

A Detailed Analysis of The LockBit Ransomware

Prepared by: Vlad Pasca, LIFARS, LLC

Date: 02/14/2022

www.LIFARS.com

info@lifars.com



©2022 SecurityScorecard Inc.

244 Fifth Avenue, Suite 2035,

New York, NY 10001

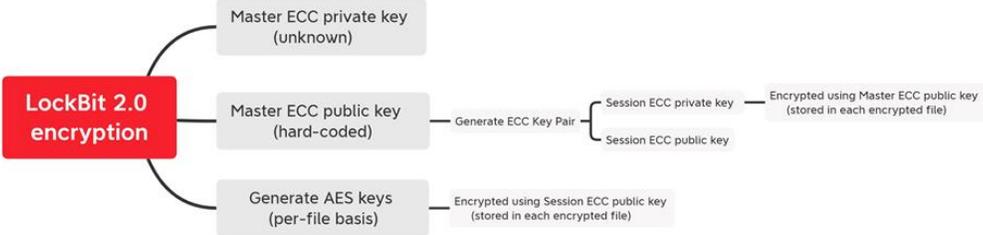
1.212.222.7061

Table of Contents

Executive Summary	2
Analysis and Findings	2
Thread activity – sub_4DF310 function	10
Thread activity – sub_4C3430 function	14
Thread activity – sub_4A2EC0 function	19
Thread activity – sub_45C960 function.....	28
Thread activity – sub_497060 function.....	34
Thread activity – sub_49E730 function.....	39
Printing ransom notes	44
LockBit Wallpaper Setup	46
Extract and save the HTA ransom note to Desktop	52
Indicators of Compromise	59
Registry Keys	59
Files Created	59
Processes spawned	59
Mutex.....	60
LockBit 2.0 Extension.....	60
LockBit 2.0 Ransom Note.....	60
Appendix	61
List of processes to be killed.....	61
List of services to be stopped.....	61

Executive Summary

LockBit 2.0 ransomware is one of the most active families in the wild and pretends to implement the fastest encryption algorithms using multithreading with I/O completion ports. The malware doesn't encrypt systems from CIS countries and can perform UAC bypass on older Windows versions if running with insufficient privileges. A hidden window that logs different actions performed by LockBit is created and might be activated using the Shift+F1 shortcut. The ransomware mounts all hidden volumes and stops a list of targeted processes and services. The malware generates a pair of ECC (Curve25519) session keys, with the private key being encrypted using a hard-coded ECC public key and stored in the registry. The binary deletes all Volume Shadow Copies using vssadmin and clears the Windows security application and system logs. LockBit obtains a list of physical printers used to print multiple ransom notes. The encrypted files have the ".lockbit" extension, and only the first 4KB of the file will be encrypted using the AES algorithm. A unique AES key is generated for each file, encrypted using the session ECC public key, and stored in each encrypted file.



Analysis and Findings

SHA256: 9feed0c7fa8c1d32390e1c168051267df61f11b048ec62aa5b8e66f60e8083af

The malware verifies whether it's being debugged by checking the NtGlobalFlag field from the PEB (process environment block). If the debugger is detected, the process jumps to an infinite loop:

```

.text:0048FF90 var_424= xmmword ptr -424h
.text:0048FF90 var_20C= dword ptr -20Ch
.text:0048FF90 var_208= byte ptr -208h
.text:0048FF90
.text:0048FF90 push    ebp
.text:0048FF91 mov     ebp, esp
.text:0048FF93 and     esp, 0FFFFFF8h
.text:0048FF96 mov     eax, large fs:30h
.text:0048FF9C sub     esp, 480h
.text:0048FFA2 test    byte ptr [eax+60h], 70h
.text:0048FFA6 push    esi
.text:0048FFA7 push    edi
.text:0048FFA8 jz     short loc_4BFFB2

.text:0048FFAA nop    word ptr [eax+eax+00h]

.text:0048FFB0
.text:0048FFB0 loc_4BFFB0:
.text:0048FFB0 jmp     short loc_4BFFB0
    
```

Figure 1

The encrypted strings are stored as stack strings and will be decrypted using the XOR operator. An example of a decryption algorithm is shown in figure 2, along with the decrypted DLL name:

```

EIP → 004BFB2 C7 84 24 E8 00 00 00 20 00 mov dword ptr ss:[esp+E8],20
004BFB3 33 F6 xor esi,esi
004BFB4 C6 84 24 EC 00 00 00 47 mov byte ptr ss:[esp+EC],47
004BFB5 C6 84 24 ED 00 00 00 44 mov byte ptr ss:[esp+ED],44
004BFB6 C6 84 24 EE 00 00 00 49 mov byte ptr ss:[esp+EE],49
004BFB7 C6 84 24 EF 00 00 00 50 mov byte ptr ss:[esp+EF],50
004BFB8 C6 84 24 F0 00 00 00 4C mov byte ptr ss:[esp+F0],4C
004BFB9 C6 84 24 F1 00 00 00 55 mov byte ptr ss:[esp+F1],55
004BFB0 C6 84 24 F2 00 00 00 53 mov byte ptr ss:[esp+F2],53
004BFB1 C6 84 24 F3 00 00 00 0E mov byte ptr ss:[esp+F3],E
004BFB2 C6 84 24 F4 00 00 00 44 mov byte ptr ss:[esp+F4],44
004C0007 C6 84 24 F5 00 00 00 4C mov byte ptr ss:[esp+F5],4C
004C000F C6 84 24 F6 00 00 00 4C mov byte ptr ss:[esp+F6],4C
004C0017 8A 84 24 EC 00 00 00 mov al,byte ptr ss:[esp+EC]
004C001E C6 84 24 F7 00 00 00 00 mov byte ptr ss:[esp+F7],0
004C0026 66 66 0F 1F 84 00 00 00 00 nop word ptr ds:[eax+eax],ax
004C0030 8A 94 34 EC 00 00 00 mov dl,byte ptr ss:[esp+esi+EC]
004C0037 8B 84 24 E8 00 00 00 mov eax,dword ptr ss:[esp+E8]
004C003E 0F BE C8 movsx ecx,al
004C0041 0F BE C2 movsx eax,dl
004C0044 33 C8 xor ecx,eax
004C0046 8B 8C 34 EC 00 00 00 mov byte ptr ss:[esp+esi+EC],cl
004C004D 46 inc esi
004C004E 83 FE 0B cmp esi,B
004C0051 72 DD jb lockbit.4C0030
004C0053 A1 1C 08 4F 00 mov eax,dword ptr ds:[4F081C]
004C0058 C6 84 24 F7 00 00 00 00 mov byte ptr ss:[esp+F7],0
004C0060 85 C0 test eax,eax
004C0062 75 0A jne lockbit.4C006E
004C0064 E8 A7 19 F5 FF call lockbit.411A10
Address Hex ASCII
0019FB00 20 00 00 00 67 64 69 70 6C 75 73 2E 64 6C 00 ...gdipus.dll
  
```

Figure 2

The binary implements the API hashing technique to hide the API functions used. As we can see below, the malware computes a 4-byte hash value and compares it with a hard-coded one (0xA3E6F6C3 in this case):

```

.text:00411A30
.text:00411A30 loc_411A30:
.text:00411A30 movzx  edx, word ptr [edi+2Ch]
.text:00411A34 xor     esi, esi
.text:00411A36 mov     eax, [edi+30h]
.text:00411A39 xor     ebx, ebx
.text:00411A3B shr     edx, 1
.text:00411A3D mov     [ebp+var_4], 811C9DC5h
.text:00411A44 lea    ecx, [eax+edx*2]
.text:00411A47 cmp     eax, ecx
.text:00411A49 cmova  edx, esi
.text:00411A4C mov     [ebp+var_C], edx
.text:00411A4F test   edx, edx
.text:00411A51 jz     short loc_411A87

.text:00411A53 mov     edi, edx

.text:00411A55
.text:00411A55 loc_411A55:
.text:00411A55 mov     dl, [eax]
.text:00411A57 lea    eax, [eax+2]
.text:00411A5A movsx  esi, dl
.text:00411A5D sub     dl, 41h ; 'A'
.text:00411A60 mov     ecx, esi
.text:00411A62 or     ecx, 20h
.text:00411A65 cmp     dl, 19h
.text:00411A68 cmova  ecx, esi
.text:00411A6B inc     ebx
.text:00411A6C xor     ecx, [ebp+var_4]
.text:00411A6F imul  ecx, 1000193h
.text:00411A75 mov     [ebp+var_4], ecx
.text:00411A78 cmp     ebx, edi
.text:00411A7A jnz    short loc_411A55

.text:00411A7C mov     edi, [ebp+var_8]
.text:00411A7F cmp     ecx, 0A3E6F6C3h
.text:00411A85 jz     short loc_411AA2
  
```

Figure 3

The malicious executable loads multiple DLLs into the address space of the process using the LoadLibraryA API:

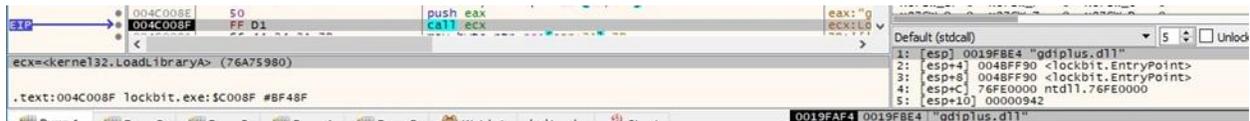


Figure 4

The following DLLs have been loaded: "gdiplus.dll", "ws2_32.dll", "shell32.dll", "advapi32.dll", "user32.dll", "ole32.dll", "netapi32.dll", "gpedit.dll", "oleaut32.dll", "shlwapi.dll", "msvcrt.dll", "activeds.dll", "mpr.dll", "bcrypt.dll", "crypt32.dll", "iphlpapi.dll", "wtsapi32.dll", "win32u.dll", "Comdlg32.dll", "cryptbase.dll", "combase.dll", "Winspool.drv".

GetSystemDefaultUILanguage is utilized to retrieve the language identifier for the system default UI language of the OS. The return value is compared with multiple identifiers that correspond to CIS countries (LockBit doesn't encrypt these systems):



Figure 5

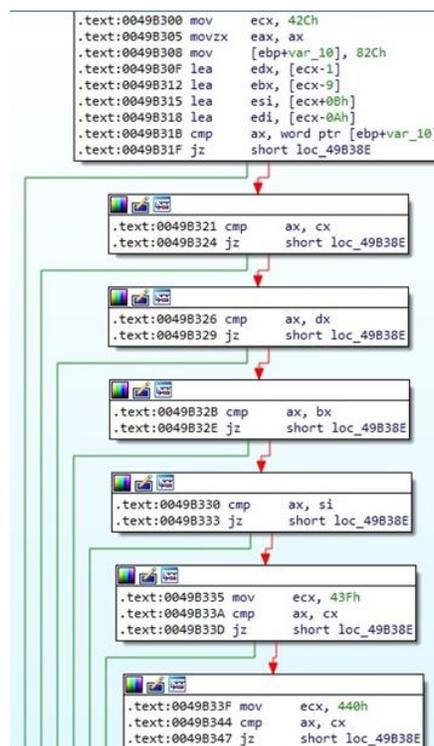


Figure 6

The following language identifiers have been found:

- 0x82c - Azerbaijani (Cyrillic)
- 0x42c - Azerbaijani (Latin)
- 0x42b – Armenian

- 0x423 – Belarusian
- 0x437 – Georgian
- 0x43F – Kazakh
- 0x440 – Kyrgyz
- 0x819 - Russian (Moldova)
- 0x419 – Russian
- 0x428 – Tajik
- 0x442 – Turkmen
- 0x843 - Uzbek (Cyrillic)
- 0x443 - Uzbek (Latin)
- 0x422 – Ukrainian

The GetUserDefaultUILanguage routine extracts the language identifier for the user UI language for the current user. The extracted value is compared with the same identifiers from above:

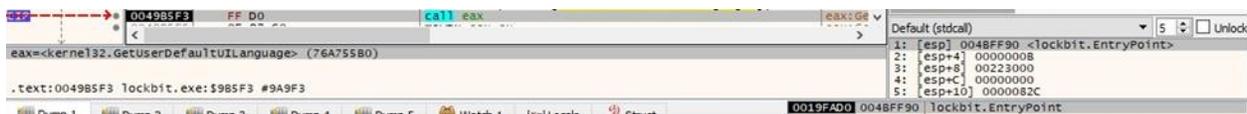


Figure 7

The NtQuerySystemInformation function is utilized to retrieve the number of processors in the system (0x0 = **SystemBasicInformation**):



Figure 8

The binary opens a handle to the current process (0x60000 = **WRITE_DAC | READ_CONTROL**):

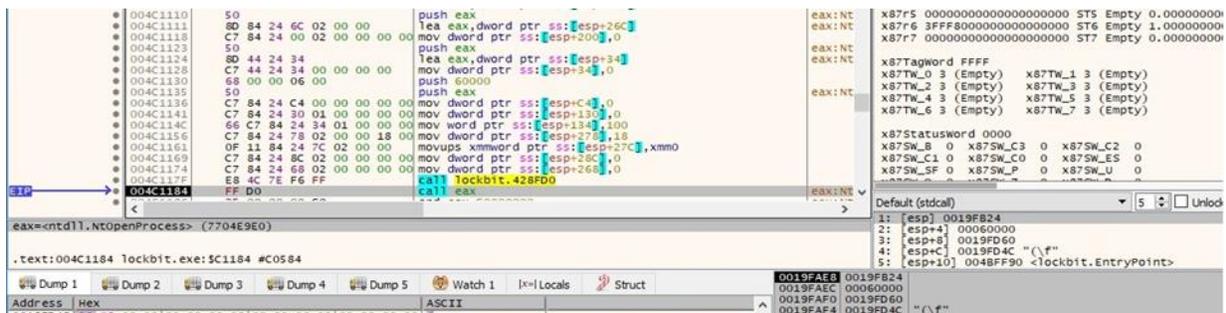


Figure 9

The RtlCreateAcl function is utilized to create and initialize an access control list (0x4 = **ACL_REVISION_DS**):



Figure 14

The RtlAddAccessDeniedAce routine is used to add an access-denied access control entry (ACE) to the ACL created earlier (0x4 = **ACL_REVISION_DS**, 0x1 = **FILE_READ_DATA**):

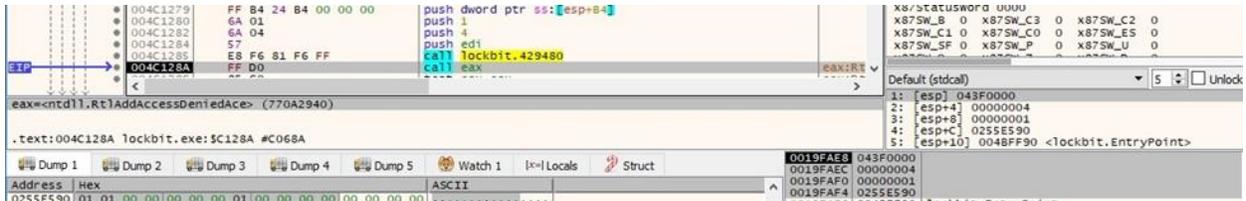


Figure 15

The malicious file obtains a pointer to the first ACE in the ACL via a function call to RtlGetAce:

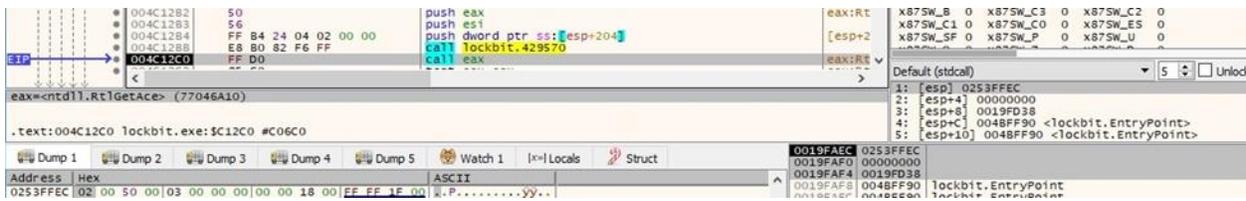


Figure 16

The process adds an ACE to the ACL previously created using RtlAddAce (0x4 = **ACL_REVISION_DS**):

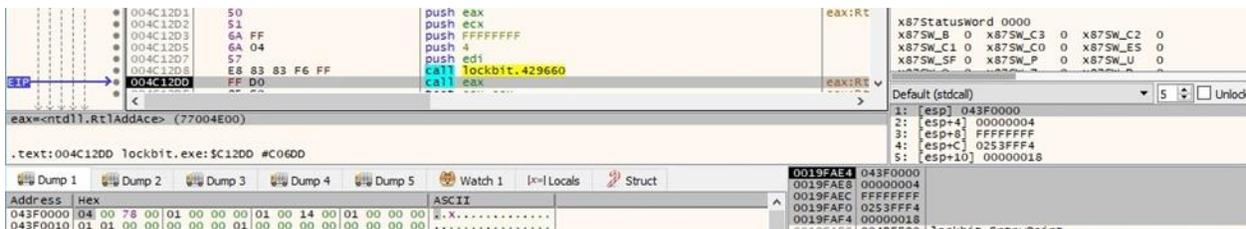


Figure 17

LockBit sets the DACL of the current process to the ACL modified earlier by calling the SetSecurityInfo API (0x6 = **SE_KERNEL_OBJECT**, 0x4 = **DACL_SECURITY_INFORMATION**):

