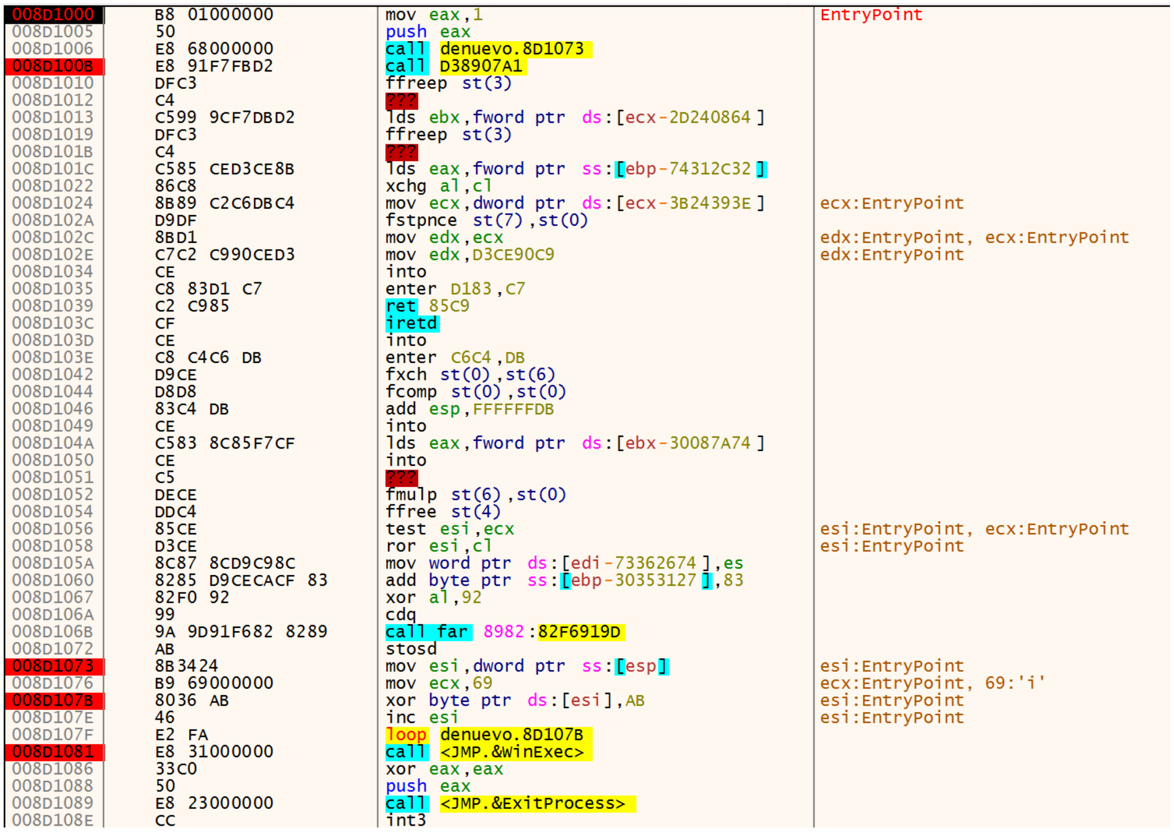
DENUEVO

1. Unpacking of the videogame

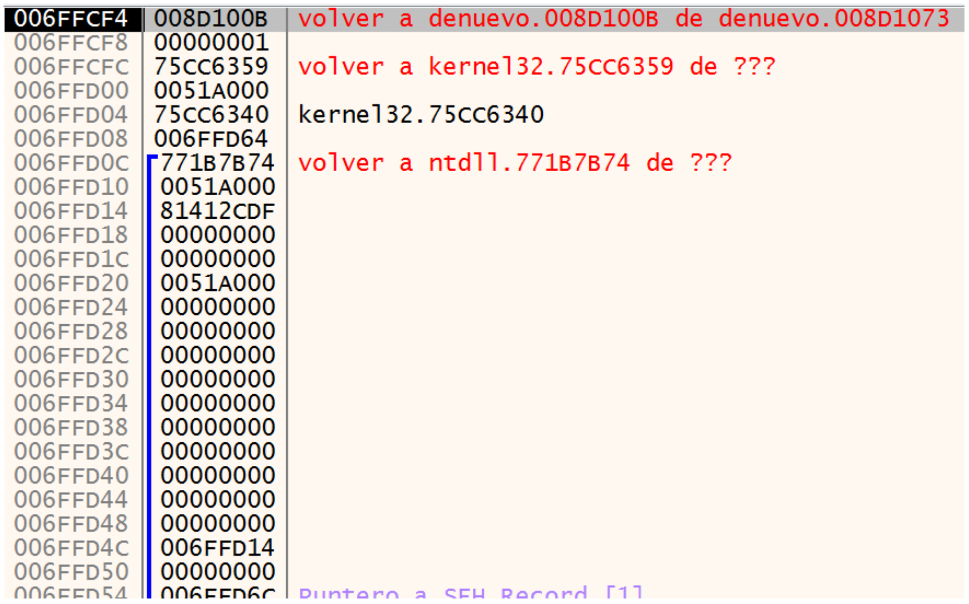
First of all, open denuevo.exe on a debugger:



```
008D1000   B8 01000000         mov eax,1                                  EntryPoint
008D1005   50                  push eax
008D1006   E8 68000000         call denuevo.8D1073
008D100B   E8 91F7FBD2         call D38907A1
008D1010   DFC3                ffreep st(3)
008D1012   C4                  ???
008D1013   C599 9CF7DBD2       lds ebx,fword ptr ds:[ecx-2D240864]
008D1019   DFC3                ffreep st(3)
008D101B   C4                  ???
008D101C   C585 CED3CE8B       lds eax,fword ptr ss:[ebp-74312C32]
008D1022   86C8                xchg al,cl
008D1024   8B89 C2C6DBC4       mov ecx,dword ptr ds:[ecx-3B24393E]        ecx:EntryPoint
008D102A   D9DF                fstpnce st(7),st(0)
008D102C   8BD1                mov edx,ecx                                edx:EntryPoint, ecx:EntryPoint
008D102E   C7C2 C990CED3       mov edx,D3CE90C9                           edx:EntryPoint
008D1034   CE                  into
008D1035   C8 83D1 C7          enter D183,C7
008D1039   C2 C985             ret 85C9
008D103C   CF                  iretd
008D103D   CE                  into
008D103E   C8 C4C6 DB          enter C6C4,DB
008D1042   D9CE                fxch st(0),st(6)
008D1044   D8D8                fcomp st(0),st(0)
008D1046   83C4 DB             add esp,FFFFFFDB
008D1049   CE                  into
008D104A   C583 8C85F7CF       lds eax,fword ptr ds:[ebx-30087A74]
008D1050   CE                  into
008D1051   C5                  ???
008D1052   DECE                fmulp st(6),st(0)
008D1054   DDC4                ffree st(4)
008D1056   85CE                test esi,ecx                               esi:EntryPoint, ecx:EntryPoint
008D1058   D3CE                ror esi,cl                                 esi:EntryPoint
008D105A   8C87 8CD9C98C       mov word ptr ds:[edi-73362674],es
