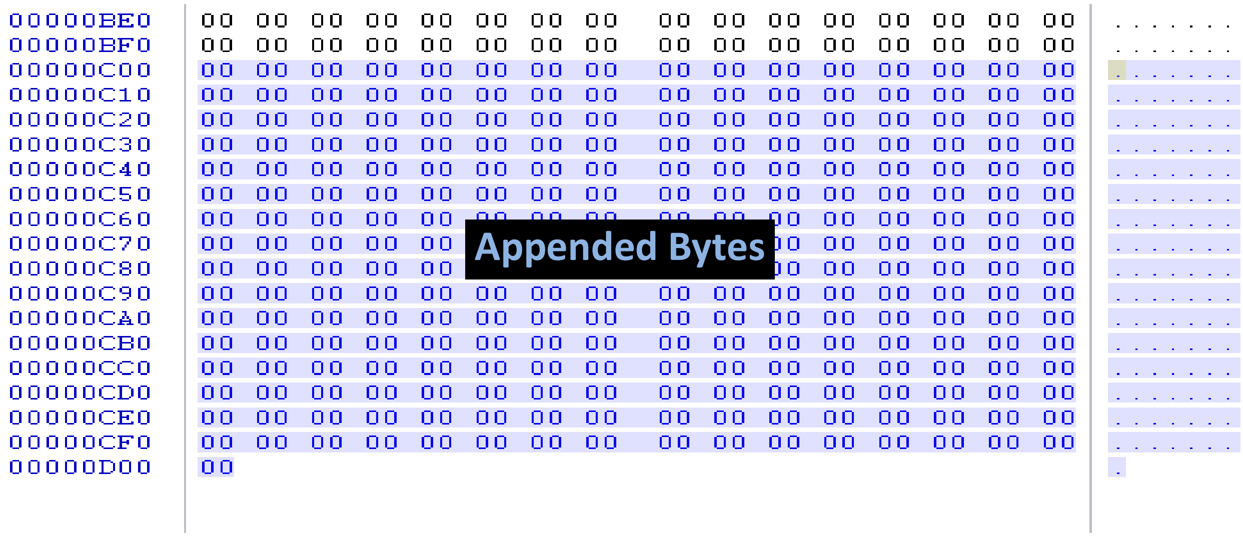
Select the everything you have modified until now > Right click > Copy to executable file, then Right Click > Save file



```
004001C4   00040000      DD 00000400        PointerToRawData = 400
004001C8   00000000      DD 00000000        PointerToRelocations = 0
004001CC   00000000      DD 00000000        PointerToLineNumbers = 0
004001D0   0000          DW 0000            NumberOfRelocations = 0
004001D2   0000          DW 0000            NumberOfLineNumbers = 0
004001D4   200000E0      DD E0000020        Characteristics = CODE|EXECUTE|READ|WRITE
004001D8   2E 72 64 6    ASCII ".rdata"   SECTION
004001E0   24010000      DD 00000124        VirtualSize = 124 (292.)
004001E4   00200000      DD 00002000        VirtualAddress = 2000
004001E8   00020000      DD 00000200        SizeOfRawData = 200 (512.)
004001EC   00060000      DD 00000600        PointerToRawData = 600
004001F0   00000000      DD 00000000        PointerToRelocations = 0
004001F4   00000000      DD 00000000        PointerToLineNumbers = 0
004001F8   0000          DW 0000            NumberOfRelocations = 0
004001FA   0000          DW 0000            NumberOfLineNumbers = 0
004001FC   40000040      DD 40000040        Characteristics = INITIALIZED_DATA|READ
00400200   2E 64 61 7    ASCII ".data"    SECTION
00400208   A4000000      DD 000000A4        VirtualSize = A4 (164.)
0040020C   00300000      DD 00003000        VirtualAddress = 3000
00400210   00020000      DD 00000200        SizeOfRawData = 200 (512.)
00400214   00080000      DD 00000800        PointerToRawData = 800
00400218   00000000      DD 00000000        PointerToRelocations = 0
0040021C   00000000      DD 00000000        PointerToLineNumbers = 0
00400220   0000          DW 0000            NumberOfRelocations = 0
00400222   0000          DW 0000            NumberOfLineNumbers = 0
00400224   400000C0      DD C0000040        Characteristics = INITIALIZED_DATA|READ|W
00400228   2E 72 73 7    ASCII ".rsrc"    SECTION
00400230   A0010000      DD 000001A0        VirtualSize = 1A0 (416.)
00400234   00400000      DD 00004000        VirtualAddress = 4000
00400238   00030000      DD 00000300        SizeOfRawData = 300 (768.)
0040023C   000A0000      DD 00000A00        PointerToRawData = A00
00400240   00000000      DD 00000000        PointerToRelocations = 0
00400244   00000000      DD 00000000        PointerToLineNumbers = 0
00400248   0000          DW 0000            NumberOfRelocations = 0
0040024A   0000          DW 0000            NumberOfLineNumbers = 0
0040024C   60000040      DD 40000060        Characteristics = CODE|INITIALIZED_DATA|R
00400250   00            DB 00
00400251   00            DB 00
```

