Malware analysis with IDA/Radare2 - Basic Unpacking (Dridex first stage)

artik.blue/malware3

Greetings again dear malware analysts! In this part of the series we are leaving the initial topics behind to start focusing on unpacking/decrypting malware by applying our reverse engineering skills to those binaries. We will start by unpacking the initial stage of Dridex, a malware focused on stealing bank credentials.

The sample to be used in this post can be found here

Introduction

As explained here Obfuscation takes code and basically makes it unreadable (to the analyst, that is: us) without destroying its intended functionality. This technique is used to delay detection and/or to make reverse engineering difficult. Obfuscation does have legitimate purpose as it can be used to protect intellectual property or other sensitive code. Packing is a subset of obfuscation: A packer is a tool that modifies the formatting of code by compressing or encrypting the data. Though often used to delay the detection of malicious code, there is still legitimate use for packing. Some legitimate use includes protecting intellectual property or other sensitive data from being copied.

Packed binaries will consist basically of two elements: The encrypted payload, that is, the “final code” to be executed that will be compressed/encrypted to avoid detection/analysis and a “stub” that is, a piece of code in charge of decrypting/decompressing that payload to run it in some way. That stub can also be written in a way to harden the analysis sometimes using various anti-analysis techniques or using an overly complex “hard to read” code. At the same time a binary can be packed several times like a Matryoshka doll. Overall Antianalysis techniques may include calling functions such as IsDebuggerPresent and closing the program if it returns 1, having the relevant functions encoded/obfuscated in the code to harden the analysis, checking if certain common analysis programs are present on the machine, checking for the ram/processor features to detect potential VM runs… and many more.

The packed program we are going to reverse today won’t contain many complex antianalysis mechanisms, it will be relatively easy to un pack by debugging it, so we can get started from here and move on to something more complex.

Detecting packed binaries

As we start reversing a malware one of the first things we should check is the packing. Usually we will quickly see that a program is packed as it will contain a bunch of nonsense strings along with very few internal functions and imports. Parts on the code may also include several loops with opcodes related to xor and other potential decode/decrypt functions. Though it is an aprox, it doesn’t have to be exactly like this.

A high entropy on the binary may also be a clear indicator that it is packed, as obviously an encrypted payload (that is, generated by an encryption/compression) algorithm will increase the entropy of the file.

We can check for that using rahash:

```
PS C:\Users\lab\Desktop > rahash2.exe -a entropy .\drdrex.exe
.\drdrex.exe: 0x00000000-0x00033fff entropy: 7.53635438
PS C:\Users\lab\Desktop >
```

As we see in our case entropy is >7, from my experience usually entropies higher than 6 and something correlate with packed binaries.

We can also check for the strings using rafind/rabin, to check for those nonesense-strings.

```
PS C:\Users\lab\Desktop > rafind2.exe -Z .\drdrex.exe | more
0x0000004d !This program cannot be run in DOS mode.\r\r\n$
0x00000190 RichWE
0x000002e4 .text
0x0000030b `.rdata
0x00000333 @.data
0x0000035c .text1
0x00000384 .rsrc
0x000003ab @.reloc
0x100010b0 D1\vM
0x100010f9 \b1Êê═Ç
0x10001193 E\b1╔ëL$
0x100011ad \fr\n)
0x100011bd L$\fr
0x10001214  ^_]
0x10001301 2f;u
0x10001327 ]\vM─ïU
0x10001356 \nEKMÈëM╚¿
0x1000137e f)╚ëU
0x100013d5 EÏï@(
0x100013db E╟ïEÏï@
0x100013e4 EÈïEÈèU
0x10001408 L_^[]├ïEÏï
0x100014b7 MÏë\f$
0x100014bd U▄ëT$
0x10001600 D$t9
0x1000161a ^[_]├ïD$(
0x10001641 JeãêÌëL$
0x10001659 EÌë\\$l
0x10001660 A█æ?)
0x100016a8 t$8+
0x100016c6 t$|8
0x100016ce ┼ïD$
0x100016d8 ╚ëD$4
0x1000170b D$$9D$xt:
[...]
0x10004706 ExitProcess
0x10004714 LoadLibraryExW
0x10004726 GetModuleHandleW
0x1000473a GetLastError
0x1000474a GetModuleFileNameA
0x1000475e KERNEL32.dll
0x1000476e SHDeleteKeyA
0x1000477e SHGetValueA
0x1000478a SHLWAPI.dll
0x10004798 InternetCombineUrlA
0x100047ac WININET.dll
0x100047ba SetupDuplicateDiskSpaceListW
0x100047d8 SETUPAPI.dll
0x100047e8 GetQueueStatus
0x100047fa GetScrollPos
0x1000480a FindWindowExW
```

```
0x10004818 USER32.dll
0x10004826 isalpha
0x1000482e msvcrt.dll
0x100072ee IF3▄£
0x100072fb ?6&/
0x10007344 `^KOB
0x10007361 _L3O
0x10007375 zm/0┬À
0x10007392 Tr@┬Ñ
[...]
```

So we start seeing interesting stuff in here. We see those nonesense strings along with other ones refering to potential api calls and libraries… an indicator that the binary may be resolving/loading stuff dynamically at some point after it starts… but let’s move on:

We can also use rabin to check for the basic info on the binary, always useful:

```
PS C:\Users\lab\Desktop > rabin2.exe -I -l -s .\drdrex.exe
[Symbols]

nth paddr      vaddr      bind   type size lib          name
------------------------------------------------------------
1   0x000044b6 0x100044b6 GLOBAL FUNC 0    vplD.dll     BpfrBpdm16
1   0x00004028 0x10004028 NONE   FUNC 0    RPCRT4.dll   imp.I_RpcNsInterfaceExported
1   0x00004000 0x10004000 NONE   FUNC 0    ADVAPI32.dll imp.PrivilegeCheck
53  0x00004020 0x10004020 NONE   FUNC 0    OLEAUT32.dll imp.VarI2FromStr
1   0x00004064 0x10004064 NONE   FUNC 0    pdh.dll      imp.PdhExpandWildCardPathW
1   0x00004008 0x10004008 NONE   FUNC 0    KERNEL32.dll imp.GetModuleHandleW
2   0x0000400c 0x1000400c NONE   FUNC 0    KERNEL32.dll imp.LoadLibraryExW
3   0x00004010 0x10004010 NONE   FUNC 0    KERNEL32.dll imp.ExitProcess
4   0x00004014 0x10004014 NONE   FUNC 0    KERNEL32.dll imp.GetLastError
5   0x00004018 0x10004018 NONE   FUNC 0    KERNEL32.dll imp.GetModuleFileNameA
1   0x00004038 0x10004038 NONE   FUNC 0    SHLWAPI.dll  imp.SHGetValueA
2   0x0000403c 0x1000403c NONE   FUNC 0    SHLWAPI.dll  imp.SHDeleteKeyA
1   0x00004054 0x10004054 NONE   FUNC 0    WININET.dll  imp.InternetCombineUrlA
1   0x00004030 0x10004030 NONE   FUNC 0    SETUPAPI.dll
imp.SetupDuplicateDiskSpaceListW
1   0x00004044 0x10004044 NONE   FUNC 0    USER32.dll   imp.GetQueueStatus
2   0x00004048 0x10004048 NONE   FUNC 0    USER32.dll   imp.GetScrollPos
3   0x0000404c 0x1000404c NONE   FUNC 0    USER32.dll   imp.FindWindowExW
1   0x0000405c 0x1000405c NONE   FUNC 0    msvcrt.dll   imp.isalpha
arch     x86
baddr    0x10000000
binsz    212992
bintype  pe
bits     32
canary   true
retguard false
class    PE32
cmp.csum 0x0003f0e0
compiled Fri Apr 24 04:04:11 2020
crypto   false
endian   little
havecode true
hdr.csum 0x00000000
laddr    0x0
lang     c
linenum  false
lsyms    false
machine  i386
nx       true
os       windows
overlay  false
cc       cdecl
pic      true
relocs   false
signed   false
sanitize false
static   false
```

```
stripped false
subsys   Windows GUI
va       true
[Linked libraries]
rpcrt4.dll
advapi32.dll
oleaut32.dll
pdh.dll
kernel32.dll
shlwapi.dll
wininet.dll
setupapi.dll
user32.dll
msvcrt.dll

10 libraries
PS C:\Users\lab\Desktop >
```

So in here we see that it is an x32 binary and also that it uses interesting functions such as GetModuleHandle and LoadLibrary, related to what we previously guessed it may be doing.