008D1060   8285 D9CECACF 83    add byte ptr ss:[ebp-30353127],83
008D1067   82F0 92             xor al,92
008D106A   99                  cdq
008D106B   9A 9D91F682 8289    call far 8982:82F6919D
008D1072   AB                  stosd
008D1073   8B3424              mov esi,dword ptr ss:[esp]                 esi:EntryPoint
008D1076   B9 69000000         mov ecx,69                                 ecx:EntryPoint, 69:'i'
008D107B   8036 AB             xor byte ptr ds:[esi],AB                   esi:EntryPoint
008D107E   46                  inc esi                                    esi:EntryPoint
008D107F   E2 FA               loop denuevo.8D107B
008D1081   E8 31000000         call <JMP.&WinExec>
008D1086   33C0                xor eax,eax
008D1088   50                  push eax
008D1089   E8 23000000         call <JMP.&ExitProcess>
008D108E   CC                  int3
```

The Entry Point is located at `0x8D1000`.

First, it pushes a 1, and then at the 3rd instruction, the program will call `0x8D1073`, pushing the memory address `0x8D100B`, which it's code looks like junk since it has no sense.



```
006FFCF4  008D100B  volver a denuevo.008D100B de denuevo.008D1073
006FFCF8  00000001
006FFCFC  75CC6359  volver a kernel32.75CC6359 de ???
006FFD00  0051A000
006FFD04  75CC6340  kernel32.75CC6340
006FFD08  006FFD64
006FFD0C  771B7B74  volver a ntdll.771B7B74 de ???
006FFD10  0051A000
006FFD14  81412CDF
006FFD18  00000000
006FFD1C  00000000
006FFD20  0051A000
006FFD24  00000000
006FFD28  00000000
006FFD2C  00000000
006FFD30  00000000
006FFD34  00000000
006FFD38  00000000
006FFD3C  00000000
006FFD40  00000000
006FFD44  00000000
006FFD48  00000000
006FFD4C  006FFD14
006FFD50  00000000
006FFD54  006FFD6C  Puntero a SEH Record [1]
```

Then moves some values to the registers, executes a loop between `0x8D107B` and `0x8D107F`, and at the last step calls WinExec at `0x8D1081`, which executes a command.