Step 11

Open the executable file with your favorite hex editor and add 0x100 (256 decimal) bytes. Make sure the bytes are exactly 256(0x100) or else the PE header will not be valid (0xD00 – 0xC00 = 0x100). Note that you might have to unload olly in order to save the new file.

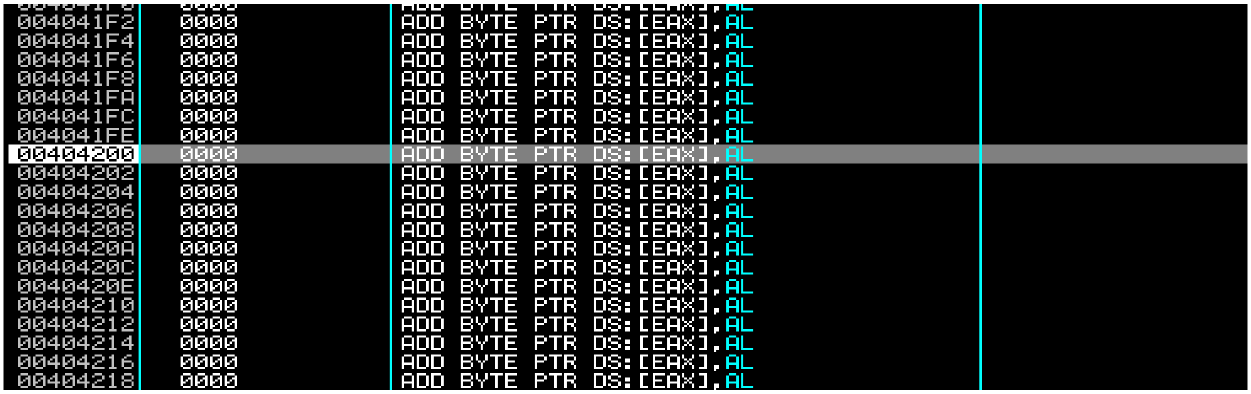


```
00000BE0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000BF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000C00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000C10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000C20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000C30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000C40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000C50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000C60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000C70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000C80  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000C90  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000CA0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000CB0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000CC0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000CD0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000CE0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000CF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........
00000D00  00                                                .
```

Step 12

Load your target with olly if you receive an error then you've either patched the wrong number of bytes to the executable file or you have just experienced a bug in the ollydbg engine. You can fix this by deleting the .udd file of the executable located at"%ollydir%\udd.

