

Unpacking Vmprotect packer

 muha2xmad.github.io/unpacking/Vmprotect/

January 9, 2022



Muhammad Hasan Ali

Malware Analysis learner

2 minute read

As-salamu Alaykum

Introduction

Unpacking a file which is packed using commercial `Vmprotect` packer.

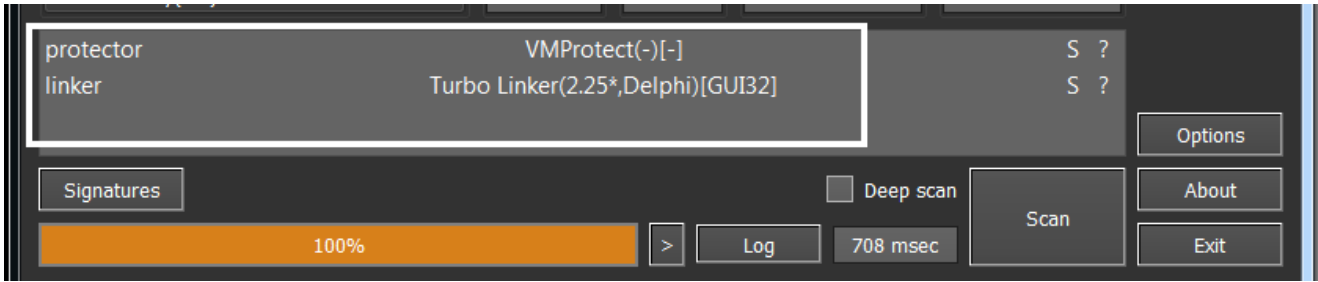
Download the sample: [Here](#)

MD5: A39B4F74B5108A2B9F1A33B2FEB22CC5

Static

DIE

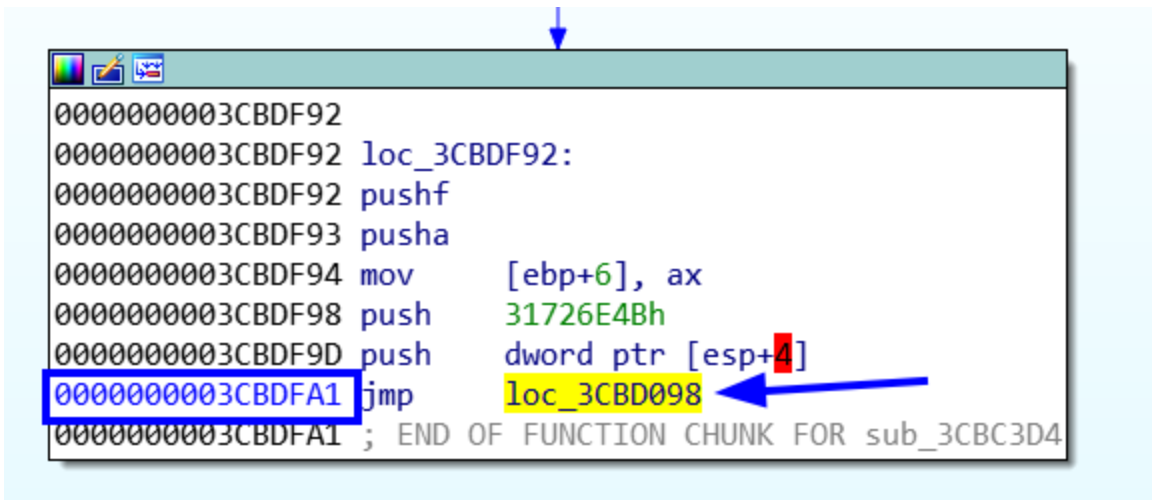
This sample is written in `Delphi` language and is packed through a commercial packer called `VmProtect`. And its `Entropy` is very high in section `Vmp1`



Figure(1):

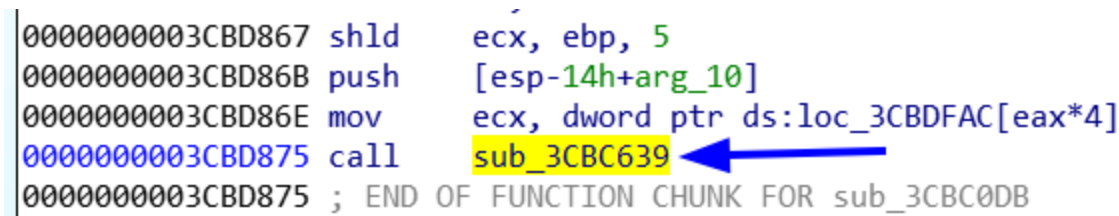
IDA

Open it in IDA. We will notice that there is so many jumps and calls which the packer uses to obfuscate the code. So if you **notice** that if you press on any jump in the function you still in the same function. But the last jump or call will go to another function. So to short efforts and time, we will go to the last call or jump of the function and keep doing that till we get to this one.