At this point we have two options on one hand we can go statically analyse the program or just try and debug the program to check for the unpacking and extract the second stage. For simplicity a common choice will be to just extract the payload after it gets unpacked, as we want to reverse the malware, not the packer in here.

Debugging in Windows with radare2

So we open the binary with radare2 and then we spawn a process and go on until we hit the entry point:

```
[0x100062b0]> ood
(6180) Finished thread 4108 Exit code 1

==> Process finished

INFO: Spawned new process with pid 5556, tid = 7052
File dbg://C:\\Users\\lab\\Desktop\\radare32\\bin\\todebug\\drdrex.exe reopened in
read-write mode
Unable to find file descriptor 6
Unable to find file descriptor 6
[0x77224f90]> dcu entry0
Continue until 0x100033f3 using 1 bpsize
(5556) loading library at 0x771B0000 (C:\Windows\SysWOW64\ntdll.dll) ntdll.dll
(5556) loading library at 0x75CF0000 (C:\Windows\SysWOW64\kernel32.dll) kernel32.dll
(5556) loading library at 0x759D0000 (C:\Windows\SysWOW64\KernelBase.dll)
KernelBase.dll
(5556) loading library at 0x76C40000 (C:\Windows\SysWOW64\rpcrt4.dll) rpcrt4.dll
(5556) loading library at 0x750D0000 (C:\Windows\SysWOW64\advapi32.dll) advapi32.dll
(5556) loading library at 0x768D0000 (C:\Windows\SysWOW64\msvcrt.dll) msvcrt.dll
(5556) loading library at 0x75950000 (C:\Windows\SysWOW64\sechost.dll) sechost.dll
(5556) loading library at 0x751B0000 (C:\Windows\SysWOW64\oleaut32.dll) oleaut32.dll
(5556) loading library at 0x75DE0000 (C:\Windows\SysWOW64\msvcp_win.dll)
msvcp_win.dll
(5556) loading library at 0x76A70000 (C:\Windows\SysWOW64\ucrtbase.dll) ucrtbase.dll
(5556) loading library at 0x76F10000 (C:\Windows\SysWOW64\combase.dll) combase.dll
(5556) loading library at 0x76420000 (C:\Windows\SysWOW64\shlwapi.dll) shlwapi.dll
(5556) loading library at 0x75510000 (C:\Windows\SysWOW64\setupapi.dll) setupapi.dll
(5556) loading library at 0x75070000 (C:\Windows\SysWOW64\cfgmgr32.dll) cfgmgr32.dll
(5556) loading library at 0x76B90000 (C:\Windows\SysWOW64\bcrypt.dll) bcrypt.dll
(5556) loading library at 0x766E0000 (C:\Windows\SysWOW64\user32.dll) user32.dll
(5556) loading library at 0x750B0000 (C:\Windows\SysWOW64\win32u.dll) win32u.dll
(5556) loading library at 0x76C10000 (C:\Windows\SysWOW64\gdi32.dll) gdi32.dll
(5556) loading library at 0x76D00000 (C:\Windows\SysWOW64\gdi32full.dll)
gdi32full.dll
(5556) loading library at 0x74410000 (C:\Windows\SysWOW64\pdh.dll) pdh.dll
(5556) loading library at 0x73D00000 (C:\Windows\SysWOW64\wininet.dll) wininet.dll
[0x77261ba3]> dcu entry0
Continue until 0x100033f3 using 1 bpsize
(5556) loading library at 0x752C0000 (C:\Windows\SysWOW64\imm32.dll) imm32.dll
(5556) Created thread 1872 (start @ 771E5900) (teb @ 00223000)
hit breakpoint at: 0x100033f3
```

And we see a bunch of functions present in the program:

```
[0x100033f3]> afl
0x100033f3    3 30           entry0
0x10007325    1 13           fcn.10007325
0x10014d56    1 4            int.10014d56
0x10020f60   11 223  -> 135  fcn.10020f60
0x10022376    1 23           fcn.10022376
0x100012a9   18 476          fcn.100012a9
0x10001c33   13 385          fcn.10001c33
0x10001000    7 163          fcn.10001000
0x10001561   35 1120         fcn.10001561
0x1000127e    1 29           fcn.1000127e
0x10002e22    4 76           fcn.10002e22
0x10003053    1 83           fcn.10003053
0x10001aea    7 258          fcn.10001aea
0x100019dd    7 229          fcn.100019dd
0x10001123    6 100          fcn.10001123
0x100030be    6 333          fcn.100030be
0x10002778    4 81           fcn.10002778
0x1000320b   10 467          fcn.1000320b
0x100025dc    6 240          fcn.100025dc
0x10002f87    6 204          fcn.10002f87
0x10002879   18 257          fcn.10002879
0x10002bb8   19 618          fcn.10002bb8
0x100010a3   10 128          fcn.100010a3
0x10001187    3 89           fcn.10001187
0x100011e0    6 158          fcn.100011e0
0x1000129c    1 13           fcn.1000129c
0x10001485    9 220          fcn.10001485
0x100019c1    1 28           fcn.100019c1
0x10001ac2    1 36           fcn.10001ac2
0x10001bec    6 71           fcn.10001bec
0x10001db4    1 67           fcn.10001db4
0x10002304    2 69           fcn.10002304
0x10002349    2 71           fcn.10002349
0x10002390    3 106          fcn.10002390
0x100023fa    8 230          fcn.100023fa
0x100024e0   13 172          fcn.100024e0
0x1000258c    1 80           fcn.1000258c
0x100026cc    1 23           fcn.100026cc
0x100026e3    6 149          fcn.100026e3
0x100027c9    3 61           fcn.100027c9
0x10002806    1 29           fcn.10002806
0x10002823    6 86           fcn.10002823
0x1000297a    6 119          fcn.1000297a
0x100029f1    9 182          fcn.100029f1
0x10002e6e    3 54           fcn.10002e6e
0x10002ea4    6 227          fcn.10002ea4
0x100030a6    1 24           fcn.100030a6
0x100033de    1 21           fcn.100033de
0x10003414    6 162          fcn.10003414
0x100034b6    1 5            fcn.100034b6
[0x100033f3]>
```