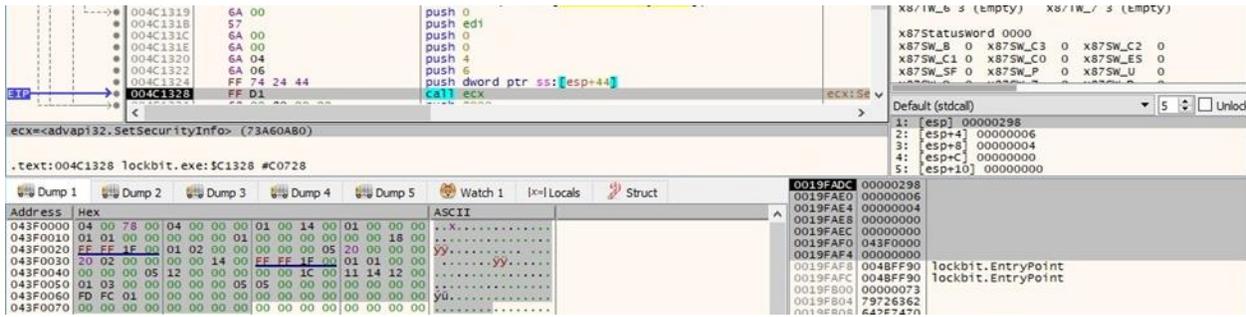


Figure 18

The malware modifies the hard error mode in a way that some error types are not displayed to the user (0xC = **ProcessDefaultHardErrorMode**, 0x7 = **SEM_FAILCRITICALERRORS** | **SEM_NOGPFAULTERRORBOX** | **SEM_NOALIGNMENTFAULTEXCEPT**):

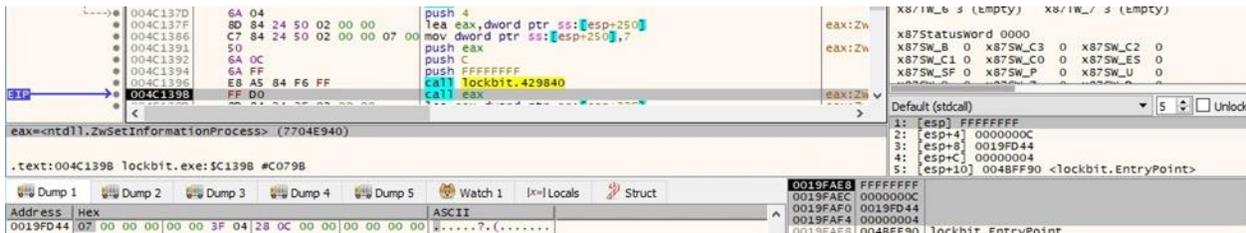


Figure 19

The ransomware enables the SeTakeOwnershipPrivilege privilege in the current process token (0x9 = **SeTakeOwnershipPrivilege**):

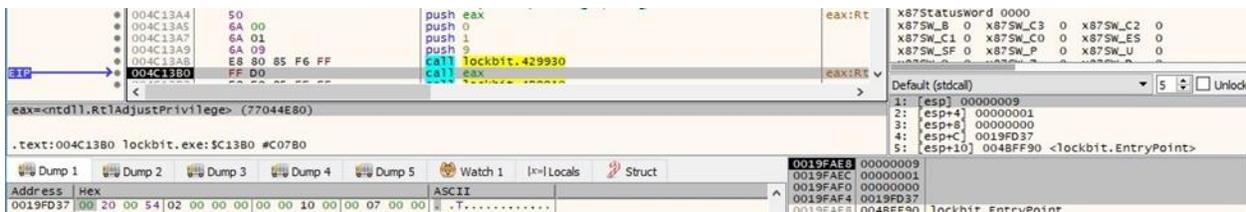


Figure 20

LockBit decrypts a list of processes and services that will be stopped during the infection (the entire list can be found in the appendix):

Address	Hex	ASCII
04460000	77 78 53 65 72 76 65 72 2C 77 78 53 65 72 76 65	WxServer,wxServe
04460010	72 56 69 65 77 2C 73 71 6C 6D 61 6E 67 72 2C 52	rview,sqImangr,R
04460020	41 67 75 69 2C 73 75 70 65 72 76 69 73 65 2C 43	Agui,supervise,C
04460030	75 6C 74 75 72 65 2C 44 65 66 77 61 74 63 68 2C	ulture,Defwatch,
04460040	77 69 6E 77 6F 72 64 2C 51 42 57 33 32 2C 51 42	winword,QBW32,Q
04460050	44 42 4D 67 72 2C 71 62 75 70 64 61 74 65 2C 61	DBMgr,qbupdate,a
04460060	78 6C 62 72 69 64 67 65 2C 68 74 74 70 64 2C 66	xlbridge,httpd,f
04460070	64 6C 61 75 6E 63 68 65 72 2C 4D 73 44 74 53 72	dlauncher,MsDtsR
04460080	76 72 2C 6A 61 76 61 2C 33 36 30 73 65 2C 33 36	vr,java,360se,36
04460090	30 64 6F 63 74 6F 72 2C 77 64 73 77 66 73 61 66	odoctor,wdsfwsaf
044600A0	65 2C 66 64 68 6F 73 74 2C 47 44 73 63 61 6E 2C	e,fdhost,Gdscan,
044600B0	5A 68 75 44 6F 6E 67 46 61 6E 67 59 75 2C 51 42	ZhuDongFangYu,QB
044600C0	44 42 4D 67 72 4E 2C 6D 79 73 71 6C 64 2C 41 75	DBMgrn,mysqlD,Au
044600D0	74 6F 64 65 73 68 44 65 73 68 74 6F 70 41 70 70	todeskDesktopApp
044600E0	2C 61 63 77 65 62 62 72 6F 77 73 65 72 2C 43 72	acwebbrowser,Cr
044600F0	65 61 74 69 76 65 20 43 6C 6F 75 64 2C 41 64 6F	eative Cloud,Ado
04460100	62 65 20 44 65 73 68 74 6F 70 20 53 65 72 76 69	be Desktop Servi
04460110	63 65 2C 43 6F 72 65 53 79 6E 63 2C 41 64 6F 62	ce,CoreSync,Adob
04460120	65 20 43 45 46 2C 48 65 6C 70 65 72 2C 6E 6F 64	e CEF,Helper,nod

Figure 21

Address	Hex	ASCII
04460000	77 72 61 70 70 65 72 2C 44 65 66 57 61 74 63 68	Wrapper,Defwatch
04460010	2C 63 63 45 76 74 40 67 72 2C 63 63 53 65 74 4D	,ccEvtMgr,ccSetM
04460020	67 72 2C 53 61 76 52 6F 61 6D 2C 53 71 6C 73 65	gr,savRoam,sqlse
04460030	72 76 72 2C 73 71 6C 61 67 65 6E 74 2C 73 71 6C	rvr,sqlagent,sql
04460040	61 64 68 6C 70 2C 43 75 6C 73 65 72 76 65 72 2C	adhlp,cu1server,
04460050	52 54 56 73 63 61 6E 2C 73 71 6C 62 72 6F 77 73	RTVscan,sqlbrows
04460060	65 72 2C 53 51 4C 41 44 48 4C 50 2C 51 42 49 44	er,SQLADHLP,Q8BD
04460070	50 53 65 72 76 69 63 65 2C 49 6E 74 75 69 74 2E	PService,Intuit.
04460080	51 75 69 63 68 42 6F 6F 6B 73 2E 46 43 53 2C 51	QuickBooks.FCS,Q
04460090	42 43 46 40 6F 6E 69 74 6F 72 53 65 72 76 69 63	BCFMonitorServic
044600A0	65 2C 20 6D 73 6D 64 73 72 76 2C 74 6F 6D 63 61	e,msmdsrv,tomca
044600B0	74 36 2C 7A 68 75 64 6F 6E 67 66 61 6E 67 79 75	t6,zhudongfangyu
044600C0	2C 76 6D 77 61 72 65 2D 75 73 62 61 72 62 69 74	,vmware-usbarbit
044600D0	61 74 6F 72 36 34 2C 76 6D 77 61 72 65 2D 63 6F	ator64,vmware-co
044600E0	6E 76 65 72 74 65 72 2C 64 62 73 72 76 31 32 2C	nverter,dbsrv12,
044600F0	64 62 65 6E 67 38 2C 4D 53 53 51 4C 24 4D 49 43	dbeng8,MSSQL\$MIC
04460100	52 4F 53 4F 46 54 23 23 57 49 44 2C 4D 53 53 51	ROSOFT##WID,MSSQ
04460110	4C 24 56 45 45 41 4D 53 51 4C 32 30 31 32 2C 53	L\$VEEAMSQL2012,S
04460120	51 4C 41 67 65 6E 74 24 56 45 45 41 4D 53 51 4C	QLAgent\$VEEAMSQL

Figure 22

The malware calls the ZwOpenProcessToken API in order to open the access token associated with the current process (0x8 = **TOKEN_QUERY**):

Figure 23

GetTokenInformation is utilized to extract the user account of the token (0x1 = **TokenUser**):

Figure 24

The AllocateAndInitializeSid routine is used to allocate and initialize a security identifier (SID) with a single subauthority:

Figure 25

The executable compares two security identifier (SID) values using the EqualSid API:

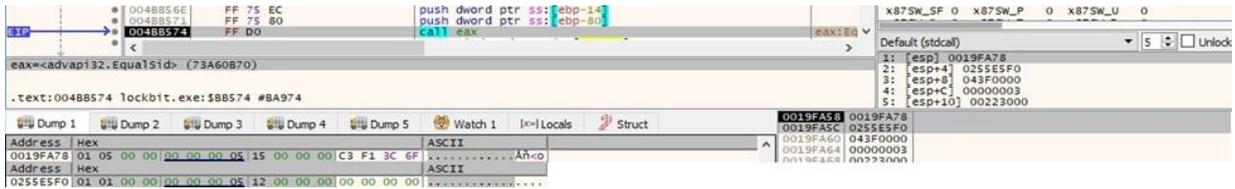


Figure 26

There is a recurrent function call to GlobalMemoryStatusEx that retrieves information about the current usage of both physical and virtual memory:

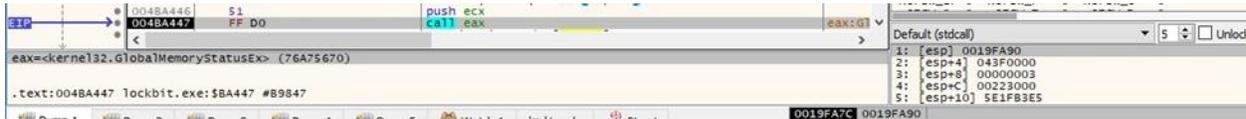


Figure 27

LockBit creates a new thread using the CreateThread API, which will run the sub_4DF310 function:



Figure 28

ZwSetInformationThread is used to hide the thread from our debugger however, the x32dbg's plugin called ScyllaHide can circumvent its effect (0x11 = **HideThreadFromDebugger**):

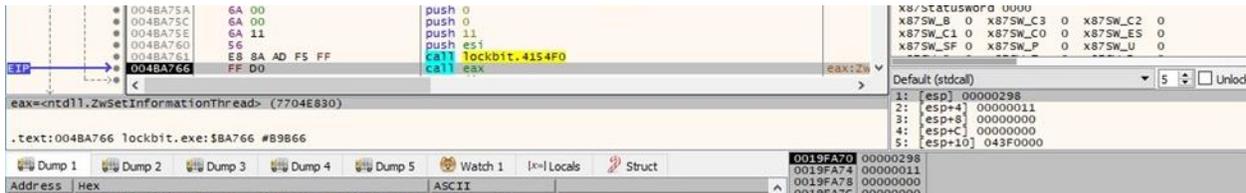


Figure 29

Thread activity – sub_4DF310 function

The shutdown priority for the current process relative to other processes in the system is set to 0, which means that it's set to be the last process to be shut down:

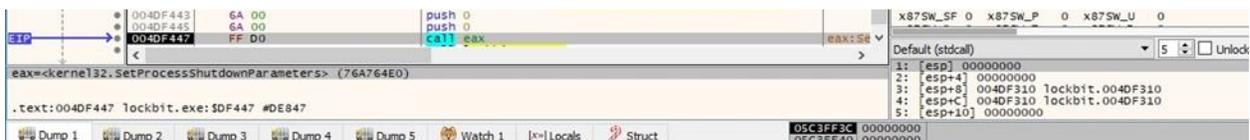


Figure 30

GetSystemDirectoryW is utilized to retrieve the path of the system directory:



Figure 31

The process creates an activation context and activates it using the CreateActCtxW and ActivateActCtx routines:

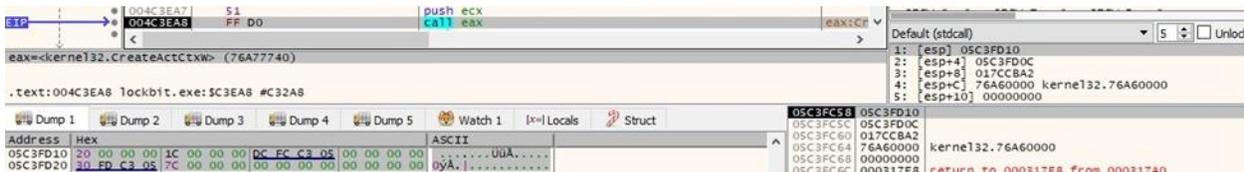


Figure 32



Figure 33

The binary registers and initializes specific common control window classes using the InitCommonControls API:

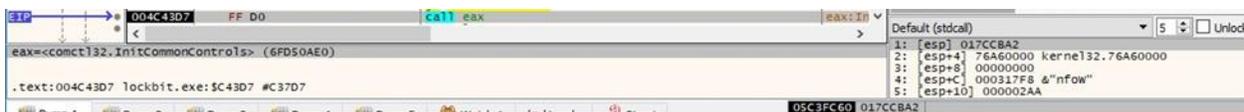


Figure 34

GdiplusStartup is used to initialize Windows GDI+:



Figure 35

The malicious file initializes the COM library on the current thread:



Figure 36

The GetVersion routine is used to retrieve the operating system version:



Figure 37

CreateStreamOnHGlobal is utilized to create a stream object that uses an HGLOBAL memory handle to store the content:

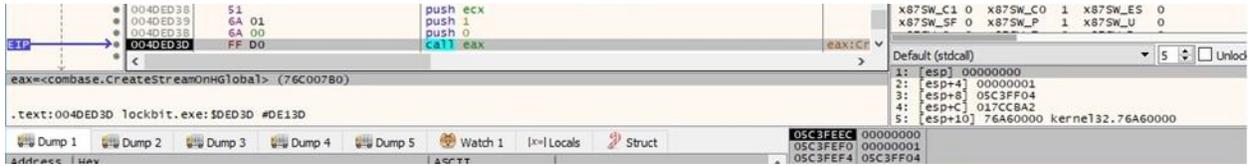


Figure 38

The stream content is modified, and the process uses the GdipCreateBitmapFromStream function to create a Bitmap object based on the stream:

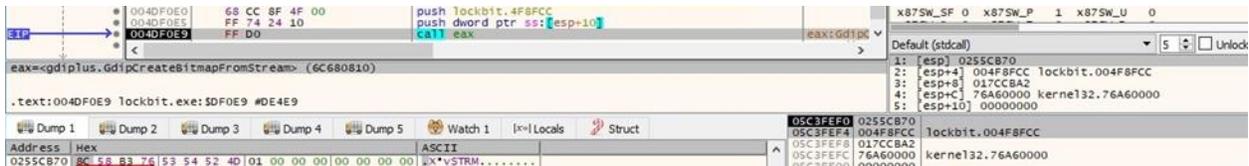


Figure 39

The malware loads the standard arrow cursor resource via a function call to LoadCursorW (0x7F00 = **IDC_ARROW**):

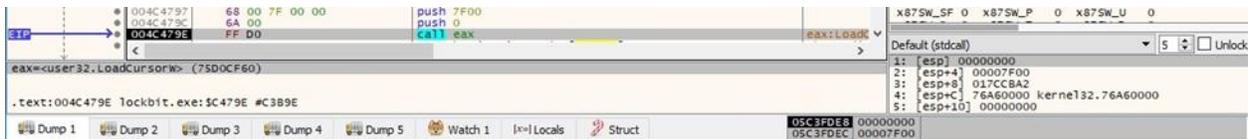


Figure 40

GdipAlloc is utilized to allocate memory for a Windows GDI+ object:

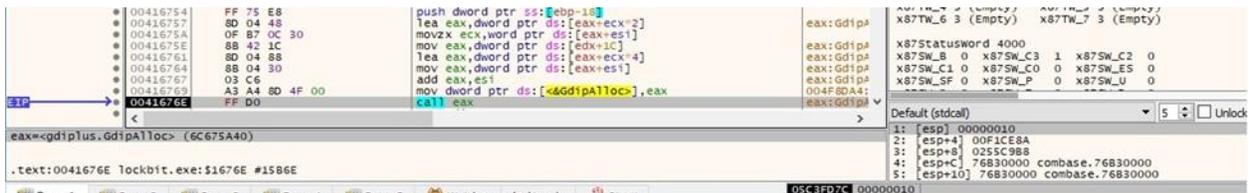


Figure 41

There is another call to GdipCreateBitmapFromStream followed by a call to GdipDisposeImage, which releases resources used by the Image object:



Figure 42

LockBit defines a Shift+F1 hot key for the new window that can be used to unhide it (0x70 = **VK_F1**, 0x4 = **MOD_SHIFT**):



Figure 48



Figure 49

GetMessageW is used to retrieve a message from the thread's message queue:



Figure 50

The malicious file translates virtual-key messages into character messages via a call to TranslateMessage:



Figure 51

DispatchMessageW is utilized to dispatch a message retrieved by the GetMessage function:



Figure 52

Thread activity – sub_4C3430 function

The process sends the **LVM_GETITEMCOUNT** message to the newly created window (0x1004 = **LVM_GETITEMCOUNT**):



Figure 53

The malware calls the InvalidateRect API many times to add multiple rectangles to the window's update region:



Figure 54

We continue with the analysis of the main thread.