Figure(2):

We will reach to a function which has a loop if we press on the last jump of that function it will bring us to the same function. And after checking all the calls and jumps. this call is our way.



Figure(3):

Till we find an interesting function. because its `retn 48` it will return the last instruction which is `push [esp+14h+arg_2C]` and we need its address `03CBF12E` because it will help us in debugging.

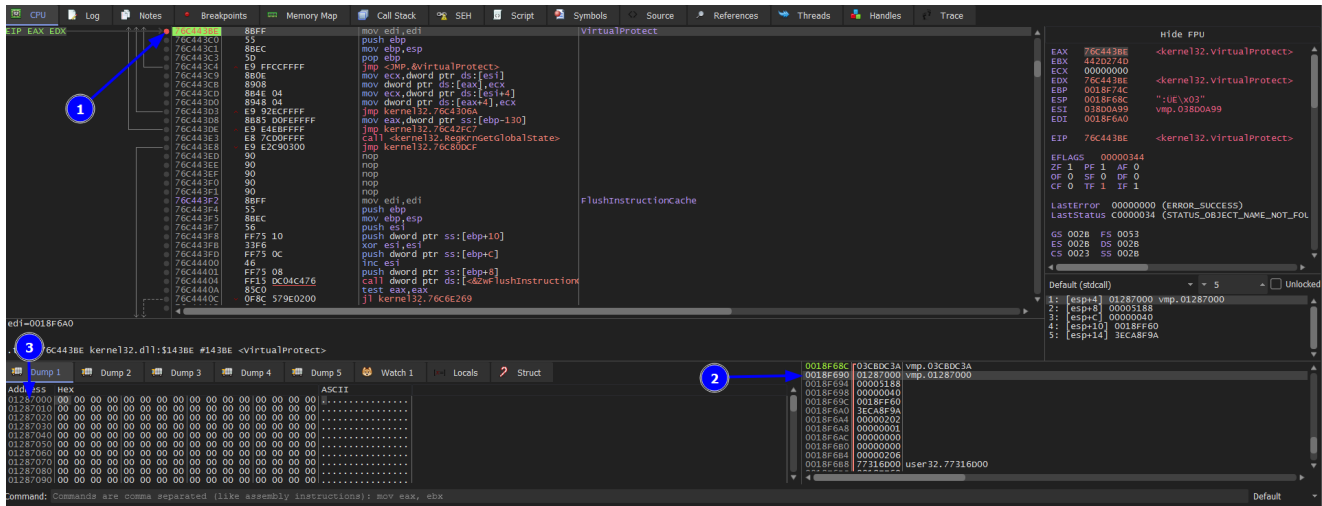
```
0000000003CBF0FE
0000000003CBF0FE
0000000003CBF0FE sub_3CBF0FE proc near
0000000003CBF0FE
0000000003CBF0FE var_10= dword ptr -10h
0000000003CBF0FE var_8= word ptr -8
0000000003CBF0FE arg_2C= dword ptr 30h
0000000003CBF0FE
0000000003CBF0FE xor     esi, 24DFCCF6h
0000000003CBF104 rol     ebp, cl
0000000003CBF106 xor     bp, 9579h
0000000003CBF10B sar     bp, 0Bh
0000000003CBF10F sub     esi, offset dword_35D2024
0000000003CBF115 call    sub_3CBDC8A
```

```
0000000003CBF11A
0000000003CBF11A loc_3CBF11A:
0000000003CBF11A push   esp
0000000003CBF11B push   edx
0000000003CBF11C mov    [esp+8+arg_2C], ecx
0000000003CBF120 push   0CA35606Ch
0000000003CBF125 mov    [esp+0Ch+var_8], ax
0000000003CBF12A pushf
0000000003CBF12B push   [esp+10h+var_10]
0000000003CBF12E push   [esp+14h+arg_2C]
0000000003CBF132 retn   48h
0000000003CBF132 sub_3CBF0FE endp ; sp-4
0000000003CBF132
```