If everything went good then the Entry Point in your CPU window should look similar to this:



```
004041F2   0000    ADD BYTE PTR DS:[EAX],AL
004041F4   0000    ADD BYTE PTR DS:[EAX],AL
004041F6   0000    ADD BYTE PTR DS:[EAX],AL
004041F8   0000    ADD BYTE PTR DS:[EAX],AL
004041FA   0000    ADD BYTE PTR DS:[EAX],AL
004041FC   0000    ADD BYTE PTR DS:[EAX],AL
004041FE   0000    ADD BYTE PTR DS:[EAX],AL
00404200   0000    ADD BYTE PTR DS:[EAX],AL
00404202   0000    ADD BYTE PTR DS:[EAX],AL
00404204   0000    ADD BYTE PTR DS:[EAX],AL
00404206   0000    ADD BYTE PTR DS:[EAX],AL
00404208   0000    ADD BYTE PTR DS:[EAX],AL
0040420A   0000    ADD BYTE PTR DS:[EAX],AL
0040420C   0000    ADD BYTE PTR DS:[EAX],AL
0040420E   0000    ADD BYTE PTR DS:[EAX],AL
00404210   0000    ADD BYTE PTR DS:[EAX],AL
00404212   0000    ADD BYTE PTR DS:[EAX],AL
00404214   0000    ADD BYTE PTR DS:[EAX],AL
00404216   0000    ADD BYTE PTR DS:[EAX],AL
00404218   0000    ADD BYTE PTR DS:[EAX],AL
```

Step 13

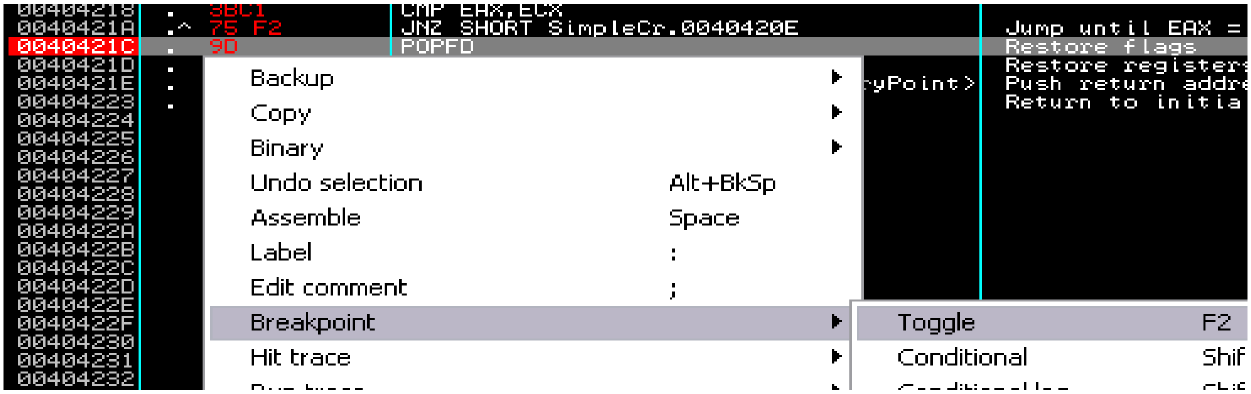
Patch your code responsible for encrypting the .text (code) section of the program. For example:

```
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
```

The code above stores in EAX the starting address of our .text (code) section (the module original entry point), the address of the last byte+1 of executable code and then encrypts everything between them one byte at the time.

Step 14

Set a breakpoint right after the loop and let the program run (press F9)