The CommandLineToArgvW routine obtains an array of pointers to the command line arguments:



Figure 55

The file tries to see if the access token is elevated by calling the NtQueryInformationToken API (0x14 = **TokenElevation**):



Figure 56

Depending on the result, the malware proceeds by decrypting the "[+] Process created with admin rights" or "[-] Process created with limited rights" strings. We know that this sample is supposed to perform UAC bypass in the case of low-level privileges however, this method wasn't employed on our Windows 10 analysis machine (it's supposed to be used on older Windows versions).

The process sends the "[+] Process created with admin rights" message to the hidden window by calling the SendMessageA API:



Figure 57

The binary creates a mutex called "\\BaseNamedObjects\\{3FE573D4-3FE5-DD38-399C-886767BD8875}" to ensure that only one instance of the malware is running at one time (0x1F0001 = **MUTEX_ALL_ACCESS**):

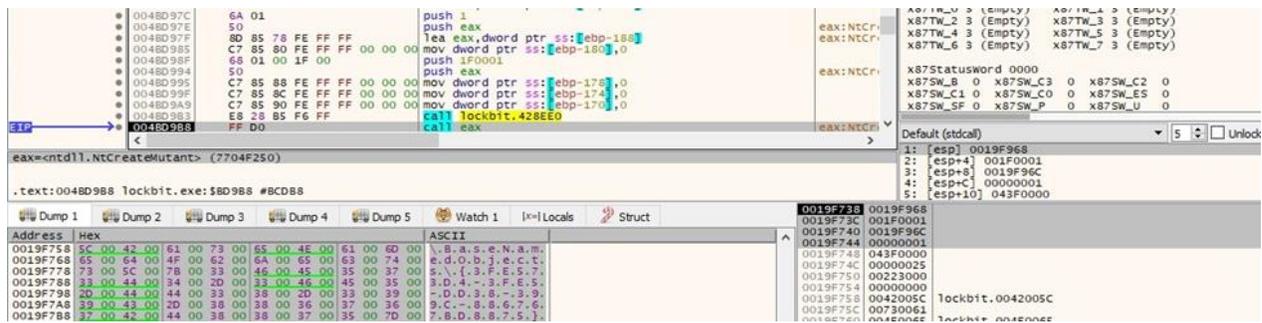


Figure 58

The NetBIOS name of the local computer is extracted using GetComputerNameW:



Figure 59

The malicious executable retrieves the name of the primary domain controller by calling the NetGetDCName function. LockBit has the ability to propagate on the network and kill processes and services via malicious GPOs (group policy objects); however, these features weren't activated in this sample:



Figure 60

The process opens the Run registry key using RegCreateKeyExA (0x80000001 = **HKEY_CURRENT_USER**, 0x2001F = **KEY_READ | KEY_WRITE**):

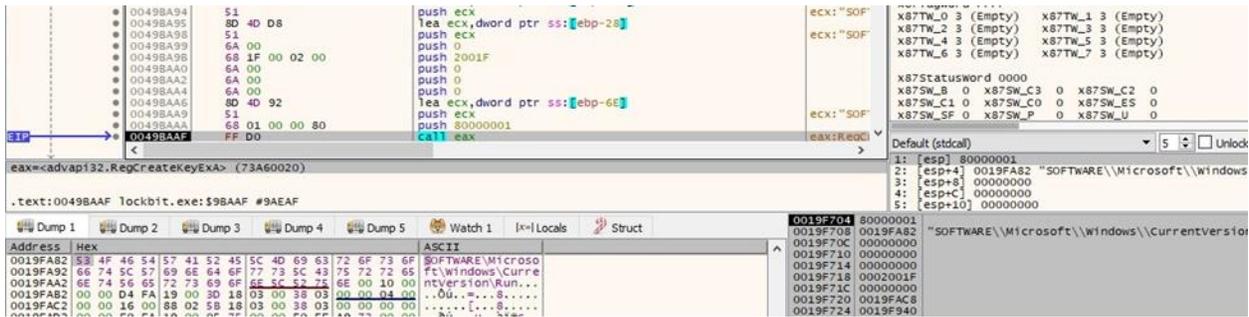


Figure 61

The file is looking for a registry value called "{9FD872D4-E5E5-DDC5-399C-396785BDC975}":

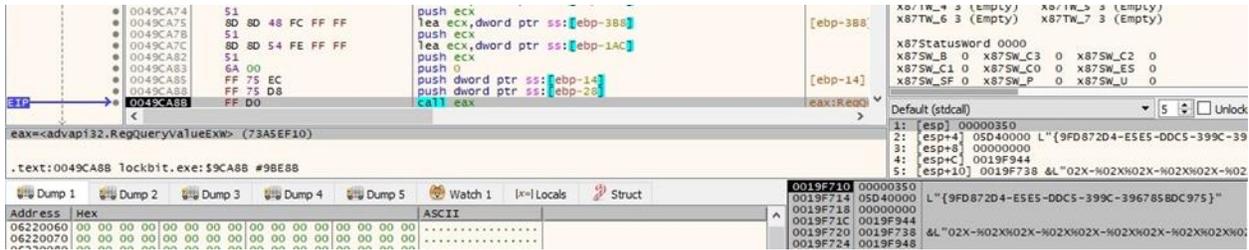


Figure 62

The malware establishes persistence by creating the above registry value:

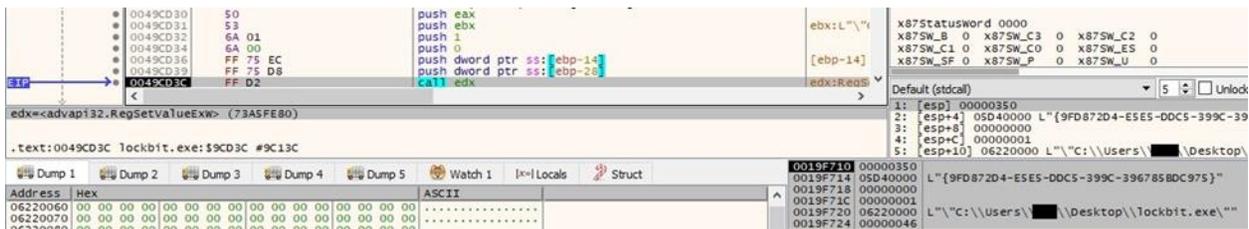


Figure 63



Figure 64

CreateThread is used to create a new thread within the address space of the process:

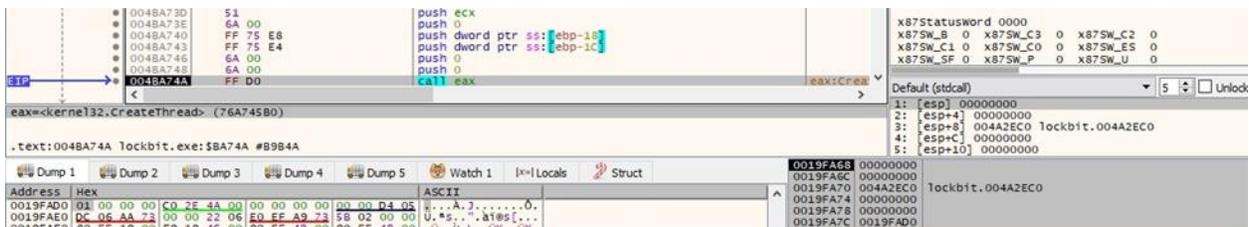


Figure 65

As in the case of every thread creation, the binary tries to hide it from the debugger using the ZwSetInformationThread API.

A file called "C:\windows\system32\2ED873.ico" is created via a function call to ZwCreateFile (0x40000000 = **GENERIC_WRITE**, 0x80 = **FILE_ATTRIBUTE_NORMAL**, 0x5 = **FILE_OVERWRITE_IF**):

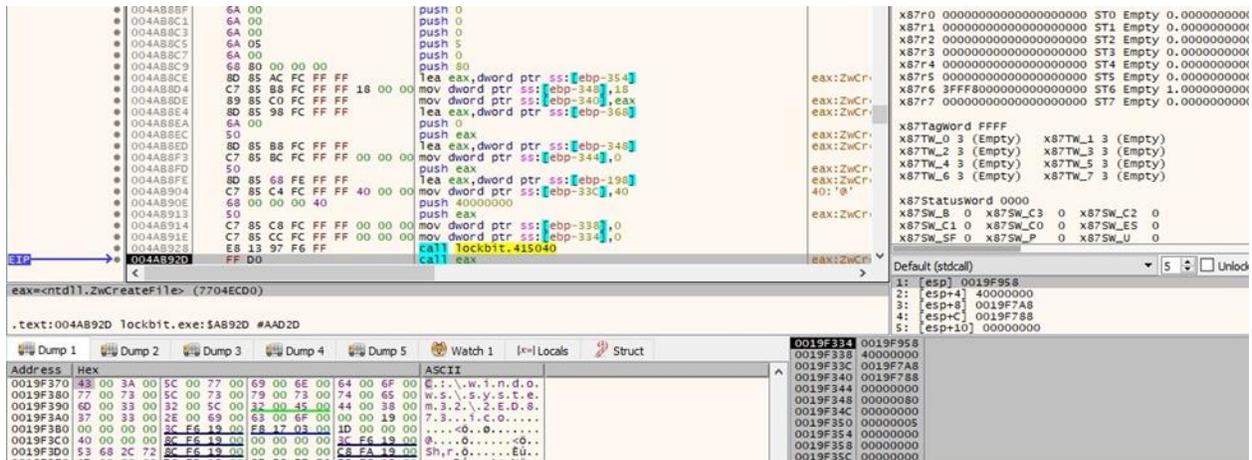


Figure 66

The ICO file is populated using the ZwWriteFile routine:

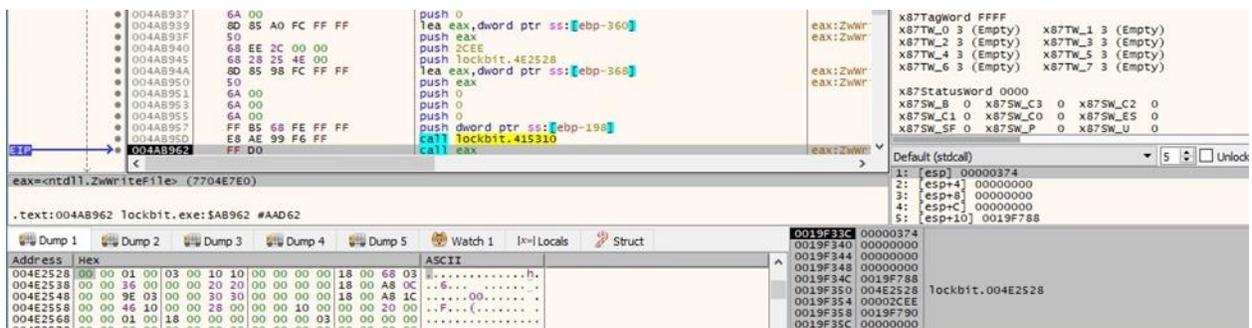


Figure 67

The executable creates the "HKCR\lockbit" registry key using ZwCreateKey (0x2000000 = **MAXIMUM_ALLOWED**):

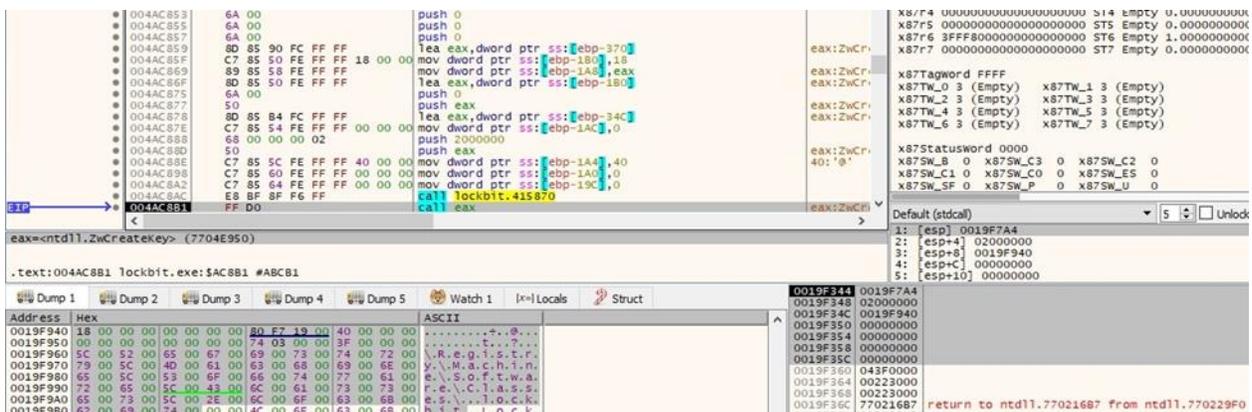


Figure 68

LockBit creates the DefaultIcon subkey and sets its value to the newly created ICO file, as highlighted below:

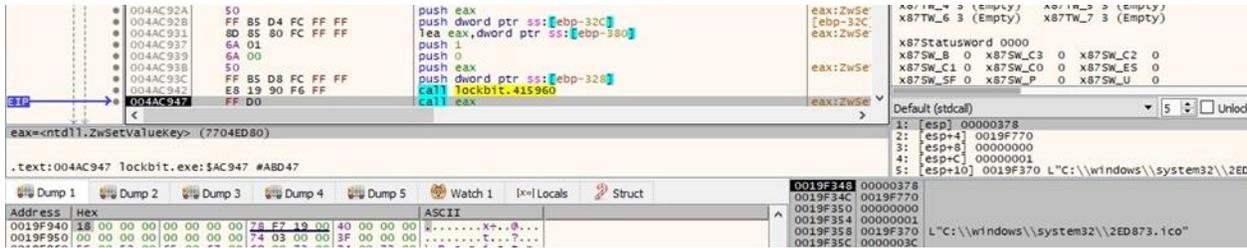


Figure 69



Figure 70

Thread activity – sub_4A2EC0 function

The FindFirstVolumeW API is utilized to begin scanning the volumes of the computer:

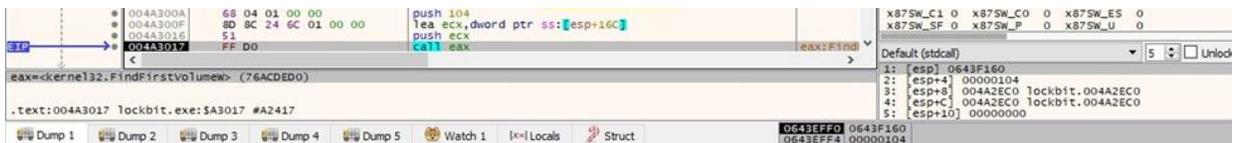


Figure 71

QueryDosDeviceW is used to obtain the current mapping for the above volume:

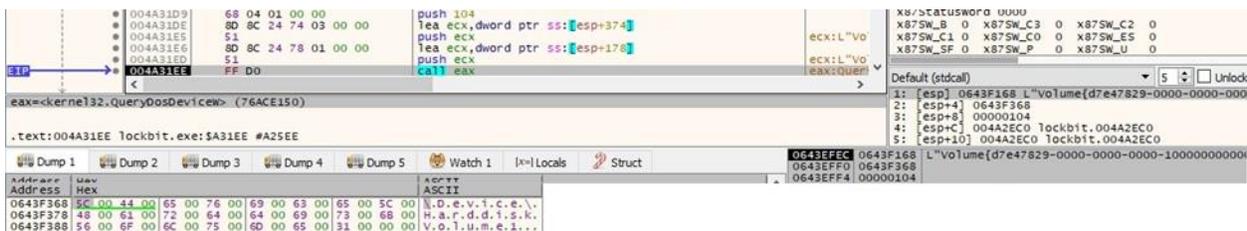


Figure 72

The malware retrieves a list of drive letters for the specified volume via a call to GetVolumePathNamesForVolumeNameW:

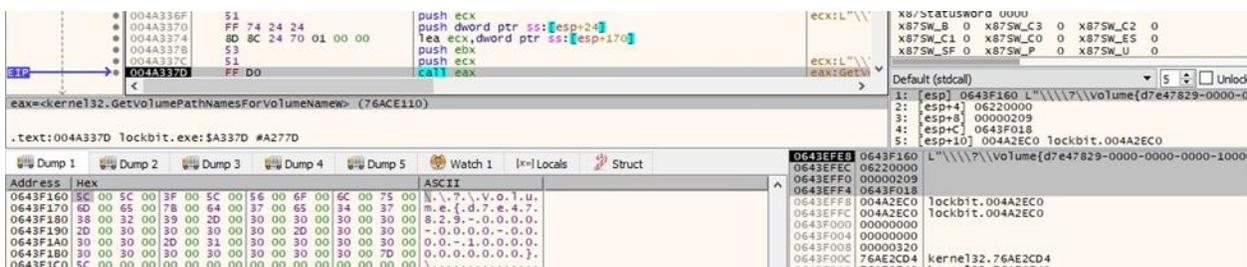


Figure 73

The drive type of the volume is extracted using GetDriveTypeW:



Figure 74

The malicious process sends a message regarding the identified volume to the LockBit hidden window, as displayed in figure 75.