Also we see several operations in the main, mostly related to moving data around and then calling stuff:

```
[0x100033f3]> pd 50
            ;-- edi:
            ;-- esi:
            ;-- edx:
            ;-- ecx:
            ;-- eip:
/ 30: entry0 ();
|           0x100033f3      8d3514340010   lea esi, [fcn.10003414]     ; 0x10003414 ;
"U\x89\xe5V\x83\xec(\xc7E\xf8~\xa8\xe5G\xc7E\xf4\x7f.K\x11\xe8K\xf4\xff\xff\xb9~\xa8\x

|           0x100033f9      892510500010   mov dword [0x10005010], esp ;
[0x10005010:4]=0
|           0x100033ff      891d08500010   mov dword [0x10005008], ebx ;
[0x10005008:4]=0
|           0x10003405      56             push esi
|       ,=< 0x10003406      eb01           jmp 0x10003409
|       |   ; CODE XREF from entry0 @ 0x1000340f(x)
|      .--> 0x10003408      c3             ret
|      :|   ; CODE XREF from entry0 @ 0x10003406(x)
|      :`-> 0x10003409      892d0c500010   mov dword [0x1000500c], ebp ;
[0x1000500c:4]=0
\      `==< 0x1000340f      ebf7           jmp 0x10003408
            0x10003411      8945fc         mov dword [ebp - 4], eax
            ; DATA XREF from entry0 @ 0x100033f3(r)
/ 162: fcn.10003414 ();
|           ; var int32_t var_8h @ ebp-0x8
|           ; var int32_t var_ch @ ebp-0xc
|           ; var int32_t var_10h @ ebp-0x10
|           ; var int32_t var_14h @ ebp-0x14
|           ; var int32_t var_18h @ ebp-0x18
|           ; var int32_t var_1ch @ ebp-0x1c
|           ; var int32_t var_20h @ ebp-0x20
|           0x10003414      55             push ebp
|           0x10003415      89e5           mov ebp, esp
|           0x10003417      56             push esi
|           0x10003418      83ec28         sub esp, 0x28
|           0x1000341b      c745f87ea8e5.  mov dword [var_8h], 0x47e5a87e
|           0x10003422      c745f47f2e4b.  mov dword [var_ch], 0x114b2e7f ;
'\x7f.K\x11'
|           0x10003429      e84bf4ffff     call fcn.10002879
|           0x1000342e      b97ea8e547     mov ecx, 0x47e5a87e
|           0x10003433      2b4df8         sub ecx, dword [var_8h]
|           0x10003436      39c8           cmp eax, ecx
|       ,=< 0x10003438      7567           jne 0x100034a1
|       |   0x1000343a      b86ba75005     mov eax, 0x550a76b
|       |   0x1000343f      3b45f4         cmp eax, dword [var_ch]
|      ,==< 0x10003442      733c           jae 0x10003480
|      ||   0x10003444      8d0542440010   lea eax, str.kern32.ll      ; 0x10004442 ;
u"kern32.ll"
|      ||   0x1000344a      31c9           xor ecx, ecx
|      ||   0x1000344c      ba01000000     mov edx, 1
|      ||   0x10003451      89e6           mov esi, esp
|      ||   0x10003453      c74608010000.  mov dword [esi + 8], 1
|      ||   0x1000345a      c74604000000.  mov dword [esi + 4], 0
|      ||   0x10003461      c70642440010   mov dword [esi], str.kern32.ll ;
[0x10004442:4]=0x65006b ; u"kern32.ll"
|      ||   0x10003467      8b350c400010   mov esi, dword
[sym.imp.KERNEL32.dll_LoadLibraryExW] ; [0x1000400c:4]=0x75d0f3a0
|      ||   0x1000346d      8945f0         mov dword [var_10h], eax
|      ||   0x10003470      894dec         mov dword [var_14h], ecx
|      ||   0x10003473      8955e8         mov dword [var_18h], edx
|      ||   0x10003476      ffd6           call esi
|      ||   0x10003478      83ec0c         sub esp, 0xc
|      ||   0x1000347b      83f800         cmp eax, 0
|     ,===< 0x1000347e      7521           jne 0x100034a1
|     |||   ; CODE XREF from fcn.10003414 @ 0x10003442(x)
|     |`--> 0x10003480      8d0556440010   lea eax, str.self.exe       ; 0x10004456 ;
u"self.exe"
|     | |   0x10003486      89e1           mov ecx, esp
|     | |   0x10003488      c70156440010   mov dword [ecx], str.self.exe ;
[0x10004456:4]=0x650073 ; u"self.exe"
|     | |   0x1000348e      8b0d08400010   mov ecx, dword
[sym.imp.KERNEL32.dll_GetModuleHandleW] ; [0x10004008:4]=0x75d10e50 ;
"P\x0e\xd1u\xa0\xf3\xd0u\x10N\xd1u\x10\xe0\xd0u0\x0e\xd1u"
|     | |   0x10003494      8945e4         mov dword [var_1ch], eax
|     | |   0x10003497      ffd1           call ecx
|     | |   0x10003499      83ec04         sub esp, 4
|     | |   0x1000349c      83f800         cmp eax, 0
|     |,==< 0x1000349f      7406           je 0x100034a7
|     |||   ; CODE XREFS from fcn.10003414 @ 0x10003438(x), 0x1000347e(x),
0x100034b4(x)
|     `-`-> 0x100034a1      83c428         add esp, 0x28
|      |    0x100034a4      5e             pop esi
|      |    0x100034a5      5d             pop ebp
[0x100033f3]>
```

As far as I can guess firstly the program will go load needed libraries, then maybe some system checks, then the unpacking…

Let’s now check for the modules:

```
[0x100033f3]> dmi
0x10000000 0x10035000  C:\Users\lab\Desktop\radare32\bin\todebug\drdrex.exe
0x771b0000 0x77353000  C:\Windows\SYSTEM32\ntdll.dll
0x75cf0000 0x75de0000  C:\Windows\System32\KERNEL32.DLL
0x759d0000 0x75be5000  C:\Windows\System32\KERNELBASE.dll
0x76c40000 0x76cff000  C:\Windows\System32\RPCRT4.dll
0x750d0000 0x7514a000  C:\Windows\System32\ADVAPI32.dll
0x768d0000 0x7698f000  C:\Windows\System32\msvcrt.dll
0x75950000 0x759c5000  C:\Windows\System32\sechost.dll
0x751b0000 0x75246000  C:\Windows\System32\OLEAUT32.dll
0x75de0000 0x75e5b000  C:\Windows\System32\msvcp_win.dll
0x76a70000 0x76b90000  C:\Windows\System32\ucrtbase.dll
0x76f10000 0x77191000  C:\Windows\System32\combase.dll
0x76420000 0x76465000  C:\Windows\System32\SHLWAPI.dll
0x75510000 0x7594c000  C:\Windows\System32\SETUPAPI.dll
0x75070000 0x750ab000  C:\Windows\System32\cfgmgr32.dll
0x76b90000 0x76ba9000  C:\Windows\System32\bcrypt.dll
0x766e0000 0x76880000  C:\Windows\System32\USER32.dll
0x750b0000 0x750c8000  C:\Windows\System32\win32u.dll
0x76c10000 0x76c34000  C:\Windows\System32\GDI32.dll
0x76d00000 0x76ddc000  C:\Windows\System32\gdi32full.dll
0x74410000 0x7444f000  C:\Windows\SYSTEM32\pdh.dll
0x73d00000 0x74150000  C:\Windows\SYSTEM32\WININET.dll
0x752c0000 0x752e5000  C:\Windows\System32\IMM32.DLL
```

OK so a bunch of interesting functions are getting loaded after the program starts, right before the unpacking process..

Unpacking related api-calls

Next thing, after we know that will be to identify and place breakpoints inside COMMON unpacking related functions. So in general during unpacking the final code to be executed USUALLY will be either written in memory overwritting the process memory or in another section (newly allocated memory), written inside other process memory (process injection) or maybe written on disk and ran (I would not recommend), so we should be looking for calls related to stuff like that.

In here we'll look for VirtualAlloc, that allocates new memory space, VirtualProtect used to set permissions on that memory and we can also look for stuff like CreateRemoteThread and NTResumeThread, createprocess stuff and the like. I would suggest you to go check MSDN to know more.