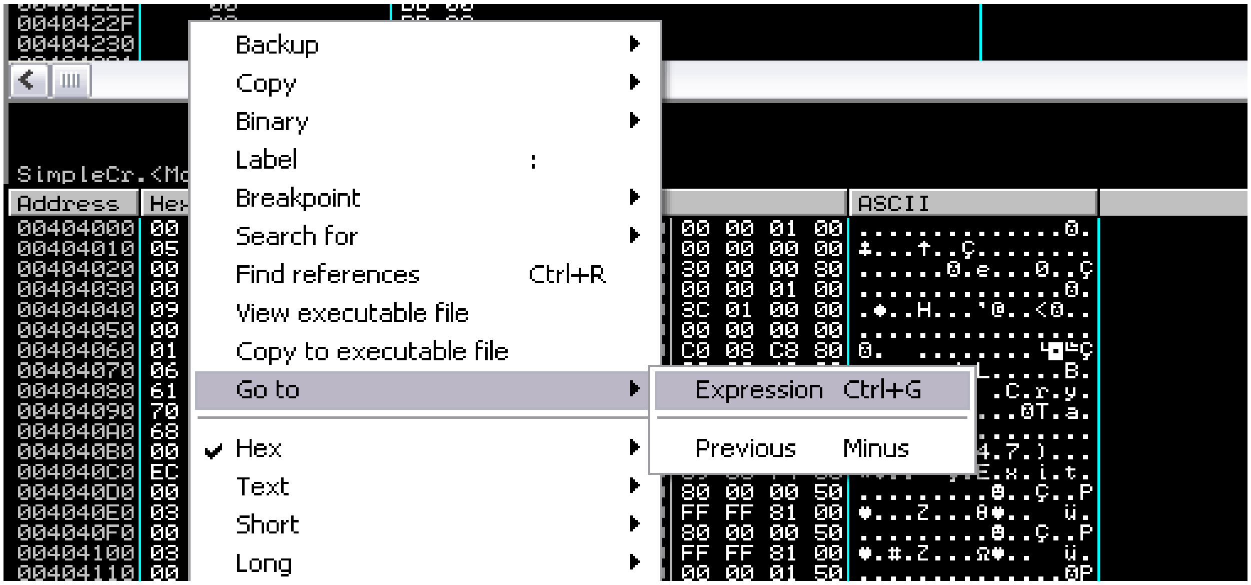
```
00404218  .   3BC1     CMP EAX,ECX
0040421A  .^  75 F2    JNZ SHORT SimpleCr.0040420E      Jump until EAX =
0040421C  .   9D       POPFD                            Restore flags
0040421D  .                                             Restore register
0040421E  .                                ryPoint>     Push return addre
00404223  .                                             Return to initia
00404224
00404225
00404226
00404227
00404228
00404229
0040422A
0040422B
0040422C
0040422D
0040422E
0040422F
00404230
00404231
00404232
```

Step 15

If the breakpoint is successfully reached then it means that everything went as planned. If not, then you should go back a few steps and recheck everything.

We now need to save the encrypted .text section to the file.

Right click at the dump window > Go to > Expression > Enter 00401000 which is the offset of the Original Entry Point (OEP).