Figure(4):

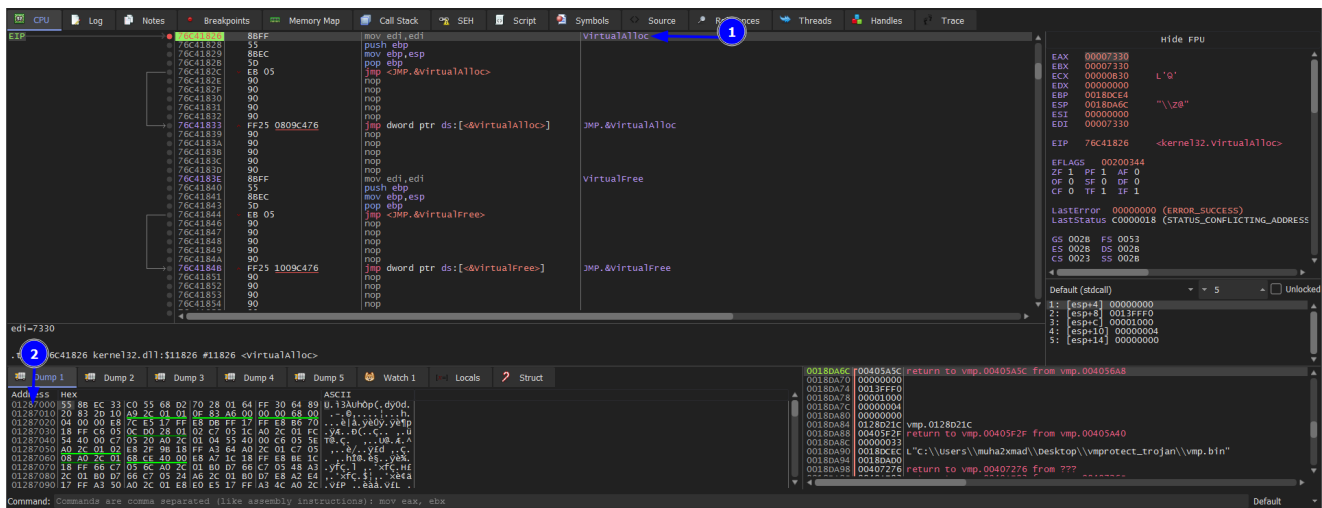
Unpacking process

We set breakpoints on `03CBF12E` , `VirtualAlloc` ,and `VirtualProtect` . Then we hit `run` to see us in `03CBF12E` and we `Follow in disassembler` of its `value` . We did that we might see a call to a register in this section. But we won't find any then we unset this breakpoint `03CBF12E` . And see `Strings` references you will see a few strings. Then press `run` to hit `VirtualProtect` breakpoint and keep pressing `run` . **Till** we see this address `01287000` then we dump it.



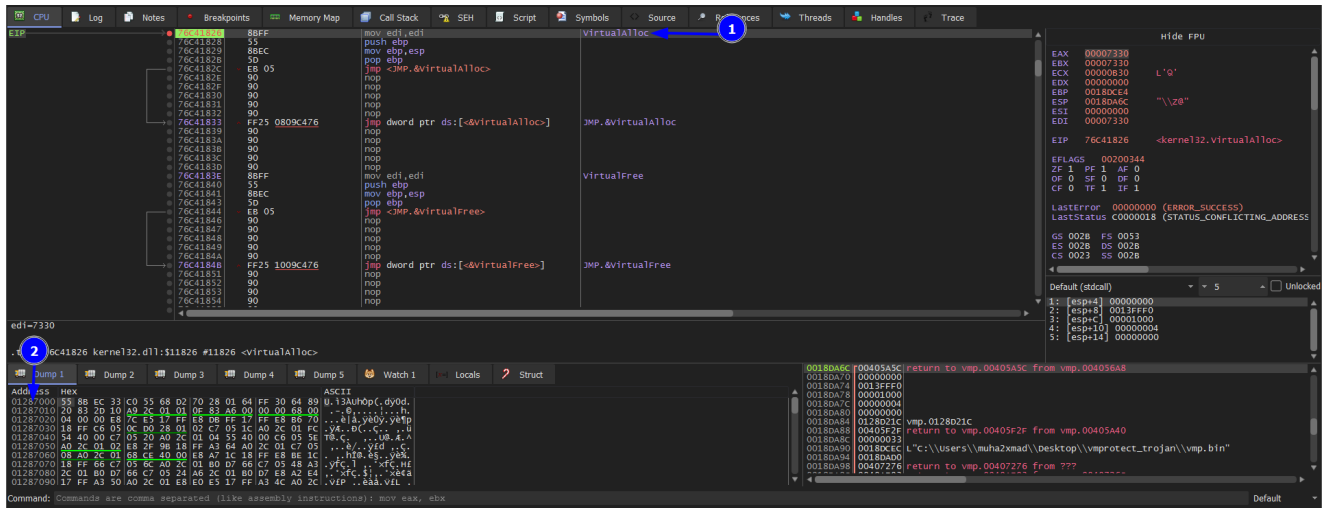
Figure(5):