We can do it in radare2 like this:

```
[0x100033f3]> e search.in =  dbg.maps

[0x100033f3]> dmi KERNEL32 VirtualProtect
[Symbols]

nth  paddr      vaddr      bind   type size lib                              name
---------------------------------------------------------------------------------
1490 0x000114c0 0x75d104c0 GLOBAL FUNC 0    KERNEL32.dll
VirtualProtect
4    0x00067364 0x75d71364 NONE   FUNC 0    api-ms-win-core-memory-l1-1-0.dll
imp.VirtualProtect
[0x100033f3]> dmi KERNEL32 VirtualAlloc
[Symbols]

nth  paddr      vaddr      bind   type size lib                              name
---------------------------------------------------------------------------------
1484 0x000103c0 0x75d0f3c0 GLOBAL FUNC 0    KERNEL32.dll
VirtualAlloc
7    0x00067370 0x75d71370 NONE   FUNC 0    api-ms-win-core-memory-l1-1-0.dll
imp.VirtualAlloc
[0x100033f3]> dmi KERNEL32 IsDebuggerPresent
[Symbols]

nth paddr      vaddr      bind   type size lib                              name
--------------------------------------------------------------------------------
900 0x000130d0 0x75d120d0 GLOBAL FUNC 0    KERNEL32.dll
IsDebuggerPresent
2   0x00066ee4 0x75d70ee4 NONE   FUNC 0    api-ms-win-core-debug-l1-1-0.dll
imp.IsDebuggerPresent

[0x100033f3]> dmi KERNEL32 CreateRemoteThread
[Symbols]

nth paddr      vaddr      bind   type size lib
name
------------------------------------------------------------------------------------
----
237 0x00024b50 0x75d23b50 GLOBAL FUNC 0    KERNEL32.dll
CreateRemoteThread
10  0x00067518 0x75d71518 NONE   FUNC 0    api-ms-win-core-processthreads-l1-1-0.dll
imp.CreateRemoteThread
```

After identifying the positions of those, we can do “db” on their mem addrs to place breakpoints. So each time the program would call them we should be able to inspect their params and results.

Having said that, let’s do it and go use “dc” to move until the first call to VirtualAlloc:

```
[0x75d0f3c0]> pd 10
            ;-- ecx:
            ;-- eip:
            0x75d0f3c0 b    8bff           mov edi, edi
            0x75d0f3c2      55             push ebp
            0x75d0f3c3      8bec           mov ebp, esp
            0x75d0f3c5      5d             pop ebp
            0x75d0f3c6      ff257013d775   jmp dword [0x75d71370]
            0x75d0f3cc      cc             int3
            0x75d0f3cd      cc             int3
            0x75d0f3ce      cc             int3
            0x75d0f3cf      cc             int3
            0x75d0f3d0      cc             int3
[0x75d0f3c0]> dcr
hit breakpoint at: 0x75af4b01
[0x75af4b0c]> dr eax
0x00600000
[0x75af4b0c]> pxw @ 0x00600000
0x00600000  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600010  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600020  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600030  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600040  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600050  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600060  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600070  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600080  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600090  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x006000a0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x006000b0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x006000c0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x006000d0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x006000e0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x006000f0  0x00000000 0x00000000 0x00000000 0x00000000  ................
[0x75af4b0c]>
```

After we use “dcr” to run it until it returns, we see that the returned value stored in EAX refers to a newly set mem addr. Let’s note it.

Now to the next call:

```
[0x75d0f3c0]> dcr
hit breakpoint at: 0x75af4b01
[0x75af4b0c]> dr eax
0x00730000
[0x75af4b0c]> pxw @ 0x00730000
0x00730000  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00730010  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00730020  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00730030  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00730040  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00730050  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00730060  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00730070  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00730080  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00730090  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x007300a0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x007300b0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x007300c0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x007300d0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x007300e0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x007300f0  0x00000000 0x00000000 0x00000000 0x00000000  ................
```

Again, more memory allocated, we note it.

And now as the program moved on it is always useful to go check what’s been written (if it has) on the addr of the first VirtualAlloc:

```
[0x75af4b0c]> pxw 400 @ 0x00600000+0x200
0x00600200  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600210  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600220  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600230  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600240  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600250  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600260  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600270  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600280  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00600290  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x006002a0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x006002b0  0x00000000 0x80000000 0xc9be9b0f 0x0400009c  ................
0x006002c0  0xff000000 0xb80000ff 0x00000000 0x40000000  ...............@
0x006002d0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x006002e0  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x006002f0  0xc6000000 0x0e985092 0x000eba1f 0x21cd09b4  .....P.........!
0x00600300  0xcd4c01b8 0x69685421 0x72702073 0x6172676f  ..L.!This progra
0x00600310  0x6163206d 0x746f6e6e 0x20656220 0x206e7572  m cannot be run 
0x00600320  0x44206e69 0x6d20534f 0x2e65646f 0x240a0d0d  in DOS mode....$
0x00600330  0x00000000 0x28000000 0x831f44a7 0x838eab98  .......(.D......
0x00600340  0x838eab98 0x8a8eab98 0x808e38e0 0x838eab98  .........8......
0x00600350  0x818eaa98 0x838eab98 0x828eab98 0x8e8eab98  ................
0x00600360  0x828e75ca 0x528eab98 0x83686369 0x008eab98  .u.....Rich.....
0x00600370  0x00000000 0x00000000 0x00000000 0x78000000  ...............x
0x00600380  0xf50eead8 0xe5000ec7 0x005ea18b 0x00000000  ..........^.....
```

And we quickly see that a Windows executable has been written there! Interesting!

Let’s move on:

```
[0x75af4b0c]> dc
hit breakpoint at: 0x75d0f3c0
[0x75d0f3c0]> pd 10
            ;-- esi:
            ;-- eip:
            0x75d0f3c0 b    8bff           mov edi, edi
            0x75d0f3c2      55             push ebp
            0x75d0f3c3      8bec           mov ebp, esp
            0x75d0f3c5      5d             pop ebp
            0x75d0f3c6      ff257013d775   jmp dword [0x75d71370]
            0x75d0f3cc      cc             int3
            0x75d0f3cd      cc             int3
            0x75d0f3ce      cc             int3
            0x75d0f3cf      cc             int3
            0x75d0f3d0      cc             int3
[0x75d0f3c0]> dr eax
0x0060292a
[0x75d0f3c0]> pxw @ 0x0060292a
0x0060292a  0x3ee05a4d 0x000051ed 0x00000004 0x0000ffff  MZ.>.Q..........
0x0060293a  0x000000b8 0x00000000 0x00000040 0x00000000  ........@.......
0x0060294a  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x0060295a  0x00000000 0x00000000 0x00000000 0x000000e0  ................
0x0060296a  0x0eba1f0e 0xcd09b400 0x4c01b821 0x685421cd  ........!..L.!Th
0x0060297a  0x70207369 0x72676f72 0x63206d61 0x6f6e6e61  is program canno
0x0060298a  0x65622074 0x6e757220 0x206e6920 0x20534f44  t be run in DOS
0x0060299a  0x65646f6d 0x0a0d0d2e 0x00000024 0x00000000  mode....$.......
0x006029aa  0xb55e5b7e 0xe6303a3a 0xe6303a3a 0xe6303a3a  ~[^.::0.::0.::0.
0x006029ba  0xe6b34233 0xe6303a3b 0xe6a34233 0xe6303a3d  3B..;:0.3B..=:0.
0x006029ca  0xe6313a3a 0xe6303a36 0xe6303a3a 0xe6303a3b  ::1.6:0.::0.;:0.
0x006029da  0xe69ba721 0xe6303a1a 0xe6aba721 0xe6303a3b  !....:0.!...;:0.
0x006029ea  0xe6ada721 0xe6303a3b 0x68636952 0xe6303a3a  !...;:0.Rich::0.
0x006029fa  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00602a0a  0x00004550 0x0004014c 0x5e97ab1c 0x00000000  PE..L......^....
0x00602a1a  0x00000000 0x010200e0 0x00a7010b 0x00020e00  ................
[0x75d0f3c0]> dcr
hit breakpoint at: 0x75af4b01
[0x75af4b0c]> dr eax
0x00740000
[0x75af4b0c]>
```