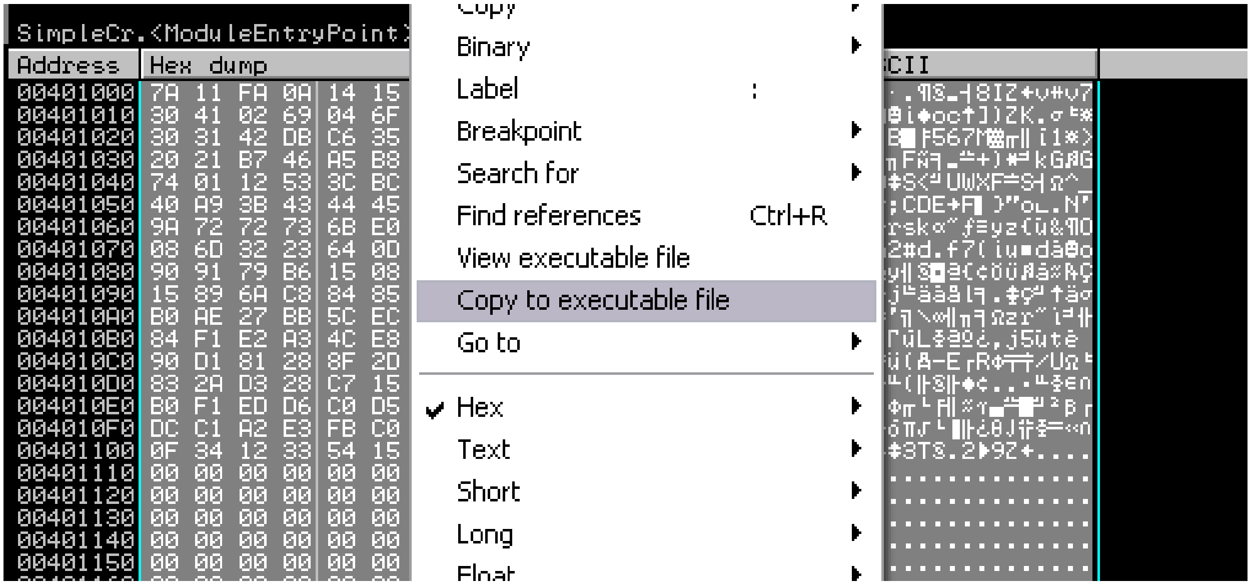


```
0040422F
00404230

SimpleCr.<Mo
Address   Hex                                         ASCII
00404000  00                             00 00 01 00  ..............0.
00404010  05                             00 00 00 00  ♣...↑..Ç.......
00404020  00                             30 00 00 80  ......0.e...0..Ç
00404030  00                             00 00 01 00  ..............0.
00404040  09                             3C 01 00 00  .◆..H...'@..<0..
00404050  00                             00 00 00 00  ................
00404060  01                             C0 08 C8 80  0.  .........╚Ç
00404070  06                                          L.....B.
00404080  61                                          .C.r.y.
00404090  70                                          ...0T.a.
004040A0  68                                          ........
004040B0  00                                          4.7.)...
004040C0  EC                                          E.x.i.t.
004040D0  00                             80 00 00 50  .........@..Ç..P
004040E0  03                             FF FF 81 00  ♥...Z...θ♥..  ü.
004040F0  00                             80 00 00 50  .........@..Ç..P
00404100  03                             FF FF 81 00  ♥.#.Z...Ω♥..  ü.
00404110  00                             00 00 01 50  ..............0P
```

Step 16

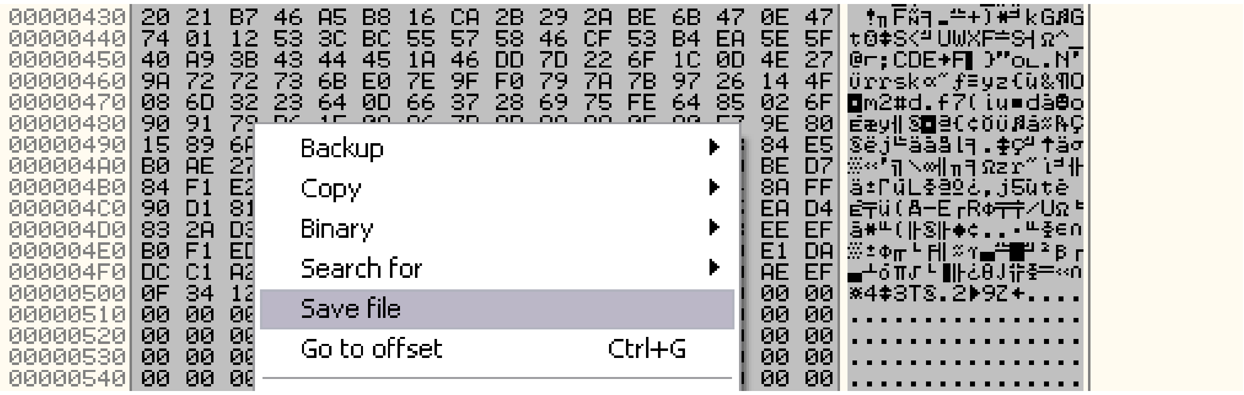
Select all the encrypted bytes from the Dump window > Right click > Copy to executable file



```
SimpleCr.<ModuleEntryPoint>
Address  Hex dump
00401000 7A 11 FA 0A 14 15
00401010 30 41 02 69 04 6F
00401020 30 31 42 DB C6 35
00401030 20 21 B7 46 A5 B8
00401040 74 01 12 53 3C BC
00401050 40 A9 3B 43 44 45
00401060 9A 72 72 73 6B E0
00401070 08 6D 32 23 64 0D
00401080 90 91 79 B6 15 08
00401090 15 89 6A C8 84 85
004010A0 B0 AE 27 BB 5C EC
004010B0 84 F1 E2 A3 4C E8
004010C0 90 D1 81 28 8F 2D
004010D0 83 2A D3 28 C7 15
004010E0 B0 F1 ED D6 C0 D5
004010F0 DC C1 A2 E3 FB C0
00401100 0F 34 12 33 54 15
00401110 00 00 00 00 00 00
00401120 00 00 00 00 00 00
00401130 00 00 00 00 00 00
00401140 00 00 00 00 00 00
00401150 00 00 00 00 00 00
```