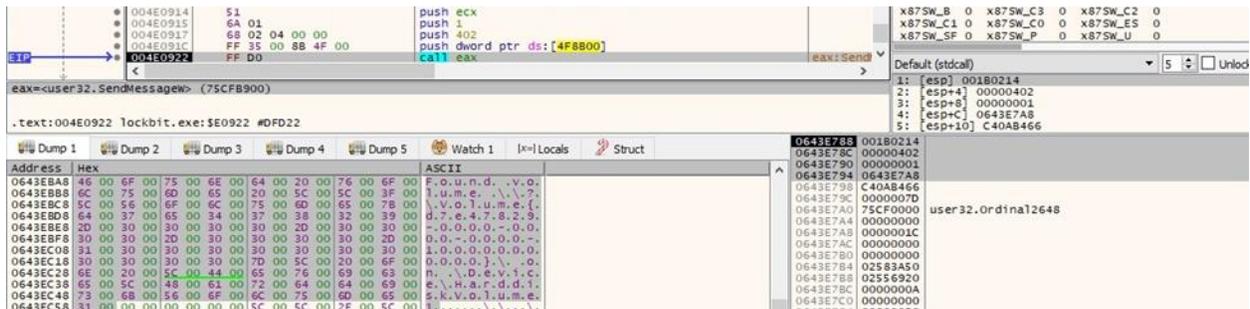


Figure 75

The malicious file continues the volume search via a function call to FindNextVolumeW:

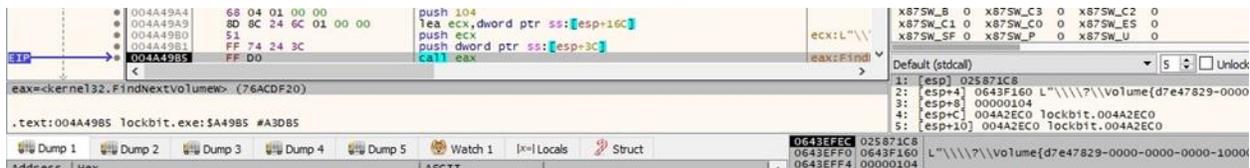


Figure 76

The purpose of the malware is to find unmounted volumes and mount them.

LockBit tries to open the BOOTMGR file from the volume (0x80000000 = **GENERIC_READ**, 0x3 = **FILE_SHARE_READ** | **FILE_SHARE_WRITE**, 0x3 = **OPEN_EXISTING**, 0x80 = **FILE_ATTRIBUTE_NORMAL**):

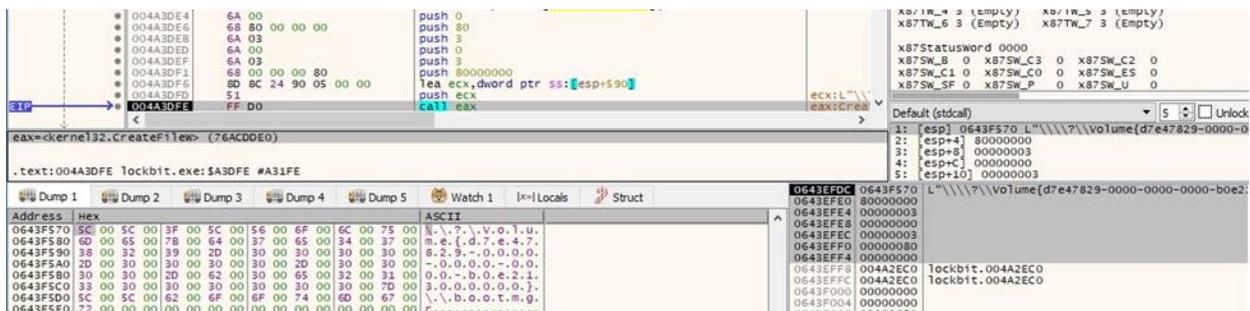


Figure 77

An unmounted volume is mounted by calling the SetVolumeMountPointW routine:



Figure 78



Figure 79

LockBit sends a message regarding the successful mount operation to the hidden window (see figure 80). After the enumeration is complete, the thread exits by calling the `RtlExitUserThread` function.

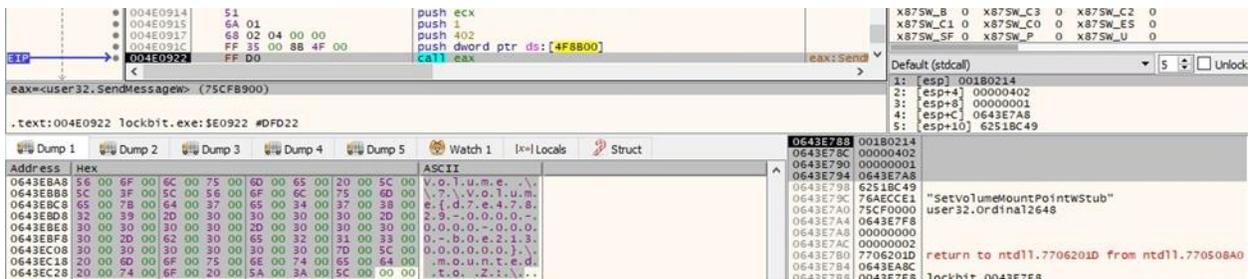


Figure 80

The binary calls the `SHChangeNotify` API with the `SHCNE_ASSOCCHANGED` parameter (`0x80000000 = SHCNE_ASSOCCHANGED`):

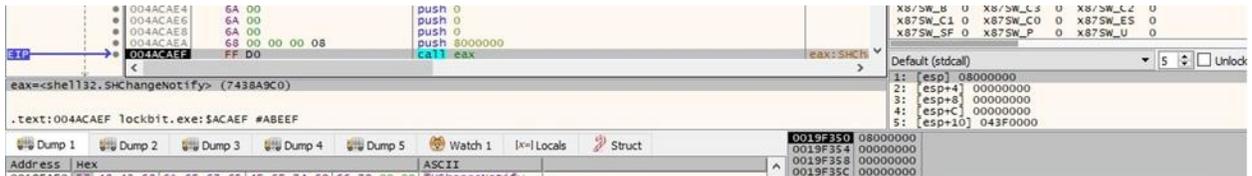


Figure 81

A new thread is created by the malware using `CreateThread`:

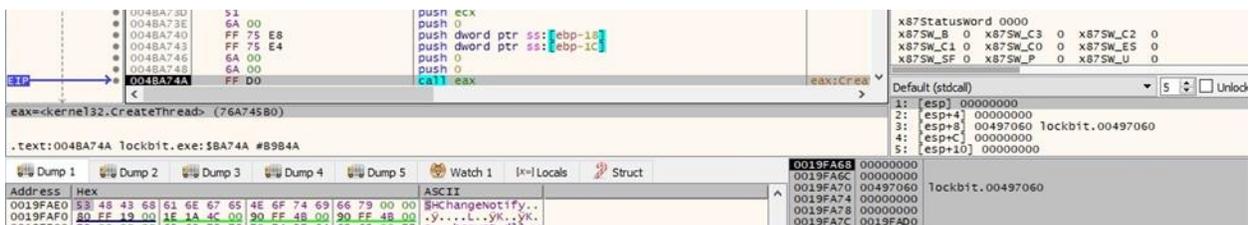


Figure 82

Intel and AMD CPUs implement a functionality called "AES-NI" (Advanced Encryption Standard New Instructions), which can be used for high-speed AES encryption processing. The binary uses the `cpuid` instruction in order to retrieve the CPU type of the machine and the vendor of the CPU:

```

0040F65D 31 C0 xor eax,eax
EIP 0040F65F 0F A2
0040F661 81 F9 6E 74 65 6C cmp ecx,edi plus_6C65746E
0040F667 74 2A je lockbit.40F693
0040F669 81 F9 63 41 4D 44 cmp ecx,444D4163
0040F66F 74 27 je lockbit.40F698
0040F671 81 FB 43 65 6E 74 cmp ebx,shell32.746E6543
0040F677 74 24 je lockbit.40F69D
0040F679 81 F8 56 49 41 20 cmp ebx,20414956
0040F67F 74 1C je lockbit.40F69D
0040F681 81 FB 43 79 72 69 cmp ebx,69727943
0040F687 74 19 je lockbit.40F6A2
0040F689 81 FB 4E 65 78 47 cmp ebx,4778654E
0040F68F 74 16 je lockbit.40F6A7
0040F691 EB 17 jmp lockbit.40F6AA
0040F693 83 CE 01 or esi,1
0040F696 EB 12 jmp lockbit.40F6AA
0040F698 83 CE 02 or esi,2
0040F69B EB 0D jmp lockbit.40F6AA
0040F69D 83 CE 03 or esi,3
0040F6A0 EB 08 jmp lockbit.40F6AA
0040F6A2 83 CE 04 or esi,4
0040F6A5 EB 03 jmp lockbit.40F6AA
0040F6A7 83 CE 05 or esi,5
0040F6A9 85 C0 test eax,eax
0040F6AC 74 55 je lockbit.40F703

```

Figure 83

Whether the CPU supports "AES-NI" the process sends the "[+] AES-NI enabled" message to the hidden window using SendMessageA.

The malicious process generates 16 random bytes by calling the BCryptGenRandom routine (0x2 = **BCRYPT_USE_SYSTEM_PREFERRED_RNG**):

```

0048E1AA 6A 02 push 2
0048E1AC FF 75 0C push dword ptr ss:[ebp+C]
0048E1AF FF 75 08 push dword ptr ss:[ebp+8]
EIP 0048E1B4 FF D0 call eax

```

eax=bcrypt.BCryptGenRandom (73817DE0)

.text:0048E1B4 lockbit.exe:\$8E1B4 #8D584

Address	Hex	ASCII
004F8828	F8 F0 85 22 4E 80 5D 21 FF 95 2B E5 CE 53 2D 15	00, "N", "l", "y", "t", "a", "S", "

Figure 84

The ransom note is also stored in an encrypted form as a stack string that will be decrypted using a custom algorithm:

```

EIP 0049EF54 C6 85 D3 FC FF FF 57 mov byte ptr ss:[ebp-32D],57
0049EF58 C6 85 D4 FC FF FF 7A mov byte ptr ss:[ebp-32C],7A
0049EF5C C6 85 D5 FC FF FF 6E mov byte ptr ss:[ebp-32B],6E
0049EF63 C6 85 D6 FC FF FF 76 mov byte ptr ss:[ebp-32A],76
0049EF70 C6 85 D7 FC FF FF 4D mov byte ptr ss:[ebp-329],4D
0049EF77 C6 85 D8 FC FF FF 74 mov byte ptr ss:[ebp-328],74
0049EF7E C6 85 D9 FC FF FF 7F mov byte ptr ss:[ebp-327],7F
0049EF85 C6 85 DA FC FF FF 2B mov byte ptr ss:[ebp-326],2B
0049EF8C C6 85 DB FC FF FF 3D mov byte ptr ss:[ebp-325],3D
0049EF93 C6 85 DC FC FF FF 39 mov byte ptr ss:[ebp-324],39
0049EF9A C6 85 DD FC FF FF 3B mov byte ptr ss:[ebp-323],3B
0049EFA1 C6 85 DE FC FF FF 2B mov byte ptr ss:[ebp-322],2B
0049EFA8 C6 85 DF FC FF FF 5D mov byte ptr ss:[ebp-321],5D
0049EFAF C6 85 E0 FC FF FF 6C mov byte ptr ss:[ebp-320],6C
0049EFB6 C6 85 E1 FC FF FF 79 mov byte ptr ss:[ebp-31F],79
0049EFBD C6 85 E2 FC FF FF 7E mov byte ptr ss:[ebp-31E],7E
0049EFC4 C6 85 E3 FC FF FF 7A mov byte ptr ss:[ebp-31D],7A
0049EFCB C6 85 E4 FC FF FF 78 mov byte ptr ss:[ebp-31C],78
0049EFD2 C6 85 E5 FC FF FF 82 mov byte ptr ss:[ebp-31B],82
0049EFD9 C6 85 E6 FC FF FF 6C mov byte ptr ss:[ebp-31A],6C
0049EFE0 C6 85 E7 FC FF FF 7D mov byte ptr ss:[ebp-319],7D
0049EFE7 C6 85 E8 FC FF FF 70 mov byte ptr ss:[ebp-318],70
0049EFEE C6 85 E9 FC FF FF 18 mov byte ptr ss:[ebp-317],18
0049EFF5 C6 85 EA FC FF FF 15 mov byte ptr ss:[ebp-316],15
0049EFFC C6 85 EB FC FF FF 18 mov byte ptr ss:[ebp-315],18
0049F003 C6 85 EC FC FF FF 15 mov byte ptr ss:[ebp-314],15
0049F00A C6 85 ED FC FF FF 64 mov byte ptr ss:[ebp-313],64
0049F011 C6 85 EE FC FF FF 7A mov byte ptr ss:[ebp-312],7A
0049F018 C6 85 EF FC FF FF 80 mov byte ptr ss:[ebp-311],80
0049F01F C6 85 F0 FC FF FF 7D mov byte ptr ss:[ebp-310],7D
0049F026 C6 85 F1 FC FF FF 2B mov byte ptr ss:[ebp-30F],2B
0049F02D C6 85 F2 FC FF FF 6F mov byte ptr ss:[ebp-30E],6F
0049F034 C6 85 F3 FC FF FF 6C mov byte ptr ss:[ebp-30D],6C
0049F03B C6 85 F4 FC FF FF 7F mov byte ptr ss:[ebp-30C],7F
0049F042 C6 85 F5 FC FF FF 6C mov byte ptr ss:[ebp-30B],6C
0049F049 C6 85 F6 FC FF FF 2B mov byte ptr ss:[ebp-30A],2B
0049F050 C6 85 F7 FC FF FF 6C mov byte ptr ss:[ebp-309],6C
0049F057 C6 85 F8 FC FF FF 7D mov byte ptr ss:[ebp-308],7D
0049F05E C6 85 F9 FC FF FF 70 mov byte ptr ss:[ebp-307],70
0049F065 C6 85 FA FC FF FF 2B mov byte ptr ss:[ebp-306],2B
0049F06C C6 85 FB FC FF FF 7E mov byte ptr ss:[ebp-305],7E
0049F073 C6 85 FC FF FF 7F mov byte ptr ss:[ebp-304],7F
0049F07A C6 85 FD FC FF FF 7A mov byte ptr ss:[ebp-303],7A
0049F081 C6 85 FE FC FF FF 77 mov byte ptr ss:[ebp-302],77
0049F088 C6 85 FF FC FF FF 7D mov byte ptr ss:[ebp-301],7D
0049F08F C6 85 00 FD FF FF 79 mov byte ptr ss:[ebp-300],79
0049F096 C6 85 01 FD FF FF 2B mov byte ptr ss:[ebp-2FF],2B
0049F09D C6 85 02 FD FF FF 6C mov byte ptr ss:[ebp-2FE],6C
0049F0A4 C6 85 03 FD FF FF 79 mov byte ptr ss:[ebp-2FD],79
0049F0AB C6 85 04 FD FF FF 6F mov byte ptr ss:[ebp-2FC],6F