Again one exe:

```
[0x75af4b0c]> pxw @ 0x00730000
0x00730000  0xbe9b5a4d 0x00009cc9 0x00000004 0x0000ffff  MZ..............
0x00730010  0x000000b8 0x00000000 0x00000040 0x00000000  ........@.......
0x00730020  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x00730030  0x00000000 0x00000000 0x00000000 0x000000c8  ................
0x00730040  0x0eba1f0e 0xcd09b400 0x4c01b821 0x685421cd  ........!..L.!Th
0x00730050  0x70207369 0x72676f72 0x63206d61 0x6f6e6e61  is program canno
0x00730060  0x65622074 0x6e757220 0x206e6920 0x20534f44  t be run in DOS
0x00730070  0x65646f6d 0x0a0d0d2e 0x00000024 0x00000000  mode....$.......
0x00730080  0x1f44a728 0x8eab9883 0x8eab9883 0x8eab9883  (.D.............
0x00730090  0x8e38e08a 0x8eab9880 0x8eaa9883 0x8eab9881  ..8.............
0x007300a0  0x8eab9883 0x8eab9882 0x8e75ca8e 0x8eab9882  ..........u.....
0x007300b0  0x68636952 0x8eab9883 0x00000000 0x00000000  Rich............
0x007300c0  0x00000000 0x00000000 0x00004550 0x000ec7f5  ........PE......
0x007300d0  0x5ea18be5 0x00000000 0x00000000 0xfa84a2ba  ...^............
0x007300e0  0x00cafe5c 0x00001800 0x00000a00 0x00000000  \...............
0x007300f0  0x00001e97 0x00001000 0x00003000 0x00400000  .........0....@.
[0x75af4b0c]>

RESULTS OF THE LAST CALL TO VIRTUALALLOC
[0x75af4b0c]> dr eax
0x01000000
[0x75d104c0]> pxw @ 0x01000000
0x01f00000  0x3ee05a4d 0x000051ed 0x00000004 0x0000ffff  MZ.>.Q..........
0x01f00010  0x000000b8 0x00000000 0x00000040 0x00000000  ........@.......
0x01f00020  0x00000000 0x00000000 0x00000000 0x00000000  ................
0x01f00030  0x00000000 0x00000000 0x00000000 0x000000e0  ................
0x01f00040  0x0eba1f0e 0xcd09b400 0x4c01b821 0x685421cd  ........!..L.!Th
0x01f00050  0x70207369 0x72676f72 0x63206d61 0x6f6e6e61  is program canno
0x01f00060  0x65622074 0x6e757220 0x206e6920 0x20534f44  t be run in DOS
0x01f00070  0x65646f6d 0x0a0d0d2e 0x00000024 0x00000000  mode....$.......
0x01f00080  0xb55e5b7e 0xe6303a3a 0xe6303a3a 0xe6303a3a  ~[^.::0.::0.::0.
0x01f00090  0xe6b34233 0xe6303a3b 0xe6a34233 0xe6303a3d  3B..;:0.3B..=:0.
0x01f000a0  0xe6313a3a 0xe6303a36 0xe6303a3a 0xe6303a3b  ::1.6:0.::0.;:0.
0x01f000b0  0xe69ba721 0xe6303a1a 0xe6aba721 0xe6303a3b  !....:0.!...;:0.
0x01f000c0  0xe6ada721 0xe6303a3b 0x68636952 0xe6303a3a  !...;:0.Rich::0.
```

And more stuff on the second allocated space. At this point we may try to dump those executables in memory and analyze them but at the end we want to get to the payload **that is getting executed** to avoid wasting time in decoys/useles stuff etc so let’s see if we get there at some point in the code!

We move on with “dc” and we reach VirtualProtect:

```
[0x75af4b0c]> dc
hit breakpoint at: 0x75d104c0
[0x75d104c0]> pd 10
            ;-- eax:
            ;-- eip:
            0x75d104c0 b    8bff           mov edi, edi
            0x75d104c2      55             push ebp
            0x75d104c3      8bec           mov ebp, esp
            0x75d104c5      5d             pop ebp
            0x75d104c6      ff256413d775   jmp dword [0x75d71364]
            0x75d104cc      cc             int3
            0x75d104cd      cc             int3
            0x75d104ce      cc             int3
            0x75d104cf      cc             int3
            0x75d104d0      cc             int3
[0x75d104c0]> dr
edi = 0x0019fd80
esi = 0x00000000
ebx = 0x10000000
edx = 0x0019fda0
ecx = 0x00000002
eax = 0x75d104c0
ebp = 0x0019fdd0
eip = 0x75d104c0
eflags = 0x00000244
esp = 0x0019fd04
[0x75d104c0]>
[0x75d104c0]> pxr @ esp
0x0019fd04 0x007324a3  .$s. @ esp PRIVATE  R W X 'sub esp, 0x10' 'PRIVATE '
0x0019fd08 0x10000000  .... IMAGE  ebx R 0x905a4d
0x0019fd0c 0x00035000  .P..
0x0019fd10 0x00000004  .... 4
0x0019fd14 0x0019fda0  .... PRIVATE  edx R W 0x0
0x0019fd18 0x0019fe9c  .... PRIVATE  R W 0x75d0f550
0x0019fd1c 0x00762000  . v. PRIVATE  ascii ('
```

VirtualProtect changes the protection on a region of committed pages in the virtual address space of the calling process.Malware needs to change permission of the region that was reserved by VirtualAlloc before injecting it into some other legitimate executable. Thus VirtualAlloc is used here as a complimentary API to change permission of allocated memory to read-write-execute. So in here we see that the results of the last VirtualAlloc call are getting baked for execution.

We can move on to several VirtualProtect calls:

```
[0x75d104c0]> dcr
hit breakpoint at: 0x75af38ba
[0x75af38c9]> pd 10
            ;-- eip:
            0x75af38c9      c21000         ret 0x10
            0x75af38cc      cc             int3
            0x75af38cd      cc             int3
            0x75af38ce      cc             int3
            0x75af38cf      cc             int3
            0x75af38d0      cc             int3
            0x75af38d1      cc             int3
            0x75af38d2      cc             int3
            0x75af38d3      cc             int3
            0x75af38d4      cc             int3
[0x75af38c9]> ds
[0x007324a3]> pd 10
            ;-- eip:
            0x007324a3      83ec10         sub esp, 0x10
            0x007324a6      c745b0010000.  mov dword [ebp - 0x50], 1
            0x007324ad      0f2805103073.  movaps xmm0, xmmword [0x733010] ;
[0x733010:16]=-1
            0x007324b4      0f1145c0       movups xmmword [ebp - 0x40], xmm0
```