Why that address? Because we need to find the OEP which is `push ebp` which we will find it after this address `01287000` which will be later `0128C074`. After that we keep pressing `run` to hit `VirtualAlloc` breakpoint.



Figure(6):

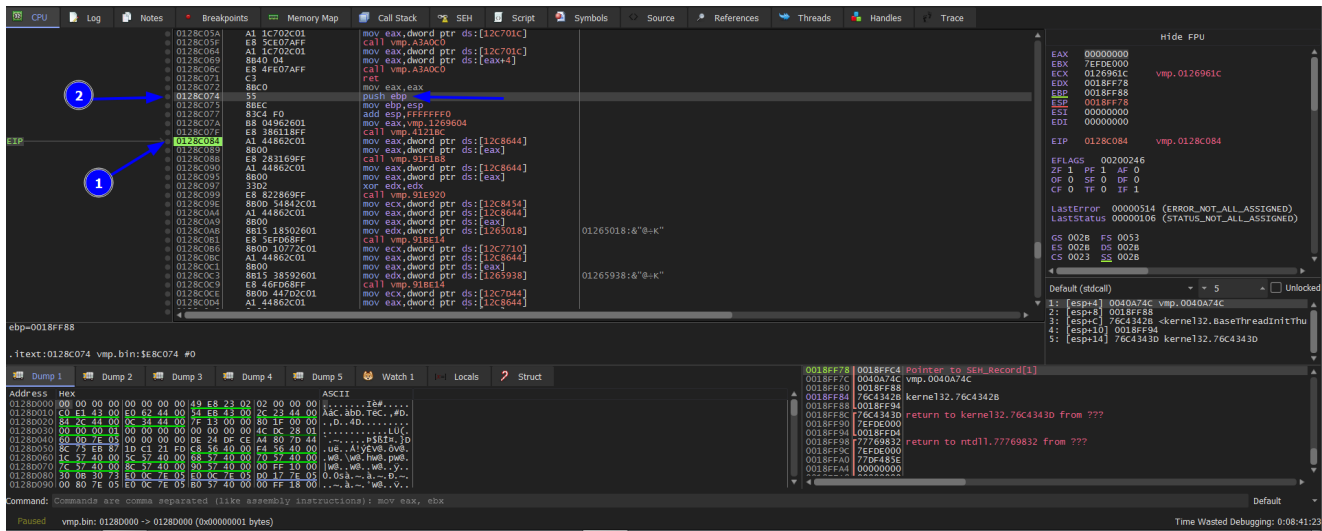
The last part is the what you need to focus on. As we said we will find the OEP above this address `01287000` which we will be searching for this instruction `push ebp`. Then press `execute till return` after that press `run to user code` while doing that keep you eyes on the `Memory address` because the OEP is in the range of `01287000`. After trying the previous and hitting `VirtualAlloc` 4 times, we found the `push ebp` our OEP.



Figure(6):

Unmapping

A friend told me to short effort and time do it automatically using `scylla`. After that the unpacked file is big **56.7MB**



Figure(7):

Article quote

ولو لم يحاصر ك جيش الظلام لما كنت نجماً يُرى أو قمر

REF

1- https://www.youtube.com/watch?v=aoa89Khfgr0&ab_channel=GuidedHacking