Step 17

Right click > Save file

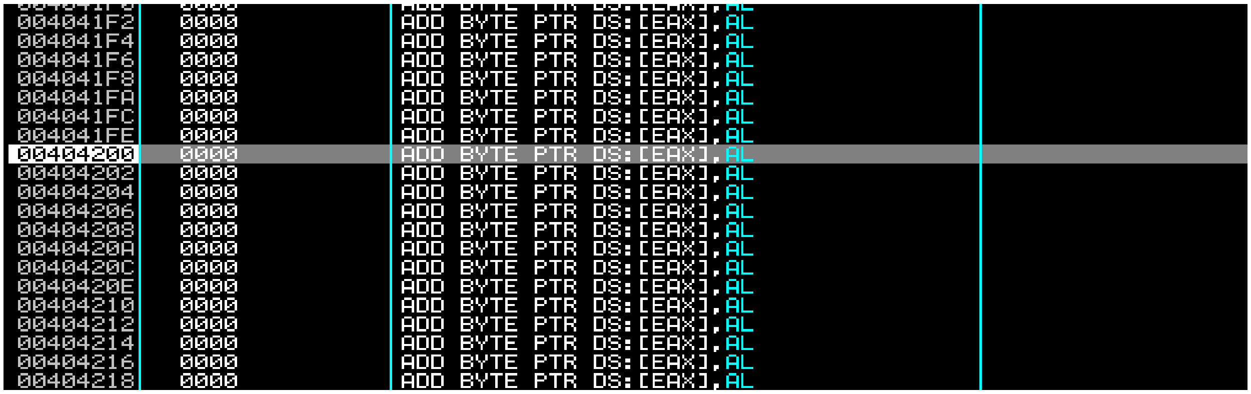


```
00000430 20 21 B7 46 A5 B8 16 CA 2B 29 2A BE 6B 47 0E 47
00000440 74 01 12 53 3C BC 55 57 58 46 CF 53 B4 EA 5E 5F
00000450 40 A9 3B 43 44 45 1A 46 DD 7D 22 6F 1C 0D 4E 27
00000460 9A 72 72 73 6B E0 7E 9F F0 79 7A 7B 97 26 14 4F
00000470 08 6D 32 23 64 0D 66 37 28 69 75 FE 64 85 02 6F
00000480 90 91 79                                  9E 80
00000490 15 89 6A                                  84 E5
000004A0 B0 AE 27                                  BE D7
000004B0 84 F1 E2                                  8A FF
000004C0 90 D1 81                                  EA D4
000004D0 83 2A D3                                  EE EF
000004E0 B0 F1 ED                                  E1 DA
000004F0 DC C1 A2                                  AE EF
00000500 0F 34 12                                  00 00
00000510 00 00 00                                  00 00
00000520 00 00 00                                  00 00
00000530 00 00 00                                  00 00
00000540 00 00 00                                  00 00
```