At this point we may want to follow the execution flow to check if the program is hitting a suspicious jmp/call/ret(without poping registers back!) as it may indicate a jmp to the unpacked code…

```
[0x007324a3]> pd 80
            ;-- eip:
            0x007324a3      83ec10         sub esp, 0x10
            0x007324a6      c745b0010000.  mov dword [ebp - 0x50], 1
            0x007324ad      0f2805103073.  movaps xmm0, xmmword [0x733010] ;
[0x733010:16]=-1
            0x007324b4      0f1145c0       movups xmmword [ebp - 0x40], xmm0
            0x007324b8      8b4dac         mov ecx, dword [ebp - 0x54]
            0x007324bb      8b5104         mov edx, dword [ecx + 4]
            0x007324be      8b75a0         mov esi, dword [ebp - 0x60]
            0x007324c1      893424         mov dword [esp], esi
            0x007324c4      c74424040000.  mov dword [esp + 4], 0
            0x007324cc      89542408       mov dword [esp + 8], edx
            0x007324d0      894584         mov dword [ebp - 0x7c], eax
            0x007324d3      e818edffff     call 0x7311f0
            0x007324d8      8b45ac         mov eax, dword [ebp - 0x54]
            0x007324db      8b4850         mov ecx, dword [eax + 0x50]
            0x007324de      8b55a0         mov edx, dword [ebp - 0x60]
            0x007324e1      891424         mov dword [esp], edx
            0x007324e4      894c2404       mov dword [esp + 4], ecx
            0x007324e8      8b4d9c         mov ecx, dword [ebp - 0x64]
            0x007324eb      894c2408       mov dword [esp + 8], ecx
            0x007324ef      e8fbf0ffff     call 0x7315ef
            0x007324f4      8b45ac         mov eax, dword [ebp - 0x54]
            0x007324f7      8b4850         mov ecx, dword [eax + 0x50]
            0x007324fa      890c24         mov dword [esp], ecx
            0x007324fd      c74424040000.  mov dword [esp + 4], 0
            0x00732505      8b4d9c         mov ecx, dword [ebp - 0x64]
            0x00732508      894c2408       mov dword [esp + 8], ecx
            0x0073250c      e8dfecffff     call 0x7311f0
            0x00732511      8b45ec         mov eax, dword [ebp - 0x14]
            0x00732514      35ee799d30     xor eax, 0x309d79ee
            0x00732519      8b4da0         mov ecx, dword [ebp - 0x60]
            0x0073251c      890c24         mov dword [esp], ecx
            0x0073251f      89442404       mov dword [esp + 4], eax
            0x00732523      c74424080200.  mov dword [esp + 8], 2
            0x0073252b      8b458c         mov eax, dword [ebp - 0x74]
            0x0073252e      8944240c       mov dword [esp + 0xc], eax
            0x00732532      8b55a4         mov edx, dword [ebp - 0x5c]
            0x00732535      ffd2           call edx
            0x00732537      83ec10         sub esp, 0x10
            0x0073253a      66be6986       mov si, 0x8669
            0x0073253e      8b4da0         mov ecx, dword [ebp - 0x60]
            0x00732541      8b513c         mov edx, dword [ecx + 0x3c]
            0x00732544      6689d7         mov di, dx
            0x00732547      662b75f2       sub si, word [ebp - 0xe]
            0x0073254b      6639f7         cmp di, si
            0x0073254e      894580         mov dword [ebp - 0x80], eax
            0x00732551      89957cffffff   mov dword [ebp - 0x84], edx
            0x00732557      898d78ffffff   mov dword [ebp - 0x88], ecx
        ,=< 0x0073255d      747a           je 0x7325d9
       ,==< 0x0073255f      e9ac000000     jmp 0x732610
       ||   0x00732564      31c0           xor eax, eax
       ||   0x00732566      b964000000     mov ecx, 0x64               ; 'd' ; 100
       ||   0x0073256b      8b55a0         mov edx, dword [ebp - 0x60]
       ||   0x0073256e      8b75ac         mov esi, dword [ebp - 0x54]
       ||   0x00732571      0316           add edx, dword [esi]
       ||   0x00732573      8b7e28         mov edi, dword [esi + 0x28]
       ||   0x00732576      897dd4         mov dword [ebp - 0x2c], edi
       ||   0x00732579      8b7e60         mov edi, dword [esi + 0x60]
       ||   0x0073257c      897dd8         mov dword [ebp - 0x28], edi
       ||   0x0073257f      8b7e08         mov edi, dword [esi + 8]
       ||   0x00732582      897ddc         mov dword [ebp - 0x24], edi
       ||   0x00732585      8b7e40         mov edi, dword [esi + 0x40]
       ||   0x00732588      897de0         mov dword [ebp - 0x20], edi
       ||   0x0073258b      8b7e0c         mov edi, dword [esi + 0xc]
       ||   0x0073258e      897de4         mov dword [ebp - 0x1c], edi
       ||   0x00732591      8955e8         mov dword [ebp - 0x18], edx
       ||   0x00732594      893424         mov dword [esp], esi
       ||   0x00732597      c74424040000.  mov dword [esp + 4], 0
       ||   0x0073259f      c74424086400.  mov dword [esp + 8], 0x64   ; 'd'
       ||                                                             ; [0x64:4]=-1
; 100
       ||   0x007325a7      898574ffffff   mov dword [ebp - 0x8c], eax
       ||   0x007325ad      898d70ffffff   mov dword [ebp - 0x90], ecx
       ||   0x007325b3      e838ecffff     call 0x7311f0
       ||   0x007325b8      8d45d4         lea eax, [ebp - 0x2c]
       ||   0x007325bb      8b30           mov esi, dword [eax]
       ||   0x007325bd      8b7804         mov edi, dword [eax + 4]
       ||   0x007325c0      8b5808         mov ebx, dword [eax + 8]
       ||   0x007325c3      8b680c         mov ebp, dword [eax + 0xc]
       ||   0x007325c6      8b6010         mov esp, dword [eax + 0x10]
       ||   0x007325c9      8b4014         mov eax, dword [eax + 0x14]
       ||   0x007325cc      ffe0           jmp eax
```

Getting to the unpacked code

So for example, those call edx and jmp eax look interesting

```
            0x00732535      ffd2           call edx

       ||   0x007325cc      ffe0           jmp eax
```

```
[0x007324a3]> dc
hit breakpoint at: 0x75d104c0
[0x75d104c0]> dr
edi = 0x0019fd80
esi = 0x10000000
ebx = 0x10000000
edx = 0x75d104c0
ecx = 0x10000000
eax = 0x0019fda0
ebp = 0x0019fdd0
eip = 0x75d104c0
eflags = 0x00000204
esp = 0x0019fd04
[0x75d104c0]> pxr @ esp
0x0019fd04 0x00732537  7%s. @ esp PRIVATE  ascii ('7') R W X 'sub esp, 0x10' 'PRIVATE
'
0x0019fd08 0x10000000  .... IMAGE  ebx,esi,ecx W
0x0019fd0c 0x00000400  .... 1024
```

And after inspecting the code we see that several VirtualProtect calls are taking place and they are always getting back to the same chunk of code. Those calls as we follow the execution flow are basicall setting R, RW, RWX permissions on several areas of the program (.text,.data…)

```
[0x00732537]> dc
hit breakpoint at: 0x75d104c0
[0x75d104c0]> pxr @ esp
0x0019fd04 0x007326db  .&s. @ esp PRIVATE  R W X 'sub esp, 0x10' 'PRIVATE '
0x0019fd08 0x10001000  .... IMAGE  .text section..text,fcn.10001000,ecx fcn.10001000
R W 0x8bec8b55
0x0019fd0c 0x00020d11  .... MAPPED  ebx R 0x0
```

As we see:

```
[0x75af38c9]> dr eax
0x00000001
[0x75af38c9]> dc
hit breakpoint at: 0x75d104c0
[0x75d104c0]> pxr @ esp
0x0019fd04 0x007326db  .&s. @ esp PRIVATE  R W X 'sub esp, 0x10' 'PRIVATE '
0x0019fd08 0x10022000  . .. IMAGE  .text1 ecx R W 0x750eed20

[...]

[0x75d104c0]> pxr @ esp
0x0019fd04 0x007326db  .&s. @ esp PRIVATE  R W X 'sub esp, 0x10' 'PRIVATE '
0x0019fd08 0x10028000  .... IMAGE  .text1 ecx R W 0x6376b30c
```

Aaaand at some point the unpacked program is fully loaded and we hit the jump to eax:

```
[0x007326db]> dc
hit breakpoint at: 0x7325cc
[0x007325cc]> pd 10
            ;-- eip:
            0x007325cc b    ffe0           jmp eax
            0x007325ce      81c4bc000000   add esp, 0xbc
            0x007325d4      5f             pop edi
            0x007325d5      5e             pop esi
            0x007325d6      5b             pop ebx
            0x007325d7      5d             pop ebp
            0x007325d8      c3             ret
            0x007325d9      8b8578ffffff   mov eax, dword [ebp - 0x88]
            0x007325df      8b4d98         mov ecx, dword [ebp - 0x68]
```