```

Figure 85

Address	Hex	ASCII
0019F7C3	4C 6F 63 68 42 69 74 20 32 2E 30 20 52 61 6E 73	LockBit 2.0 Rans
0019F7D3	6F 6D 77 61 72 65 0D 0A 0D 0A 59 6F 75 72 20 64	omware....Your d
0019F7E3	61 74 61 20 61 72 65 20 73 74 6F 6C 65 6E 20 61	ata are stolen a
0019F7F3	6E 64 20 65 6E 63 72 79 70 74 65 64 0D 0A 54 68	nd encrypted..Th
0019F803	65 20 64 61 74 61 20 77 69 6C 6C 20 62 65 20 70	e data will be p
0019F813	75 62 6C 69 73 68 65 64 20 6F 6E 20 54 4F 52 20	ublished on TOR
0019F823	77 65 62 73 69 74 65 20 68 74 74 70 3A 2F 2F 6C	website http://1
0019F833	6F 63 68 62 69 74 61 70 74 36 76 78 35 37 74 33	ockbitapt6vx57t3
0019F843	65 65 71 6A 6F 66 77 67 63 67 6C 6D 75 74 72 33	eeqjofwgcglmtr3
0019F853	61 33 35 6E 79 67 76 6F 68 6A 61 35 75 75 63 63	a35nygvokj5uucc
0019F863	69 70 34 79 68 79 64 2E 6F 6E 69 6F 6E 20 61 6E	ip4ykyd.onion an
0019F873	64 20 68 74 74 70 73 3A 2F 2F 62 69 67 62 6C 6F	d https://bigblo
0019F883	67 2E 61 74 20 69 66 20 79 6F 75 20 64 6F 20 6E	g.at if you do n
0019F893	6F 74 20 70 61 79 20 74 68 65 20 72 61 6E 73 6F	ot pay the ranso
0019F8A3	6D 0D 0A 59 6F 75 20 63 61 6E 20 63 6F 6E 74 61	m..You can conta
0019F8B3	63 74 20 75 73 20 61 6E 64 20 64 65 63 72 79 70	ct us and decryp
0019F8C3	74 20 6F 6E 65 20 66 69 6C 65 20 66 6F 72 20 66	t one file for f
0019F8D3	72 65 65 20 6E 6F 20 74 68 65 73 65 20 54 4F 52	ree on these TOR
0019F8E3	20 73 69 74 65 73 0D 0A 68 74 74 70 3A 2F 2F 6C	sites..http://1
0019F8F3	6F 63 68 62 69 74 73 75 70 34 79 65 7A 63 64 35	ockbitsup4yezcd5

Figure 86

The process creates a registry key called "HKCU\SOFTWARE\2ED873D4E5389C" (0x80000001 = HKEY_CURRENT_USER, 0xF003F = KEY_ALL_ACCESS):

Figure 87

LockBit is looking for two registry values called "Private" and "Public" under the registry key above, which don't exist at this time:

Figure 88

Figure 89

The malware sends the "[+] Generate session keys" message to the hidden window. It will compute a public ECC (Curve25519) key and a private ECC (Curve25519) key.

The file generates 32 random bytes via a function call to BcryptGenRandom:

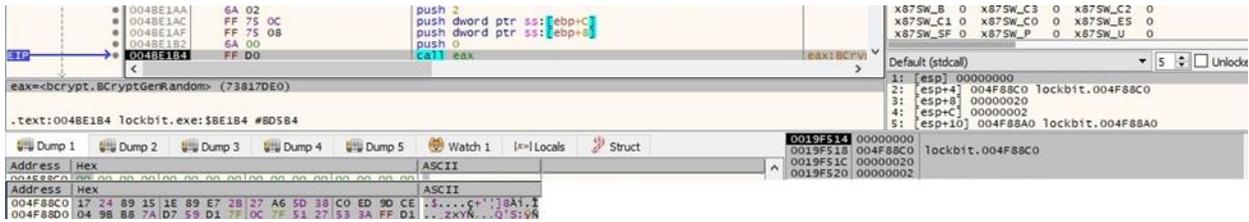


Figure 90

The malicious process implements a Curve25519 wrapper in the sub_4300C0 function. Based on the above buffer, it generates a session ECC public key:

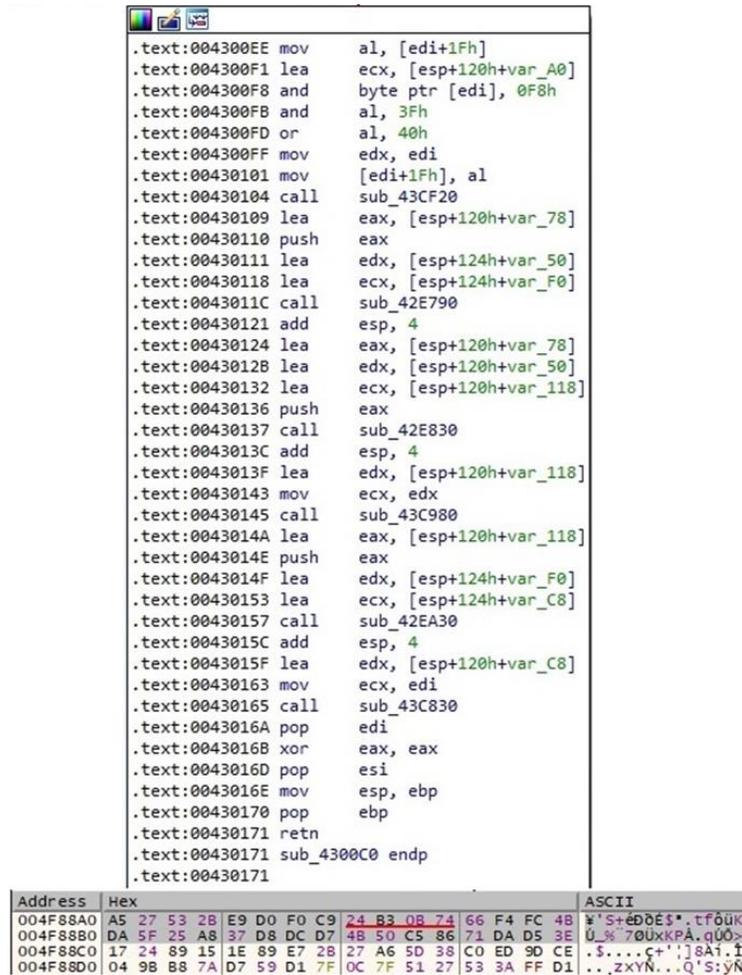


Figure 91

The above operation of generating random bytes is repeated one more time:



Figure 92

The same Curve25519 wrapper is used again to transform the above buffer:

Address	Hex	ASCII
0019F530	73 9C 00 80 3C E1 E2 91 A9 AF EF DA 53 76 8D 11	S...<aa.® iUSV..
0019F540	0B 35 23 13 26 0A 51 17 43 1F 50 DC CA 42 12 7D	.5#.&.Q.C.PUEB.}

Figure 93

The executable embedded an ECC public key that we call Master ECC public key (highlighted in figure 94). Based on the implementation of the Curve25519 algorithm, it is used to generate a shared secret (32-byte value):

Figure 94

The Master ECC public key is utilized to encrypt the session ECC private key computed above:

Figure 95

We have utilized the capa tool in order to confirm that the above function is used to encrypt data using Curve25519:

```

encrypt data using Curve25519 (2 matches)
namespace data-manipulation/encryption/elliptic-curve
author dimitre.andonov@mandiant.com
scope basic block
att&ck Defense Evasion::Obfuscated Files or Information [T1027]
examples 0a0882b8da225406cc838991b5f67d11:0x4135f6, 0a0882b8da225406cc838991b5f67d11:0x416f51,
basic block @ 0x42F89E in function 0x42F6E0
and:
and:
  number: 0xF8 @ 0x42F8AD
  mnemonic: and @ 0x42F8AB, 0x42F8AD
and:
  number: 0x3F @ 0x42F8AB
  mnemonic: and @ 0x42F8AB, 0x42F8AD
and:
  number: 0x40 @ 0x42F8B0
  mnemonic: or @ 0x42F8B0
basic block @ 0x4300EE in function 0x4300C0
and:
and:
  number: 0xF8 @ 0x4300F8
  mnemonic: and @ 0x4300F8, 0x4300FB
and:
  number: 0x3F @ 0x4300FB
  mnemonic: and @ 0x4300F8, 0x4300FB
and:
  number: 0x40 @ 0x4300FD
  mnemonic: or @ 0x4300FD

```

Figure 96

LockBit stores the encrypted session ECC private key in the “HKCU\Software\2ED873D4E5389C\Private” registry value:

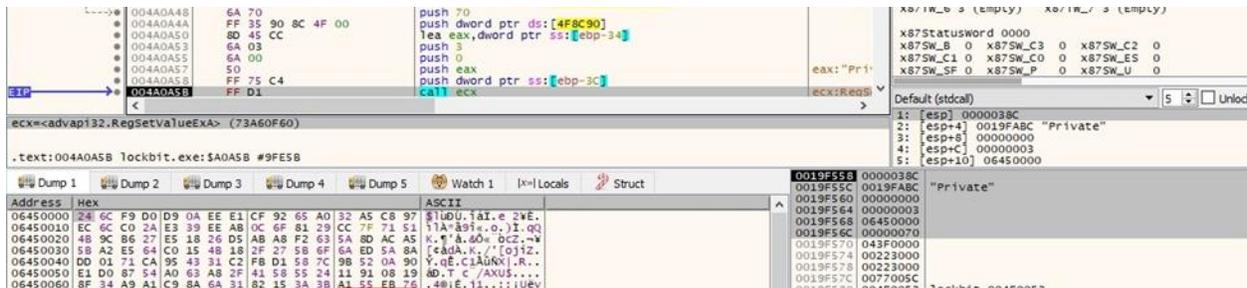


Figure 97

LockBit stores the session ECC public key in the “HKCU\Software\2ED873D4E5389C\Public” registry value:

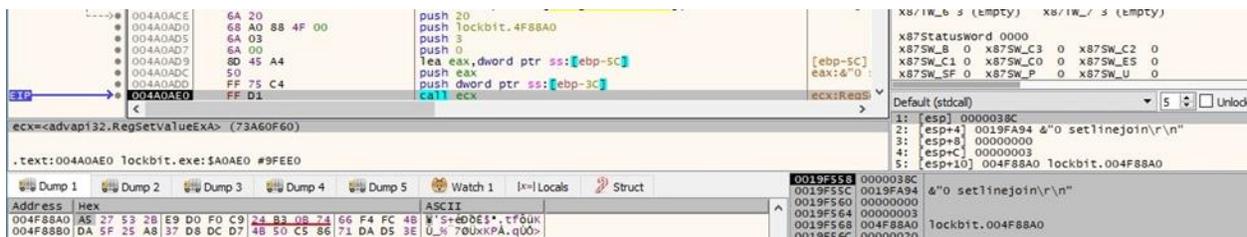


Figure 98

Figure 99 reveals both registry values with their content:

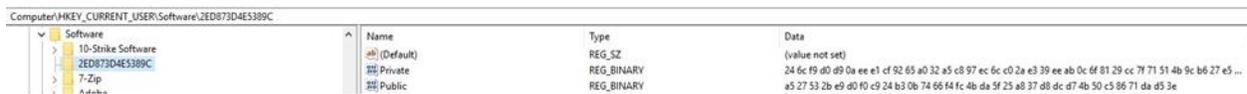


Figure 99

The malware uses I/O completion ports to improve the encryption speed. It creates an I/O completion object by calling the NtCreateIoCompletion API (0x1F0003 = IO_COMPLETION_ALL_ACCESS):

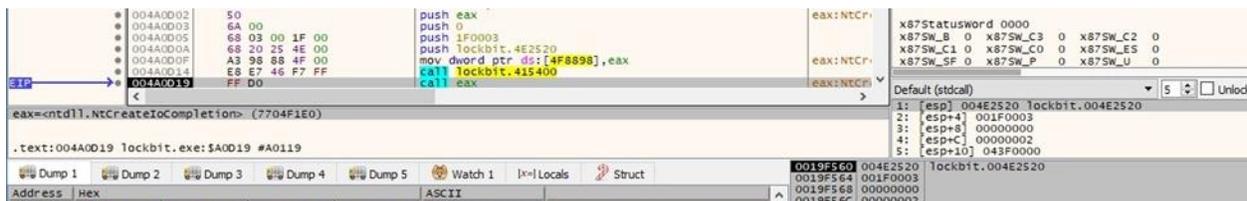


Figure 100

The binary creates 2 (# of processors/cores) that will handle the files encryption:

```

004BA73D 51 00      push ecx
004BA73E 6A 00      push 0
004BA740 FF 75 E8   push dword ptr ss:[ebp-18]
004BA743 FF 75 E4   push dword ptr ss:[ebp-1C]
004BA746 6A 00      push 0
004BA748 6A 00      push 0
004BA74A FF D0      call eax

```

eax=Crea

Default (stdcall)

1:	[esp]	00000000
2:	[esp+4]	00000000
3:	[esp+8]	0049E730 lockbit.0049E730
4:	[esp+C]	00000000
5:	[esp+10]	00000000

Address Hex

0019F548	02 00 00 00 30 E7 49 00	00 00 00 00 A6 00 00 00	ASCII
0019F558	FC F1 04 77 1B 00 4A 00	20 25 4E 00 03 00 1F 00F...%

Figure 101

The thread affinity mask is set to 1 via a function call to ZwSetInformationThread (0x4 = **ThreadAffinityMask**):

```

004A0087 6A 04      push 4
004A0089 50 00      push eax
004A008A 6A 04      push 4
004A008C 52 00      push edx
004A0090 E8 5E 47 F7 FF   call lockbit.4154F0
004A0092 FF D0      call eax

```

eax=ZwSe

Default (stdcall)

1:	[esp]	00000394
2:	[esp+4]	00000004
3:	[esp+8]	0019F79C
4:	[esp+C]	00000004
5:	[esp+10]	043F0000

Address Hex

0019F79C	01 00 00 00 01 00 00 00	00 00 4E 06 2E 00 25 00F...%
----------	-------------------------	-------------------------	------------

Figure 102

GetLogicalDrives is used to retrieve the available disk drives:

```

0045BE8E FF D0      call eax

```

eax=GetL

Default (stdcall)

1:	[esp]	043F0000
2:	[esp+4]	00223000
3:	[esp+8]	00223000
4:	[esp+C]	00223000
5:	[esp+10]	00223000

Address Hex

0019FA78	04 3F 00 00		
----------	-------------	--	--

Figure 103

The malicious binary determines the disk drive type using the GetDriveTypeW routine:

```

0045BFF5 51 00      push ecx
0045BFF6 FF D0      call eax

```

ecx:L"Z:"

eax:GetD

Default (stdcall)

1:	[esp]	0019FAC4 L"Z:"
2:	[esp+4]	043F0000
3:	[esp+8]	00223000
4:	[esp+C]	00223000
5:	[esp+10]	00223000

Address Hex

0019FA74	0019FAC4	L"Z:"	
----------	----------	-------	--

Figure 104

The process is looking for type 2 (**DRIVE_REMOVABLE**), type 3 (**DRIVE_FIXED**) and type 6 (**DRIVE_RAMDISK**) drives:

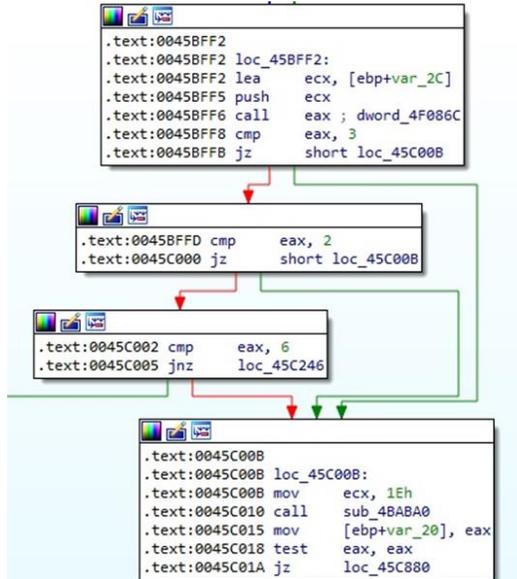


Figure 105

For each targeted drive, the malware creates a new thread that will traverse it and locate all files selected for encryption:

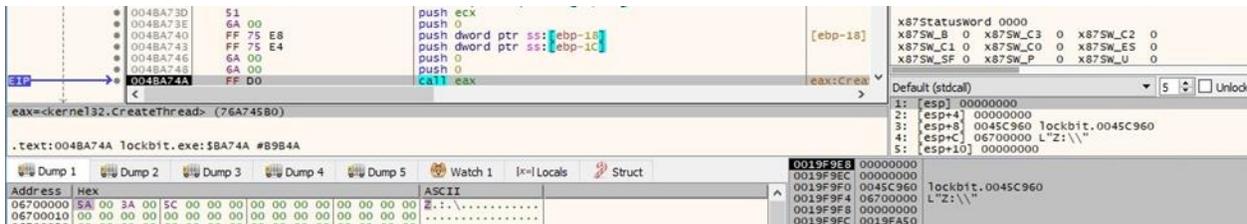


Figure 106

Thread activity – sub_45C960 function

The file compares the drive name with the tsclient (Terminal Server Client) share:

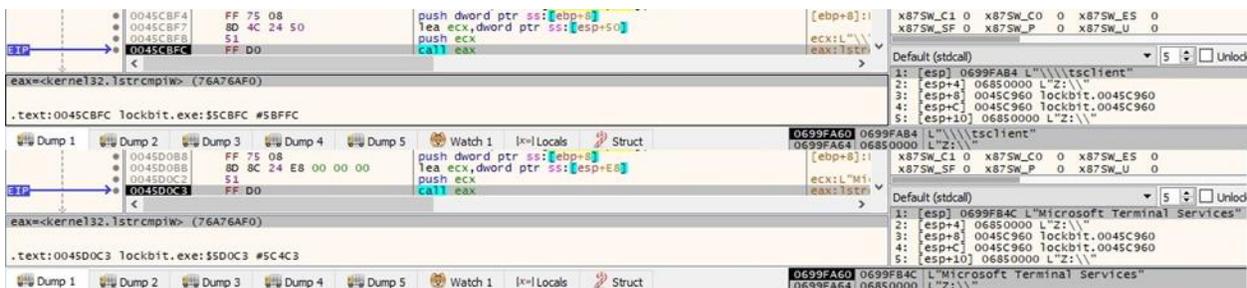


Figure 107

The CreateFileW function is utilized to create a file called “2ED873D4.lock” (0xC0000000 = **GENERIC_READ** | **GENERIC_WRITE**, 0x1 = **CREATE_NEW**, 0x04000100 = **FILE_FLAG_DELETE_ON_CLOSE** | **FILE_ATTRIBUTE_TEMPORARY**):

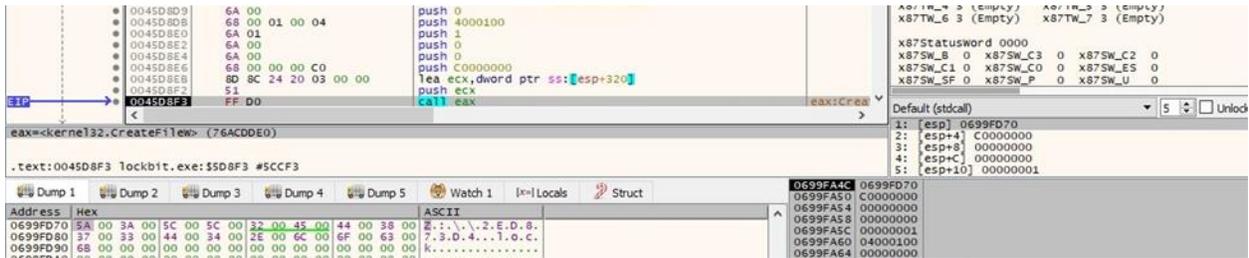


Figure 108

SHEmptyRecycleBinW is used to empty the Recycle Bin on the drive (0x7 = **SHERB_NOCONFIRMATION** | **SHERB_NOPROGRESSUI** | **SHERB_NOSOUND**):



Figure 109

The executable retrieves information about the total amount of space and the total amount of free space on the drive by calling the GetDiskFreeSpaceW and GetDiskFreeSpaceExW APIs:



Figure 110

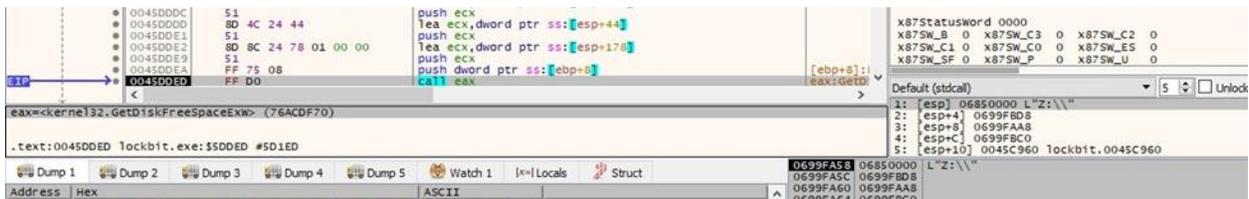


Figure 111

The user interface language for the current thread is set to “English - United States”:

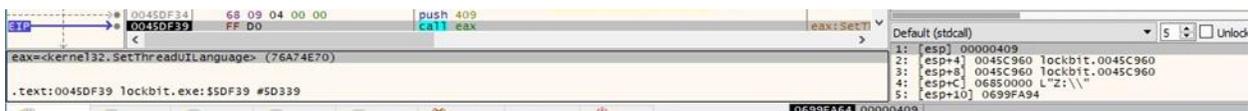


Figure 112

The numeric values extracted above are converted into a string that represents the size values in bytes, kilobytes, megabytes, or gigabytes, depending on their size:

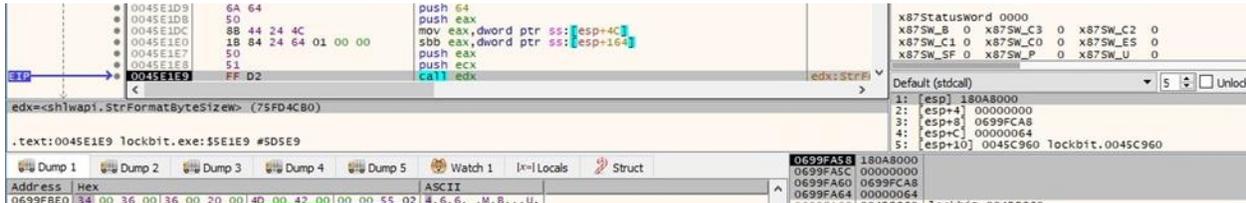


Figure 113

The drive name and the information regarding its size are sent to the hidden window via SendMessageW.

The FindFirstFileExW API is utilized to enumerate the drive:



Figure 114

The following directories will be skipped:

- system volume information
- windows photo viewer
- windows powershell
- internet explorer
- windows security
- windows defender
- microsoft shared
- application data
- windows journal
- \$recycle.bin
- \$windows~bt
- windows.old

The files enumeration is continued via a function call to FindNextFileW:

Figure 119

The NtQueryInformationFile routine is used to query file information (0x5 = FileStandardInformation):

Figure 120

NtSetInformationFile is utilized to set end-of-file information for the file (0x14 = FileEndOfFileInformation):

Figure 121

The following extensions list has been found:

- ".rar" ".zip" ".ckp" ".db3" ".dbf" ".dbc" ".dbs" ".dbt" ".dbv" ".frm" ".mdf"
- ".mrg" ".mwb" ".myd" ".ndf" ".qry" ".sdb" ".sdf" ".sql" ".tmd" ".wdb" ".bz2"
- ".tgz" ".lzo" ".db" ".7z" ".sqlite" ".accdb" ".sqlite3" ".sqlitedb" ".db-shm"
- ".db-wal" ".daccpac" ".zipx" ".lzma"

LockBit only encrypts the first 4KB of the file. It uses the ZwReadFile API in order to read 0x1000 (4096) bytes:

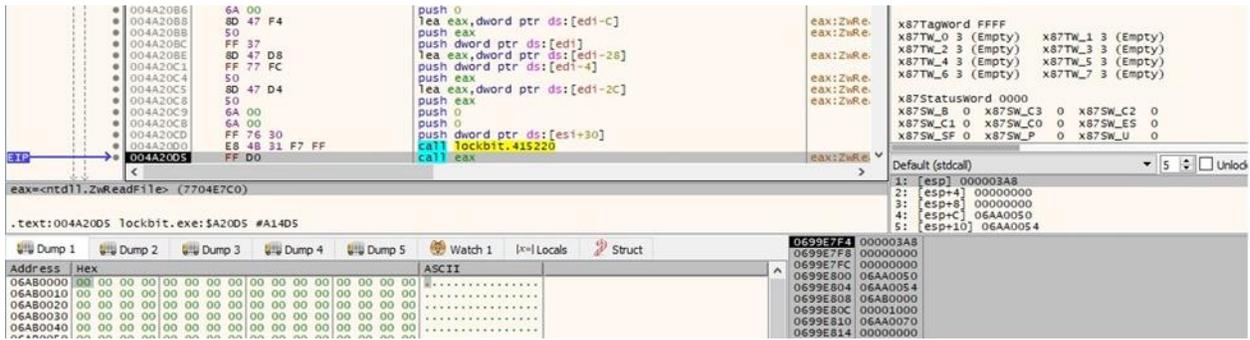


Figure 122

The GetFileAttributesW function is used to get file system attributes for the ransom note called "Restore-My-Files.txt":

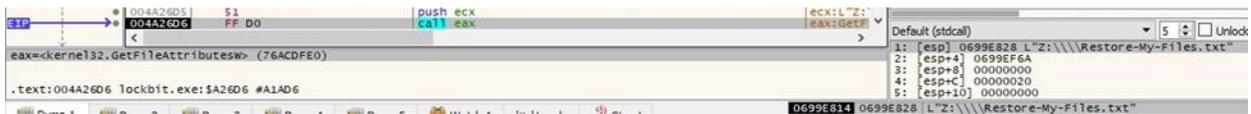


Figure 123

The ransomware creates the ransom note via a call to ZwCreateFile (0x10003 = FILE_READ_DATA | FILE_WRITE_DATA | DELETE, 0x80 = FILE_ATTRIBUTE_NORMAL, 0x2 = FILE_CREATE, 0x40 = FILE_NON_DIRECTORY_FILE):

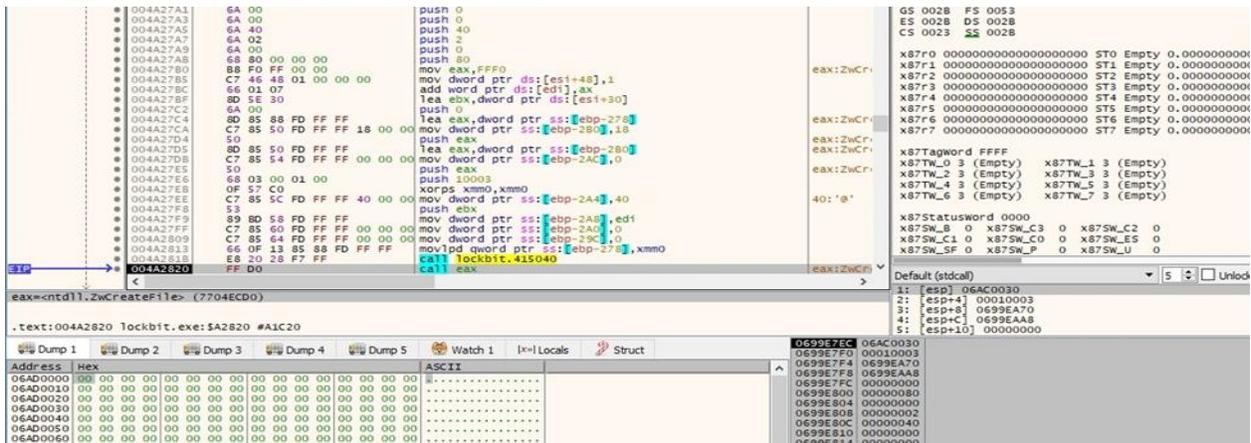


Figure 124

The ransom note is bound to the I/O completion port previously created via a function call to NtSetInformationFile (0x1E = FileCompletionInformation):

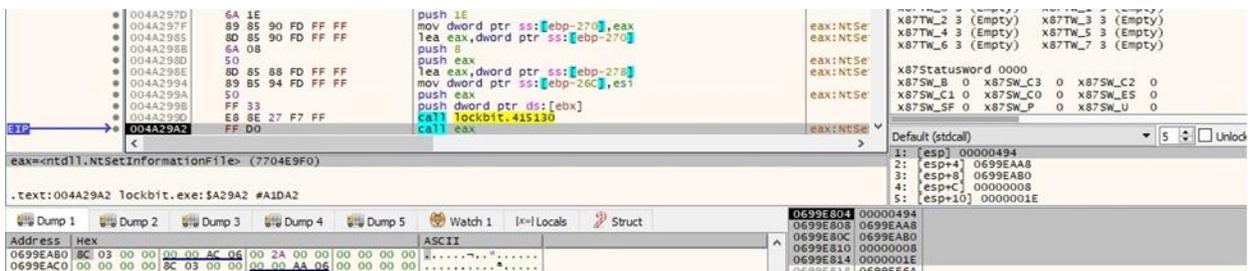


Figure 125

The note is populated using the ZwWriteFile routine:

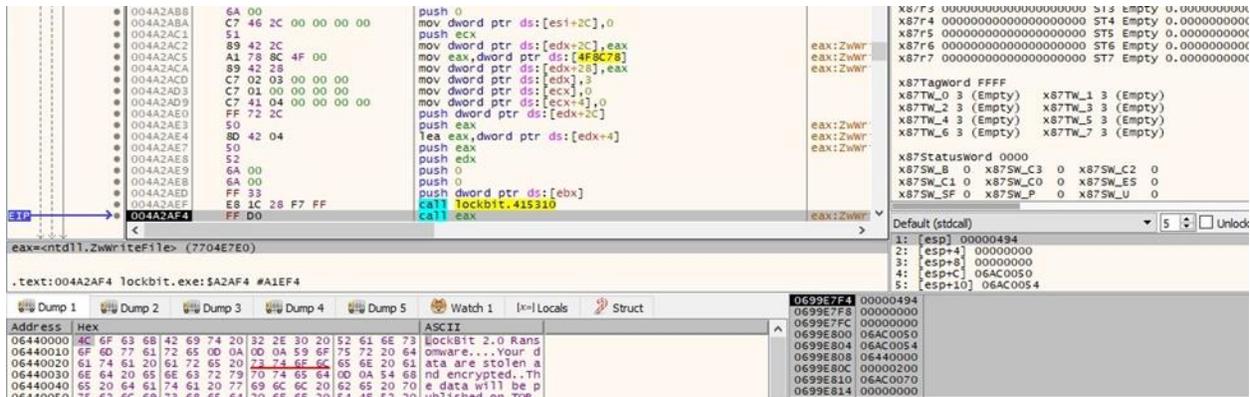


Figure 126

The ".lock" file created earlier is deleted after the drive enumeration is complete:



Figure 127

The content of the ransom note is displayed below:



Figure 128

The main thread sends the "Scan done, waiting handles..." message to the hidden window.

Thread activity – sub_497060 function

The malware retrieves the locally unique identifier (LUID) for the SeDebugPrivilege privilege using the LookupPrivilegeValueA routine:

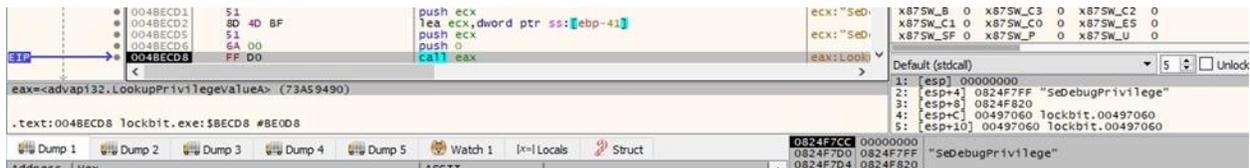


Figure 129

The privileges of the access token are adjusted to include the SeDebugPrivilege privilege via a function call to ZwAdjustPrivilegesToken:

Figure 130

OpenSCManagerA is used to establish a connection to the service control manager and to open the service control manager database (0xF003F = **SC_MANAGER_ALL_ACCESS**):

Figure 131

A targeted service is opened using the OpenServiceA API (0x2c = **SC_MANAGER_MODIFY_BOOT_CONFIG** | **SC_MANAGER_LOCK** | **SC_MANAGER_ENUMERATE_SERVICE**):

Figure 132

QueryServiceStatusEx is used to extract the current status of the service:

Figure 133

The EnumDependentServicesA routine is utilized to retrieve the name and status of each service that depends on the targeted service (see figure 134). These services will be stopped as well (0x1 = **SERVICE_ACTIVE**):

Figure 134

Every chosen service is stopped by calling the ControlService function (0x1 = **SERVICE_CONTROL_STOP**):



Figure 135

A confirmation message that the service was successfully stopped is sent to the hidden window:



Figure 136

The ransomware takes a snapshot of all processes in the system (0x2 = **TH32CS_SNAPPROCESS**):



Figure 137

The malicious file retrieves information about the first process from the snapshot via a function call to Process32First:



Figure 138

Interestingly, the malware removes the extension of the process name (if present) before the comparison with the targeted list:



Figure 139

An example of such a comparison is shown in figure 140.