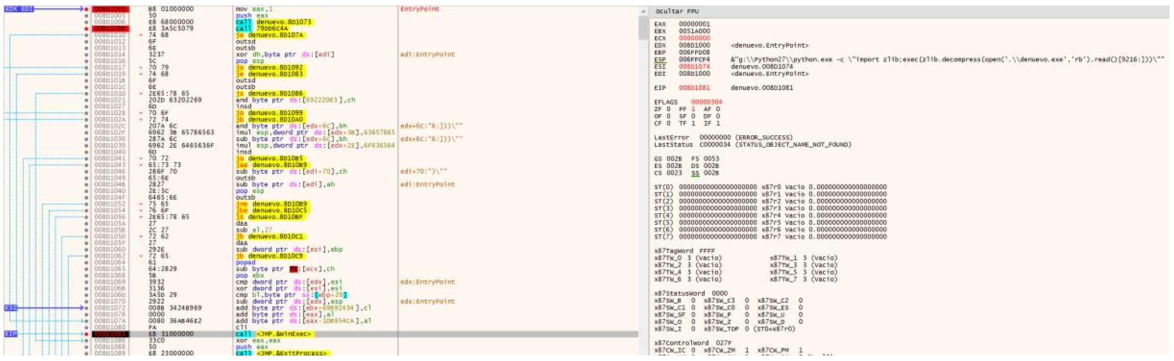
The command executed is saved at the last memory address pushed onto the stack, which is the return address of the previous call. It looks like its using as a command a array of non-printable characters.

After looking closer to the code situated between 0x8D1073 and `0x8d107F`, we can see that:

- First instruction: The memory address of the command (`0x8D100B`), which is stored at [ESP], is saved in ESI.
- Second instruction: The length of the command is saved in ECX (0x69)

Finally, we reach the loop, which basically xors every character of the command with the byte 0xAB.

After understanding this, if we stop the program with a breakpoint before WinExec is called, we will see the decrypted command:

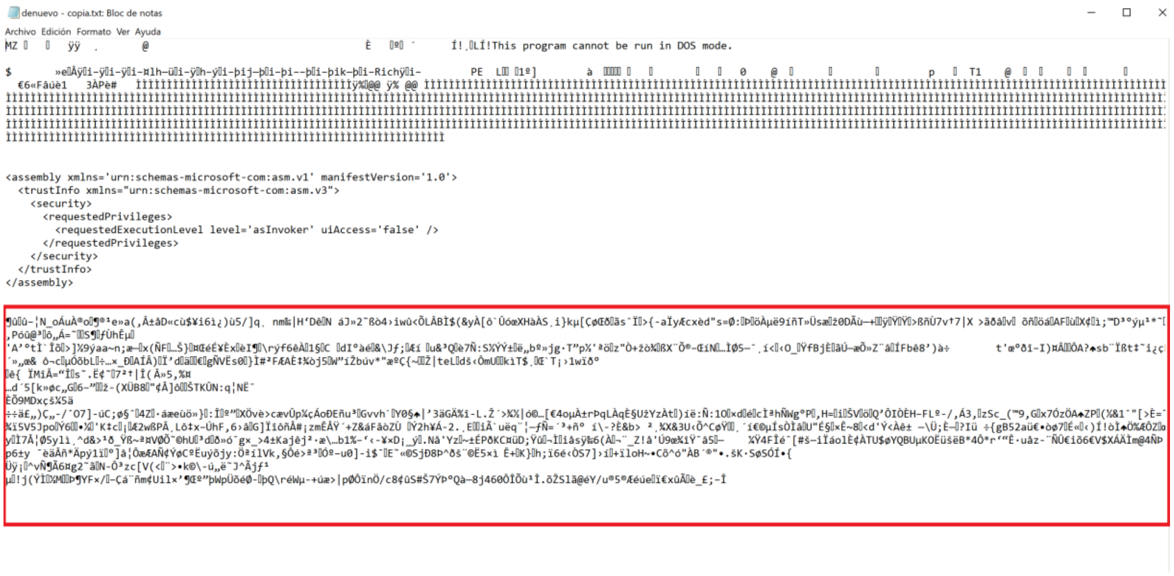


```
Ocultar FPU
EAX    00000001
EBX    0051A000
ECX    00000000
EDX    008D1000     <denuevo.EntryPoint>
EBP    006FFD08
ESP    006FFCF4     &"g:\\Python27\\python.exe -c \"import zlib;exec(zlib.decompress(open('.\\denuevo.exe','rb').read()[9216:]))\""
ESI    008D1074     denuevo.008D1074
EDI    008D1000     <denuevo.EntryPoint>

EIP    008D1081     denuevo.008D1081

EFLAGS    00000304
ZF 0  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 1  IF 1

LastError   00000000 (ERROR_SUCCESS)
LastStatus  C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) 0000000000000000000 x87r0 Vacio 0.000000000000000000
ST(1) 0000000000000000000 x87r1 Vacio 0.000000000000000000
ST(2) 0000000000000000000 x87r2 Vacio 0.000000000000000000
ST(3) 0000000000000000000 x87r3 Vacio 0.000000000000000000
ST(4) 0000000000000000000 x87r4 Vacio 0.000000000000000000
ST(5) 0000000000000000000 x87r5 Vacio 0.000000000000000000
ST(6) 0000000000000000000 x87r6 Vacio 0.000000000000000000
ST(7) 0000000000000000000 x87r7 Vacio 0.000000000000000000

x87TagWord  FFFF
x87TW_0  3 (Vacio)          x87TW_1  3 (Vacio)
x87TW_2  3 (Vacio)          x87TW_3  3 (Vacio)
x87TW_4  3 (Vacio)          x87TW_5  3 (Vacio)
x87TW_6  3 (Vacio)          x87TW_7  3 (Vacio)

x87StatusWord  0000
x87SW_B   0  x87SW_C3   0  x87SW_C2  0
x87SW_C1  0  x87SW_C0   0  x87SW_ES  0
x87SW_SF  0  x87SW_P    0  x87SW_U   0
x87SW_O   0  x87SW_Z    0  x87SW_D   0
x87SW_I   0  x87SW_TOP  0 (ST0=x87r0)

x87ControlWord  027F
x87CW_IC  0  x87CW_ZM  1  x87CW_PM  1
```

The last address stored in [ESP] points to the decrypted command:
G:\Python27\python.exe -c "import zlib; exec(zlib.decompress(open('.\denuevo.exe', 'rb').read()[9216:]))"

It reads the executable itself, and decompreses bytes located after the 9216 byte. Then, that decompressed result gets executed.

The zipped python code is located after the end of the code:



denuevo - copia.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
MZ        ÿÿ  ¸       @                         È   º  ´  Í!¸LÍ!This program cannot be run in DOS mode.

$       »eÅÿi-ÿi-ÿi-¤1h-üi-ÿh-ýi-þij-þi-þi--þi-þik-þi-Richÿi-        PE  L  1º]        à

<assembly xmlns='urn:schemas-microsoft-com:asm.v1' manifestVersion='1.0'>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel level='asInvoker' uiAccess='false' />
      </requestedPrivileges>
    </security>
  </trustInfo>
</assembly>
```

Its located after the manifest of the exe, and it wont corrupt the executable as long as it is located after the end of the .exe

Now, lets unpack the python code since we know how the protection works. It can be done with the command:

```
C:\Python27\python.exe -c "import zlib; print zlib.decompress(open('denuevo.exe', 'rb').read()[9216:])" > unpacked.py
```

2. Getting the License key

The license check code is at the beginning and it's very simple:

```
x = list(raw_input("Enter your serial key: "))
for i in ['j','7','b','g','5','6','2']:
        if x.pop() != i: exit()
i = int(i)
while i != 9:
        if x.pop() != ['j','x','d','y','z','3','5'][i]: exit()
        i += -4 if i + 3 > 6 and i != 6 else 3
print "Your serial key has activated the videogame! Submit the challenge with TMHC{serial}"
```

The first for-loop checks that the license ends with "265gb7j"

The second while-loop its a little bit more complicated but its the same, it checks that the license key ends with "5yjzx3d" before the "265gb7j".

So the serial ends up being: "5yjzx3d265gb7j".