And what’s in there????

```
[0x007325cc]> dr eax
0x100062b0
```

That is, our program:

```
0x00ce4000 - 0x02041000 - usr  19.4M s --- MAPPED  ?
0x10000000 - 0x10001000 - usr     4K s r-- IMAGE  ?
0x10001000 - 0x10022000 - usr   132K s r-x IMAGE  ? ; fcn.10001000
0x10022000 - 0x10028000 - usr    24K s r-- IMAGE  ?
0x10028000 - 0x10029000 - usr     4K s rw- IMAGE  ?
0x10029000 - 0x1002a000 - usr     4K s r-- IMAGE  ?
0x1002a000 - 0x10035000 - usr    44K s rw- IMAGE  ?
```

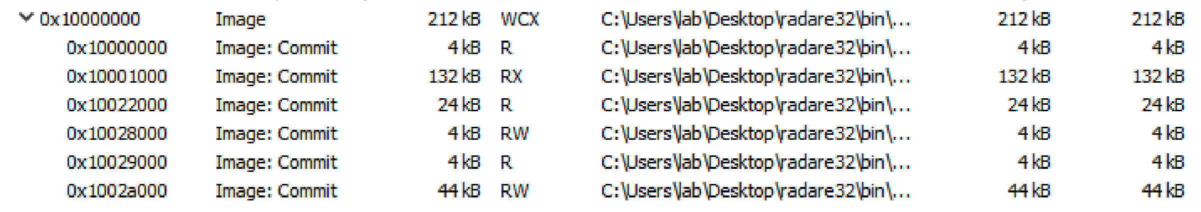
Dumping the unpacked code

So at this point we are confident that we have our program unpacked:

```
[0x007325cc]> ds
[0x100062b0]> pd 10
            ;-- eax:
            ;-- eip:
            0x100062b0      55             push ebp
            0x100062b1      8bec           mov ebp, esp
            0x100062b3      51             push ecx
            0x100062b4      6a00           push 0
            0x100062b6      8d4dfc         lea ecx, [ebp - 4]
            0x100062b9      e852dd0000     call 0x10014010
            0x100062be      a008800210     mov al, byte [0x10028008]   ;
[0x10028008:1]=0
            0x100062c3      84c0           test al, al
        ,=< 0x100062c5      746c           je 0x10006333
        |   0x100062c7      837d0c01       cmp dword [ebp + 0xc], 1
```

What comes next is dumping it to a new binary, so we can move to reversing it.

This can be done by using process hacker from the FLARE-VM or calling dmd in radare2:

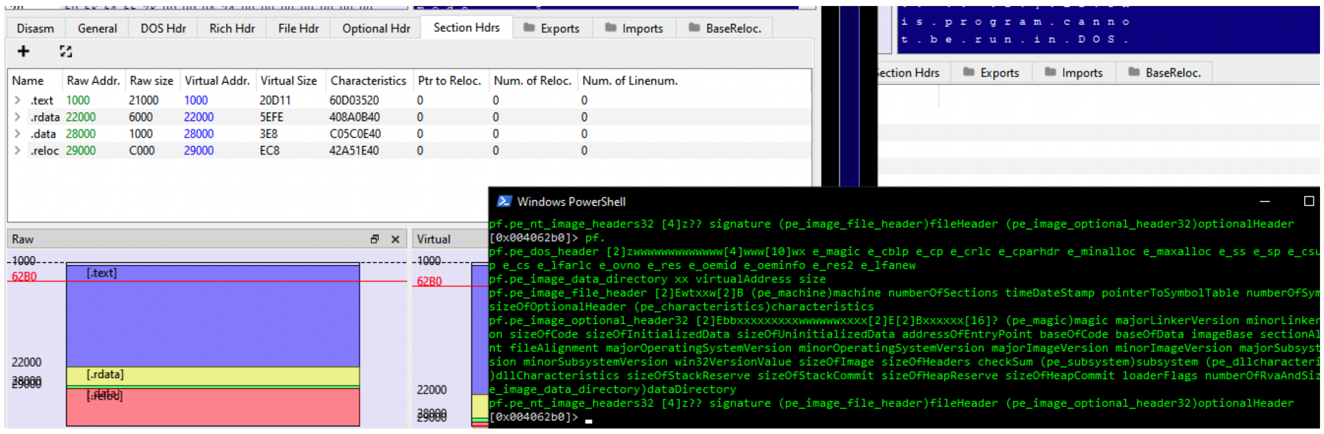


0x10000000	Image	212 kB	WCX	C:\Users\lab\Desktop\radare32\bin\...	212 kB	212 kB
0x10000000	Image: Commit	4 kB	R	C:\Users\lab\Desktop\radare32\bin\...	4 kB	4 kB
0x10001000	Image: Commit	132 kB	RX	C:\Users\lab\Desktop\radare32\bin\...	132 kB	132 kB
0x10022000	Image: Commit	24 kB	R	C:\Users\lab\Desktop\radare32\bin\...	24 kB	24 kB
0x10028000	Image: Commit	4 kB	RW	C:\Users\lab\Desktop\radare32\bin\...	4 kB	4 kB
0x10029000	Image: Commit	4 kB	R	C:\Users\lab\Desktop\radare32\bin\...	4 kB	4 kB
0x1002a000	Image: Commit	44 kB	RW	C:\Users\lab\Desktop\radare32\bin\...	44 kB	44 kB

```
dmda
[...]
Dumped 4096 byte(s) into 0x10000000-0x10001000-r--.dmp
Dumped 135168 byte(s) into 0x10001000-0x10022000-r-x.dmp
Dumped 24576 byte(s) into 0x10022000-0x10028000-r--.dmp
Dumped 4096 byte(s) into 0x10028000-0x10029000-rw-.dmp
Dumped 4096 byte(s) into 0x10029000-0x1002a000-r--.dmp
Dumped 45056 byte(s) into 0x1002a000-0x10035000-rw-.dmp
```

If we go for the radare2 option, cat can be used to concat those chunks into one exe

What comes next is to fix the program by adjusting the section headers to its real addr/sizes (match Raw/Virtual addrs and adjust the size). Again this can be done using PEBEAR for example, or by manually editing the binary (don't forget to start radare2 with the -w option)