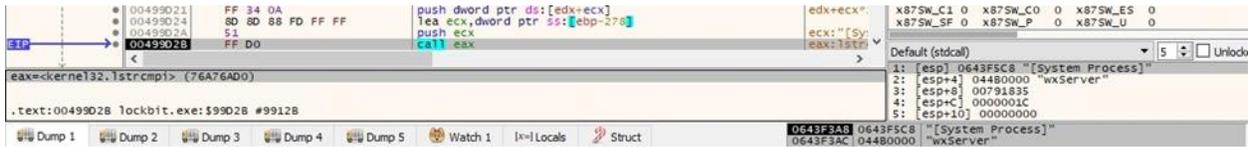


Figure 140

The process enumeration continues by calling the Process32Next routine:



Figure 141

OpenProcess is used to open a targeted process (0x1FFFFFF = **PROCESS_ALL_ACCESS**):



Figure 142

A process is killed by calling the NtTerminateProcess API:



Figure 143

LockBit initializes the COM library for apartment threading using the CoInitializeEx function (0x6 = **COINIT_APARTMENTTHREADED | COINIT_DISABLE_OLEIDE**):



Figure 144

The ransomware deletes all volume shadow copies on the system by calling the ShellExecuteEx function and running the commands shown below:



Figure 145

Address	Hex	ASCII
0643FA14	2F 63 20 76 73 73 61 64 6D 69 6E 20 64 65 6C 65	/c vssadmin dele
0643FA24	74 65 20 73 68 61 64 6F 77 73 20 2F 61 6C 6C 20	te shadows /all
0643FA34	2F 71 75 69 65 74 20 26 20 77 6D 69 63 20 73 68	/quiet & wmic sh
0643FA44	61 64 6F 77 63 6F 70 79 20 64 65 6C 65 74 65 20	adowcopy delete
0643FA54	26 20 62 63 64 65 64 69 74 20 2F 73 65 74 20 78	& bcdedit /set {
0643FA64	64 65 66 61 75 6C 74 7D 20 62 6F 6F 74 73 74 61	default} bootsta
0643FA74	74 75 73 70 6F 6C 69 63 79 20 69 67 6E 6F 72 65	tuspolicy ignore
0643FA84	61 6C 6C 66 61 69 6C 75 72 65 73 20 26 20 62 63	allfailures & bc
0643FA94	64 65 64 69 74 20 2F 73 65 74 20 78 64 65 66 61	dedit /set {defa
0643FAA4	75 6C 74 7D 20 72 65 63 6F 76 65 72 79 65 6E 61	ult} recoveryena
0643FAB4	62 6C 65 64 20 6E 6F 00 19 18 79 00 24 FB 43 06	bled no...y.Ûc.

Figure 146

The malware also creates multiple processes twice in order to delete (again) all shadow copies and Windows logs. An example of process creation is shown in figure 147 (0x08000000 = **CREATE_NO_WINDOW**):

Figure 147

The following processes have been spawned:

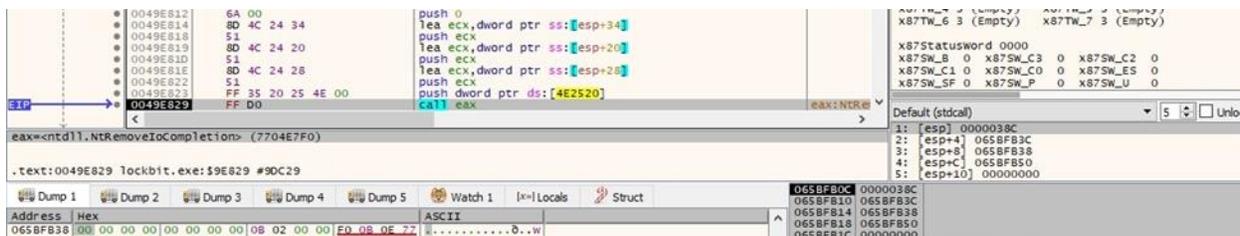
- cmd.exe /c vssadmin Delete Shadows /All /Quiet – delete all shadow copies
- cmd.exe /c bcdedit /set {default} recoveryenabled No – disable automatic repair
- cmd.exe c bcdedit set {default} bootstatuspolicy ignoreallfailures – ignore errors in the case of a failed boot / shutdown / checkpoint
- cmd.exe /c wmic SHADOWCOPY /nointeractive – invalid syntax
- cmd.exe /c wevtutil cl security – clear security log
- cmd.exe /c wevtutil cl system – clear system log
- cmd.exe /c wevtutil cl application – clear application log

The ransomware forwards the "Volume Shadow Copy & Event log clean" message to the hidden window:

Figure 148

Thread activity – sub_49E730 function

The NtRemoveIoCompletion function is utilized to wait for at least a file to be available for encryption:



```
0049E812 6A 00          push 0
0049E814 8D 4C 24 34   lea ecx,dword ptr ss:[esp+34]
0049E818 51          push ecx
0049E819 8D 4C 24 20   lea ecx,dword ptr ss:[esp+20]
0049E81D 51          push ecx
0049E81E 8D 4C 24 28   lea ecx,dword ptr ss:[esp+28]
0049E822 51          push ecx
0049E823 FF 35 20 25 4E 00 push dword ptr ds:[4E2520]
0049E829 FF D0       call eax
```

Figure 149

The following file extensions will be skipped:

- .386 .cmd .ani .adv .msi .msp .com .nls .ocx .mpa .cpl .mod .hta
- .prf .rtp .rdp .bin .hlp .shs .drv .wpx .bat .rom .msc .spl .msu
- .ics .key .exe .dll .lnk .ico .hlp .sys .drv .cur .idx .ini .reg
- .mp3 .mp4 .apk .ttf .otf .fon .fnt .dmp .tmp .pif .wav .wma .dmg
- .iso .app .ipa .xex .wad .msu .icns .lock .lockbit .theme .diagcfg
- .diagcab .diagpkg .msstyles .gadget .woff .part .sfcache .winmd

The files that can be found in the following directories will not be encrypted:

- "\$windows~bt" "intel" "\$recycle.bin" "to.msstyles" "boot" "msbuild" "system volume information"
- "google" "application data" "windows" "windows.old" "appdata" "mozilla" "microsoft shared" "internet explorer"
- "opera" "windows journal" "windows defender" "windowspowershell" "windows security" "windows photo viewer"

The following specific files will also be skipped:

- "iconcache.db" "ntuser.dat.log" "restore-my-files.txt" "autorun.inf" "bootsect.bak" "thumbs.db"

LockBit uses multiple aeskeygenassist operations in order to assist in AES round key generation, as we can see below:

```

.text:0043D970 sub_43D970 proc near
.text:0043D970 movups xmm1, xmmword ptr [edx]
.text:0043D973 aeskeygenassist xmm0, xmm1, 1
.text:0043D979 pshufd xmm3, xmm0, 0FFh
.text:0043D97E movaps xmm0, xmm1
.text:0043D981 pslldq xmm0, 4
.text:0043D986 pxor xmm0, xmm1
.text:0043D98A movups xmmword ptr [ecx], xmm1
.text:0043D98D movaps xmm1, xmm0
.text:0043D990 pslldq xmm1, 4
.text:0043D995 pxor xmm1, xmm0
.text:0043D999 movaps xmm2, xmm1
.text:0043D99C pslldq xmm2, 4
.text:0043D9A1 pxor xmm2, xmm1
.text:0043D9A5 pxor xmm2, xmm3
.text:0043D9A9 aeskeygenassist xmm0, xmm2, 2
.text:0043D9AF pshufd xmm3, xmm0, 0FFh
.text:0043D9B4 movaps xmm0, xmm2
.text:0043D9B7 pslldq xmm0, 4
.text:0043D9BC pxor xmm0, xmm2
.text:0043D9C0 movups xmmword ptr [ecx+10h], xmm2
.text:0043D9C4 movaps xmm1, xmm0
.text:0043D9C7 pslldq xmm1, 4
.text:0043D9CC pxor xmm1, xmm0
.text:0043D9D0 movaps xmm2, xmm1
.text:0043D9D3 pslldq xmm2, 4
.text:0043D9D8 pxor xmm2, xmm1
.text:0043D9DC pxor xmm2, xmm3
.text:0043D9E0 aeskeygenassist xmm0, xmm2, 4
.text:0043D9E6 pshufd xmm3, xmm0, 0FFh
.text:0043D9EB movaps xmm0, xmm2
.text:0043D9EE pslldq xmm0, 4
.text:0043D9F3 pxor xmm0, xmm2
.text:0043D9F7 movups xmmword ptr [ecx+20h], xmm2
.text:0043D9FB movaps xmm1, xmm0
.text:0043D9FE pslldq xmm1, 4
.text:0043DA03 pxor xmm1, xmm0
.text:0043DA07 movaps xmm2, xmm1
.text:0043DA0A pslldq xmm2, 4
.text:0043DA0F pxor xmm2, xmm1
.text:0043DA13 pxor xmm2, xmm3
.text:0043DA17 aeskeygenassist xmm0, xmm2, 8
.text:0043DA1D pshufd xmm3, xmm0, 0FFh
.text:0043DA22 movaps xmm0, xmm2
.text:0043DA25 pslldq xmm0, 4
.text:0043DA2A pxor xmm0, xmm2

```

Figure 150

Address	Hex	ASCII
0658FE40	BC 77 43 88 2F F4 A2 C0 63 3B F3 68 85 17 37 FC	¼wC./ðAc;òk..7Ù
0658FE50	4D ED F3 1F 62 19 51 DF 01 22 A2 B4 84 35 95 48	Miö.b.QB."c".5.H
0658FE60	D9 C7 A1 40 BB DE F0 9F BA FC 52 28 3E C9 C7 63	Üçj@»pð.ºUR+>ËCC
0658FE70	00 01 5A F2 BB DF AA 6D 01 23 F8 46 3F EA 3F 25	..Z0»E*m.#0F?E?%
0658FE80	8F 74 65 87 34 AB CF EA 35 88 37 AC 0A 62 08 89	.te.4«Ië5.7~.b..
0658FE90	35 44 C2 E0 01 EF 0D 0A 34 67 3A A6 3E 05 32 2F	SDAa.î..4g: >..b.
0658FEA0	7E 67 D7 52 7F 88 DA 58 48 EF E0 FE 75 EA D2 D1	~gxR..ÜXK1apuëN
0658FEB0	B9 D2 E9 CF C6 5A 33 97 8D B5 D3 69 F8 5F 01 88	°eIÄZ3...µ01o..
0658FEC0	F6 AE 85 8E 30 F4 B6 19 BD 41 65 70 45 1E 64 C8	0º...00!½AepE.dÉ
0658FED0	9F ED 6D E0 AF 19 DB F9 12 58 BE 89 57 46 DA 41	ima".0ü.X%.WFOA
0658FEE0	F3 BA EE BB 5C A3 35 42 4E FB 8B CB 19 BD 51 8A	0ºî»\£5BNü.E.%Q

Figure 151

The file content is encrypted using the AES128 algorithm. Basically, the malware uses aesenc instructions to perform one round of an AES encryption flow:

```

.text:0043D8E0
.text:0043D8E0 loc_43D8E0:
.text:0043D8E0 lea eax, [eax+10h]
.text:0043D8E3 movups xmm0, xmmword ptr [esi+eax-10h]
.text:0043D8E8 pxor xmm1, xmm0
.text:0043D8EC pxor xmm1, xmmword ptr [ecx]
.text:0043D8F0 aesenc xmm1, xmmword ptr [ecx+10h]
.text:0043D8F6 aesenc xmm1, xmmword ptr [ecx+20h]
.text:0043D8FC aesenc xmm1, xmmword ptr [ecx+30h]
.text:0043D902 aesenc xmm1, xmmword ptr [ecx+40h]
.text:0043D908 aesenc xmm1, xmmword ptr [ecx+50h]
.text:0043D90E aesenc xmm1, xmmword ptr [ecx+60h]
.text:0043D914 aesenc xmm1, xmmword ptr [ecx+70h]
.text:0043D91A aesenc xmm1, xmmword ptr [ecx+80h]
.text:0043D923 aesenc xmm1, xmmword ptr [ecx+90h]
.text:0043D92C aesenclast xmm1, xmmword ptr [ecx+0A0h]
.text:0043D935 movups xmmword ptr [eax-10h], xmm1
.text:0043D939 sub edx, 1
.text:0043D93C jnz short loc_43D8E0

```

Figure 152

Address	Hex	ASCII
06AB0000	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0010	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0020	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0030	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0040	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0050	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0060	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0070	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0080	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0090	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB00A0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB00B0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB00C0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB00D0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB00E0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB00F0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0100	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0110	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0120	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06AB0130	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA

Figure 153

Address	Hex	ASCII
06AB0000	C8 89 88 60 3E F8 64 12 B3 F1 7E D7 2A 1A 8A EE	E...>od.*h~x*..i
06AB0010	6C 08 52 F5 6D FD 6A 3A 80 EF D5 99 C1 D7 24 F2	l.R6myj:":t0.Ax5b
06AB0020	72 3E B1 2C 60 EA 4C 71 A1 16 DB FF 85 A0 29 67	r>:,eLqi,0y.,ig
06AB0030	09 A0 97 A2 C1 1E E0 8A 5D 47 9D 6E 5E E5 C5 76	.,eA,a.]G.n6Av
06AB0040	94 64 92 12 FE D0 7F F2 A2 82 B7 6F 03 D2 C8 A2	.d.,DD,0e*+o,0e*
06AB0050	80 56 22 43 D3 D6 F6 85 3D 38 8C 14 31 A1 E1 E3	.v"CO00u=.,iia&
06AB0060	75 66 92 5A E0 68 95 C0 86 00 94 93 D4 4C 7D 80	uf.Zak.Aq...0L}^
06AB0070	C1 CD 8F 13 52 E8 A3 F2 05 70 61 2D 35 00 BD 8D	Ai..ReEd.pa=5.%s.
06AB0080	30 03 88 8E CB 06 D0 BC 99 E0 31 26 D9 82 08 88	O...E.D%.ai&U...
06AB0090	39 E9 4C 53 A4 5F 43 8D BF FF 05 9B A1 72 E3 BE	9eLS=C.zY..Ar&A
06AB00A0	4C A6 1D AA OE 75 43 A1 0D AF 12 DF D2 2A 25 70	L'.*,ucI..B0*Sp
06AB00B0	29 CA EE DF F5 9F D0 17 A1 57 01 C5 2E F4 D7 C5)E1B6.D.iW.A.0xA
06AB00C0	06 50 B5 43 59 C8 44 A8 86 E6 00 A4 AA 31 93 C8	.PpCYED*.e.*1.E
06AB00D0	19 2F 71 B1 43 D1 1A 41 17 DA 6D 81 DD 9B F1 8F	.,qzCN.A.UmzY.h.
06AB00E0	E5 4E 35 E7 53 2E 7C 71 20 7F A7 69 CA 55 79 BE	AN5CS.lq.5iEuy%
06AB00F0	91 6C 28 88 EA E6 3E 13 C7 0D 22 95 AC 2D 9F F3	.l(.eaz.C."-.-o
06AB0100	E9 47 BC 77 70 80 DE C9 20 3F 80 44 0F B8 69 C2	eGwP.DE?D..iA
06AB0110	64 38 44 02 D0 FB F6 AF C4 C5 38 56 BC 99 31 AD	d8D.D00.AA8V%.1.
06AB0120	9A 3A 40 0C 66 68 AF 58 CC A9 9E 36 D9 82 87 C4	.:0.Fk lIe.6U..A
06AB0130	24 6E 0E A7 FC F2 0E D3 98 92 65 82 33 2D 44 5E	\$na\$uo.O..e*3-D^

Figure 154

As we mentioned before, only the first 4KB of the file is encrypted. The encrypted content is written to the file using ZwWriteFile:

Figure 155

The BcryptGenRandom routine is utilized to generate 32 random bytes:

Figure 156

The buffer generated above is transformed using the Curve25519 wrapper and then copied to a new buffer together with the session ECC public key (see figure 157). Based on the implementation of the Curve25519 algorithm, it is used to generate a shared secret (32-byte value).