Save your file to a desired location. Then load that file with ollydbg (or reload if patched the current working file)

Step 18

Once again the entry point will look similar to this:



```
004041F2   0000   ADD BYTE PTR DS:[EAX],AL
004041F4   0000   ADD BYTE PTR DS:[EAX],AL
004041F6   0000   ADD BYTE PTR DS:[EAX],AL
004041F8   0000   ADD BYTE PTR DS:[EAX],AL
004041FA   0000   ADD BYTE PTR DS:[EAX],AL
004041FC   0000   ADD BYTE PTR DS:[EAX],AL
004041FE   0000   ADD BYTE PTR DS:[EAX],AL
00404200   0000   ADD BYTE PTR DS:[EAX],AL
00404202   0000   ADD BYTE PTR DS:[EAX],AL
00404204   0000   ADD BYTE PTR DS:[EAX],AL
00404206   0000   ADD BYTE PTR DS:[EAX],AL
00404208   0000   ADD BYTE PTR DS:[EAX],AL
0040420A   0000   ADD BYTE PTR DS:[EAX],AL
0040420C   0000   ADD BYTE PTR DS:[EAX],AL
0040420E   0000   ADD BYTE PTR DS:[EAX],AL
00404210   0000   ADD BYTE PTR DS:[EAX],AL
00404212   0000   ADD BYTE PTR DS:[EAX],AL
00404214   0000   ADD BYTE PTR DS:[EAX],AL
00404216   0000   ADD BYTE PTR DS:[EAX],AL
00404218   0000   ADD BYTE PTR DS:[EAX],AL
```

Next, we have to patch the decrypting code which will be responsible for decrypting the .text (code) section. A few twicks to the original encrypting code should do. All we need to do is replace these two opcodes:

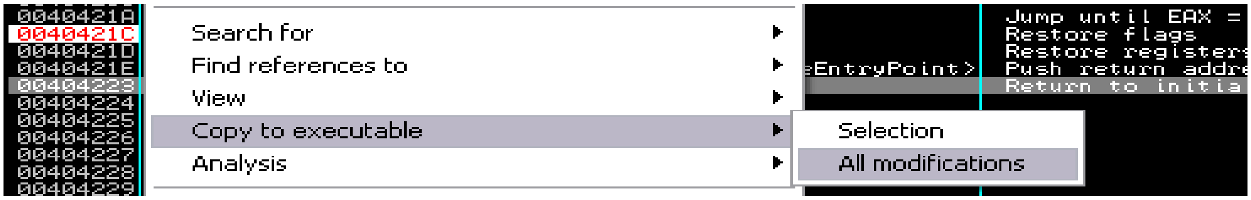
Encrypt:	Decrypt:
ADD BL,10 XOR BL,AL	XOR BL,AL SUB BL,10

Our decrypting code should look like this:

```
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        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

Apply all changes to the file, Right click > Analyze This > Right click > Copy to executable > All modifications > Copy all



```
0040421A                          Jump until EAX =
0040421C                          Restore flags
0040421D                          Restore register
0040421E        EntryPoint>       Push return addre
00404223                          Return to initia
00404224
00404225
00404226
00404227
00404228
00404229
```

Save the file to a desired location.

Step 20

Run the encrypted file

Final Words

This concludes the tutorial on how to encrypt the code section of an executable file. It is intended to be used only for educational purposes. It shows a basic approach on evading antivirus signature checking, although, antivirus solutions may as well check other sections of a PE file like the data sections therefore you will need to widen the targeted sections in order to avoid detection. Finally, if you feel that there is something missing or you would like to comment on something or even use the contents of this paper for other than personal reasons then feel free to drop us an email.