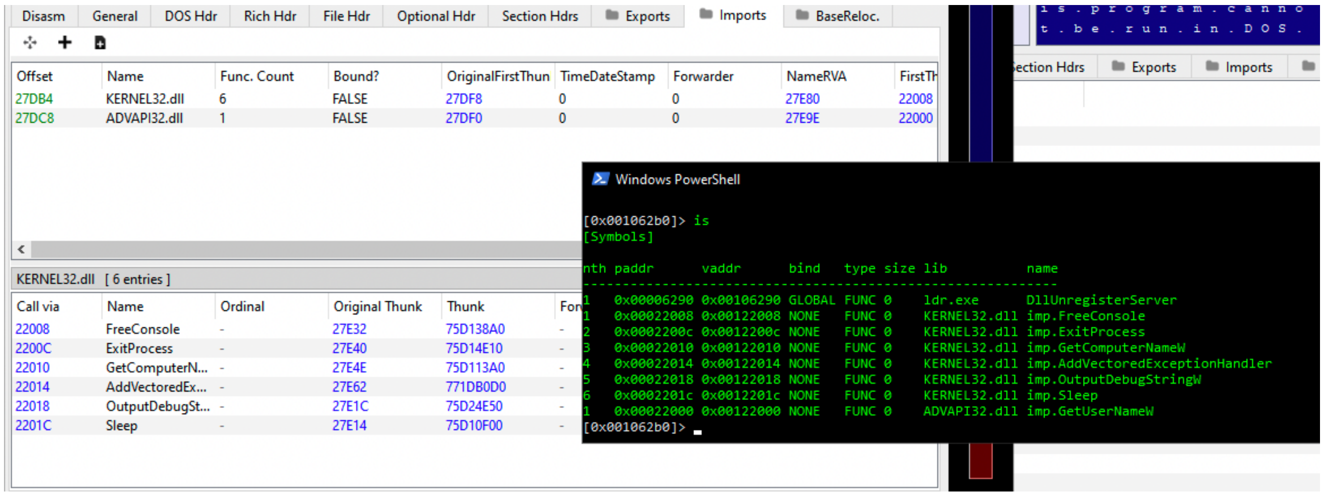
Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.text	1000	21000	1000	20D11	60D03520	0	0	0
.rdata	22000	6000	22000	5EFE	408A0B40	0	0	0
.data	28000	1000	28000	3E8	C05C0E40	0	0	0
.reloc	29000	C000	29000	EC8	42A51E40	0	0	0

```
pf.pe_nt_image_headers32 [4]z?? signature (pe_image_file_header)fileHeader (pe_image_optional_header32)optionalHeader
[0x004062b0]> pf.
pf.pe_dos_header [2]zwwwwwwwwwwwww[4]www[10]wx e_magic e_cblp e_cp e_crlc e_cparhdr e_minalloc e_maxalloc e_ss e_sp e_csu
p e_cs e_lfarlc e_ovno e_res e_oemid e_oeminfo e_res2 e_lfanew
pf.pe_image_data_directory xx virtualAddress size
pf.pe_image_file_header [2]Ewtxxw[2]B (pe_machine)machine numberOfSections timeDateStamp pointerToSymbolTable numberOfSym
sizeOfOptionalHeader (pe_characteristics)characteristics
pf.pe_image_optional_header32 [2]Ebbxxxxxxxxxwwwwwwxxxx[2]E[2]Bxxxxxx[16]? (pe_magic)magic majorLinkerVersion minorLinker
on sizeOfCode sizeOfInitializedData sizeOfUninitializedData addressOfEntryPoint baseOfCode baseOfData imageBase sectionAl
nt fileAlignment majorOperatingSystemVersion minorOperatingSystemVersion majorImageVersion minorImageVersion majorSubsyst
sion minorSubsystemVersion win32VersionValue sizeOfImage sizeOfHeaders checkSum (pe_subsystem)subsystem (pe_dllcharacteri
)dllCharacteristics sizeOfStackReserve sizeOfStackCommit sizeOfHeapReserve sizeOfHeapCommit loaderFlags numberOfRvaAndSiz
e_image_data_directory)dataDirectory
pf.pe_nt_image_headers32 [4]z?? signature (pe_image_file_header)fileHeader (pe_image_optional_header32)optionalHeader
[0x004062b0]>
```

```
[0x004062b0]> pf.
pf.pe_dos_header [2]zwwwwwwwwwwwwww[4]www[10]wx e_magic e_cblp e_cp e_crlc e_cparhdr
e_minalloc e_maxalloc e_ss e_sp e_csum e_ip e_cs e_lfarlc e_ovno e_res e_oemid
e_oeminfo e_res2 e_lfanew
pf.pe_image_data_directory xx virtualAddress size
pf.pe_image_file_header [2]Ewtxxw[2]B (pe_machine)machine numberOfSections
timeDateStamp pointerToSymbolTable numberOfSymbols sizeOfOptionalHeader
(pe_characteristics)characteristics
pf.pe_image_optional_header32 [2]Ebbxxxxxxxxxwwwwwwxxxx[2]E[2]Bxxxxxx[16]?
(pe_magic)magic majorLinkerVersion minorLinkerVersion sizeOfCode
sizeOfInitializedData sizeOfUninitializedData addressOfEntryPoint baseOfCode
baseOfData imageBase sectionAlignment fileAlignment majorOperatingSystemVersion
minorOperatingSystemVersion majorImageVersion minorImageVersion majorSubsystemVersion
minorSubsystemVersion win32VersionValue sizeOfImage sizeOfHeaders checkSum
(pe_subsystem)subsystem (pe_dllcharacteristics)dllCharacteristics sizeOfStackReserve
sizeOfStackCommit sizeOfHeapReserve sizeOfHeapCommit loaderFlags numberOfRvaAndSizes
(pe_image_data_directory)dataDirectory
pf.pe_nt_image_headers32 [4]z?? signature (pe_image_file_header)fileHeader
(pe_image_optional_header32)optionalHeader
[0x004062b0]>
```

The base address needs to be adjusted (0x10000000 in our case) to the one we saw when analysing the program (as the references will use it!).

If we do it correctly, we should see the following, all the imports resolving:



Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwarder	NameRVA	FirstTh
27DB4	KERNEL32.dll	6	FALSE	27DF8	0	0	27E80	22008
27DC8	ADVAPI32.dll	1	FALSE	27DF0	0	0	27E9E	22000

KERNEL32.dll [6 entries]

Call via	Name	Ordinal	Original Thunk	Thunk	For
22008	FreeConsole	-	27E32	75D138A0	-
2200C	ExitProcess	-	27E40	75D14E10	-
22010	GetComputerN...	-	27E4E	75D113A0	-
22014	AddVectoredEx...	-	27E62	771DB0D0	-
22018	OutputDebugSt...	-	27E1C	75D24E50	-
2201C	Sleep	-	27E14	75D10F00	-

```
[0x001062b0]> is
[Symbols]

nth paddr      vaddr      bind   type size lib          name
------------------------------------------------------------
1   0x00006290 0x00106290 GLOBAL FUNC 0    ldr.exe      DllUnregisterServer
1   0x00022008 0x00122008 NONE   FUNC 0    KERNEL32.dll imp.FreeConsole
2   0x0002200c 0x0012200c NONE   FUNC 0    KERNEL32.dll imp.ExitProcess
3   0x00022010 0x00122010 NONE   FUNC 0    KERNEL32.dll imp.GetComputerNameW
4   0x00022014 0x00122014 NONE   FUNC 0    KERNEL32.dll imp.AddVectoredExceptionHandler
5   0x00022018 0x00122018 NONE   FUNC 0    KERNEL32.dll imp.OutputDebugStringW
6   0x0002201c 0x0012201c NONE   FUNC 0    KERNEL32.dll imp.Sleep
1   0x00022000 0x00122000 NONE   FUNC 0    ADVAPI32.dll imp.GetUserNameW
[0x001062b0]>
```

And from there….. We can move to the new executable

```
PS C:\Users\lab\Desktop > radare2 -AAA .\fixed_dridex.exe
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze all functions arguments/locals
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information (aanr)
[x] Finding function preludes
[x] Enable constraint types analysis for variables
[0x100062b0]> pd 10
            ;-- eip:
/ 214: entry0 (int32_t arg_8h, uint32_t arg_ch);
| rg: 0 (vars 0, args 0)
| bp: 3 (vars 1, args 2)
| sp: 0 (vars 0, args 0)
|           0x100062b0      55             push ebp
|           0x100062b1      8bec           mov ebp, esp
|           0x100062b3      51             push ecx
|           0x100062b4      6a00           push 0                      ; int32_t
arg_8h
|           0x100062b6      8d4dfc         lea ecx, [var_4h]
|           0x100062b9      e852dd0000     call fcn.10014010
|           0x100062be      a008800210     mov al, byte [0x10028008]   ;
[0x10028008:1]=0
|           0x100062c3      84c0           test al, al
|       ,=< 0x100062c5      746c           je 0x10006333
|       |   0x100062c7      837d0c01       cmp dword [arg_ch], 1
[0x100062b0]>
```

Now the real fun begins :)