Address	Hex	ASCII
065BF8E0	39 23 1A E5 80 F7 25 91 20 63 11 0A F7 98 91 56	9#.ã.±%.ç.±.V
065BF8F0	05 C5 A3 C4 28 56 41 85 EA D0 CD 2E F1 83 D0 76	.,AfA(VAU)ëDÏ.ñ*DV
065BF900	A5 27 53 28 E9 D0 F0 C9 24 B3 0B 74 66 F4 FC 4B	ÿ'S+ëD0E\$\$.t'fôùk
065BF910	DA 5F 25 A8 37 D8 DC D7 4B 50 C5 86 71 DA D5 3E	U.%70ùxKFA.ü0Üx

Figure 157

The AES128 key and IV (initialization vector) are encrypted using Curve25519 with the session ECC public key, as highlighted below:



Figure 158

Each encrypted file has a 512-byte footer that will be explained in detail. It's written to the encrypted file by calling the ZwWriteFile API:

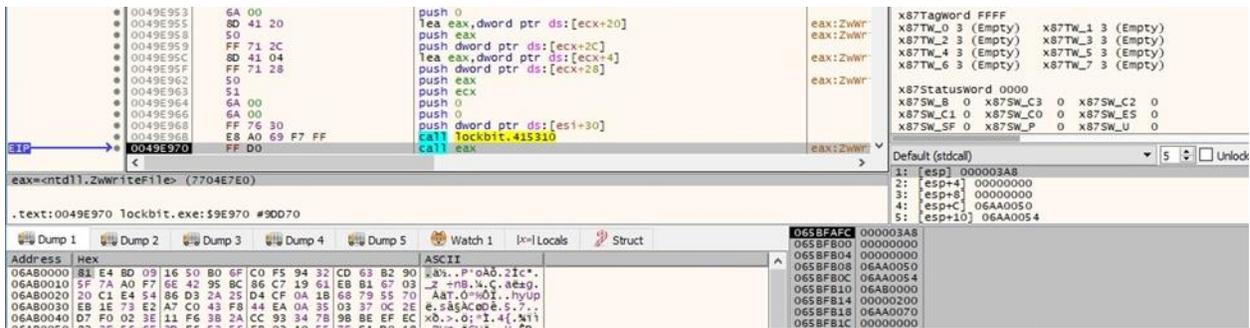


Figure 159

NtSetInformationFile is used to append the ".lockbit" extension to encrypted files (0xA = FileRenameInformation):

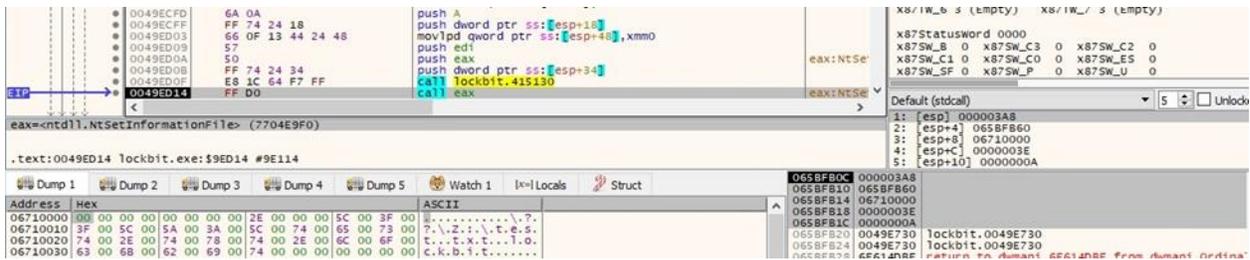


Figure 160

As we can see below, the files are partially encrypted, which is enough to make them useless without decrypting them:

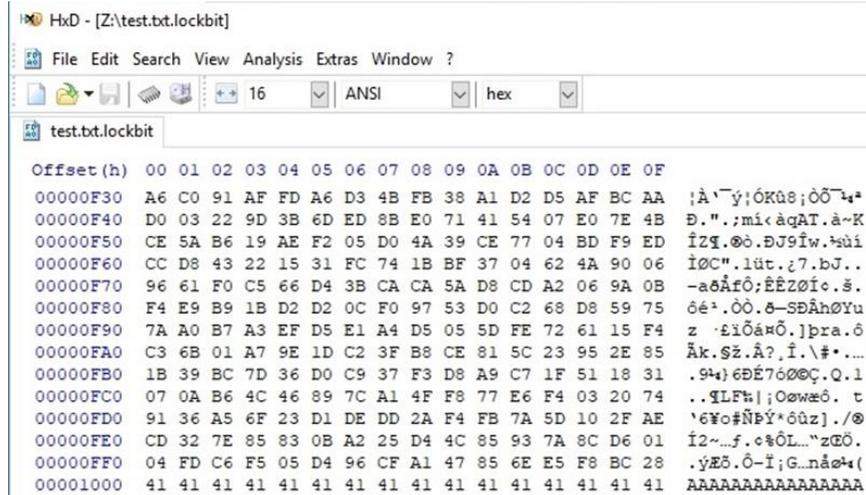


Figure 161

Out of the 512 bytes from the footer, we can highlight the following bytes:

- last 8 bytes - first 8 bytes from the session ECC public key
- previous 8 bytes - hard-coded bytes that correspond to this particular LockBit sample
- 112 bytes - session ECC private key that was encrypted using the Master ECC public key (also stored in the Private registry value)
- 96 bytes – AES key + IV that were encrypted using the session ECC public key

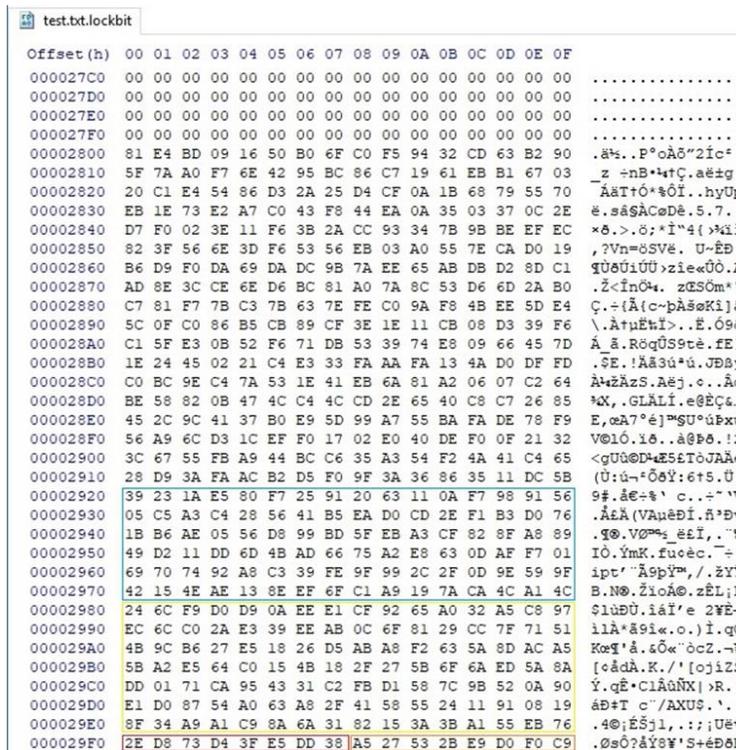


Figure 162

We can observe the icon of the encrypted files in figure 163:

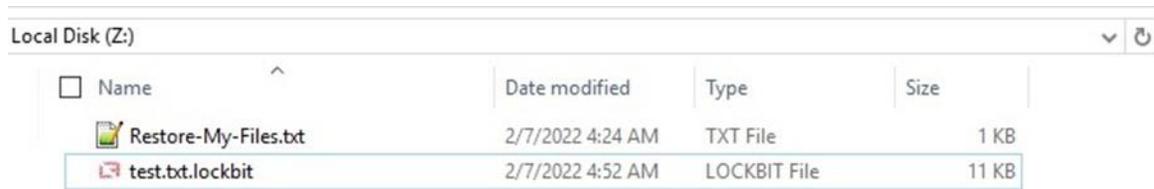


Figure 163

We continue with the analysis of the main thread.

The binary sends the "Cleanup" message to the hidden window via a function call to SendMessageA.

Printing ransom notes

The process enumerates the local printers using the EnumPrintersW function (0x2 = PRINTER_ENUM_LOCAL):

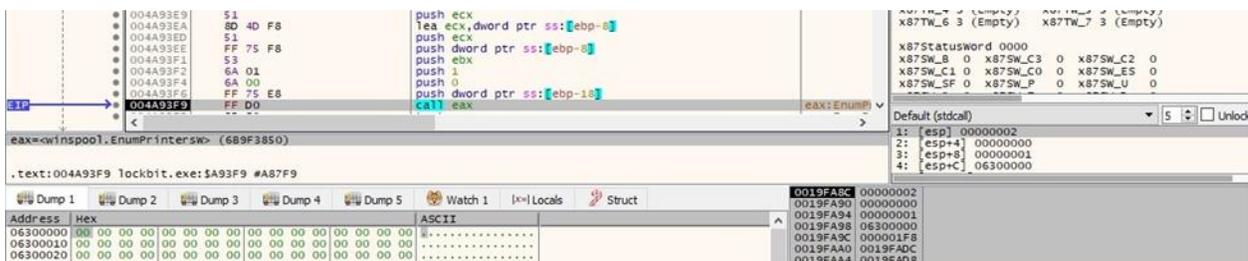


Figure 164

The ransomware avoids the following values that don't correspond to physical printers: "Microsoft XPS Document Writer" and "Microsoft Print to PDF".

The OpenPrinterW routine is utilized to retrieve a handle to the printer:



Figure 165

StartDocPrinterW is used to notify the print spooler that a document is to be spooled for printing:



Figure 166

The StartPagePrinter API notifies the spooler that a page will be printed on the printer:



Figure 167

The ransom note is printed via a function call to WritePrinter:

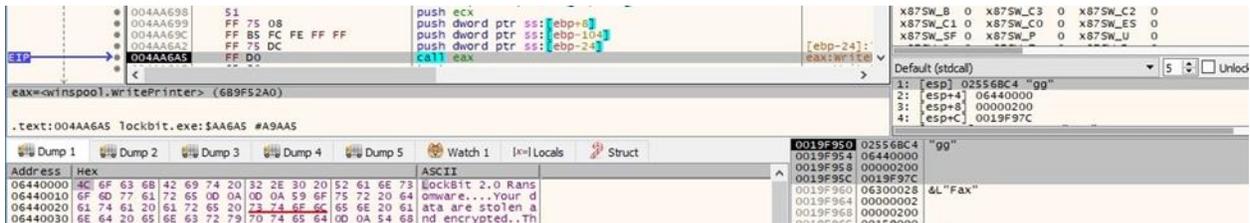


Figure 168

The EndPagePrinter routine notifies the print spooler that the application is at the end of a page in the print job:



Figure 169

The printing operation is effected 10000 times, as displayed in figure 170:

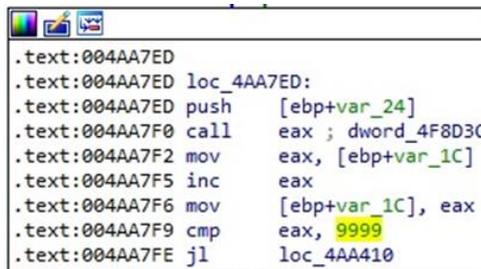


Figure 170

The print job operation is completed by calling the EndDocPrinter and ClosePrinter APIs.

LockBit continues the printer enumeration by searching for network printers in the computer's domain, network printers and print servers in the computer's domain, and the list of printers to which the user has made previous connections. These function calls can be seen below (0x40 = **PRINTER_ENUM_NETWORK**, 0x10 = **PRINTER_ENUM_REMOTE**, 0x4 = **PRINTER_ENUM_CONNECTIONS**):



Figure 171

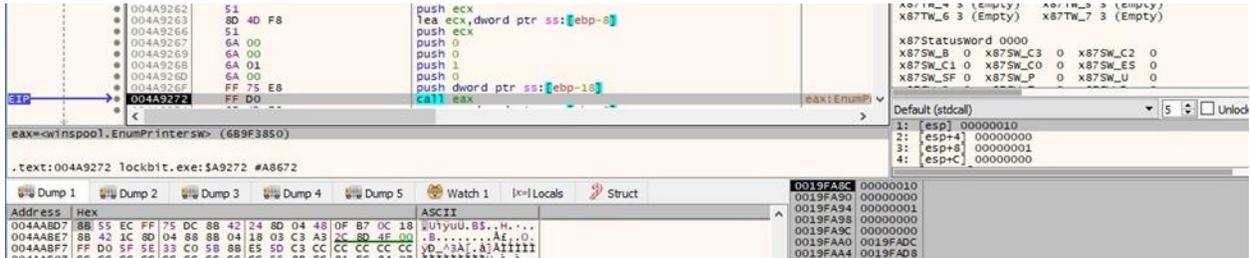


Figure 172

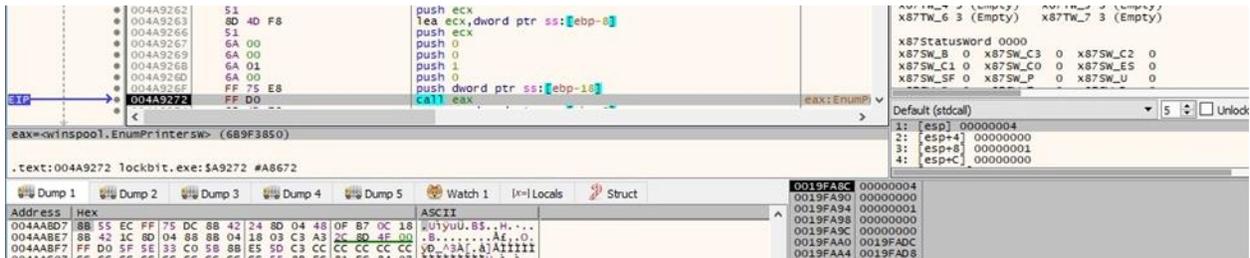


Figure 173

LockBit Wallpaper Setup

The ransomware sends the "[+] Setup wallpaper" message to the hidden window.

The GdiplusStartup API is utilized to initialize Windows GDI+:

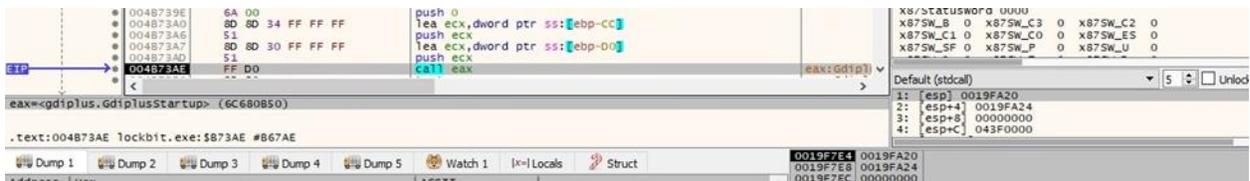


Figure 174

The file retrieves the width of the screen of the primary display monitor via a function call to GetSystemMetrics:

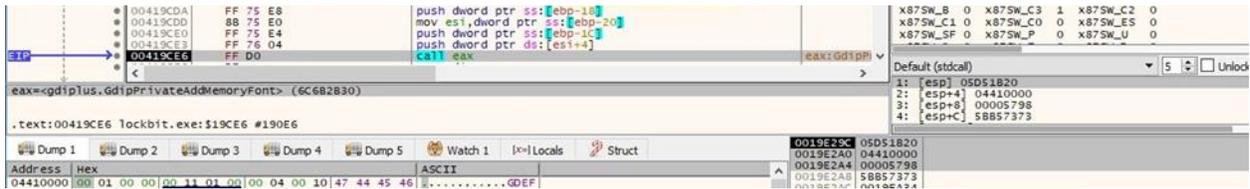


Figure 181

The GdipGetImageGraphicsContext function is used to create a Graphics object that is associated with an image object:

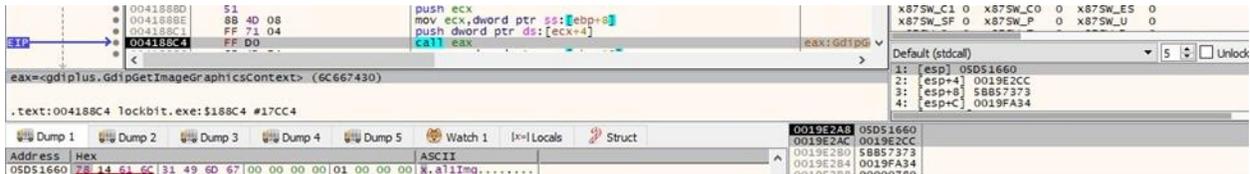


Figure 182

The malware creates multiple SolidBrush objects based on different colors using the GdipCreateSolidFill routine:



Figure 183

All SolidBrush objects are used to fill the interior of multiple rectangles using GdipFillRectangle. The GdipSetPageUnit API is utilized to set the unit of measure for a Graphics object:

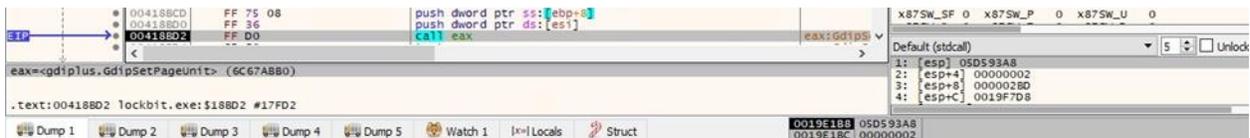


Figure 184

GdipCreatePen1 is used to create a Pen object:

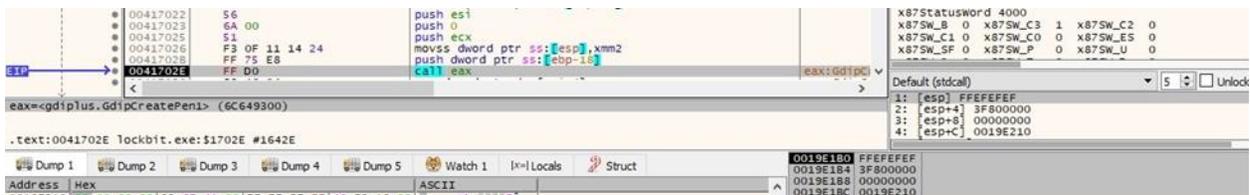


Figure 185

LockBit creates a GraphicsPath object via a function call to GdipCreatePath:



Figure 186

The process performs multiple GdiplAddPathArc calls in order to add elliptical arcs to the current figure of the path:



Figure 187

The ransomware performs function calls such as GdiplFillPath and GdiplDrawPath in order to transform the path. It creates a FontFamily object based on the Proxima Nova Font family:

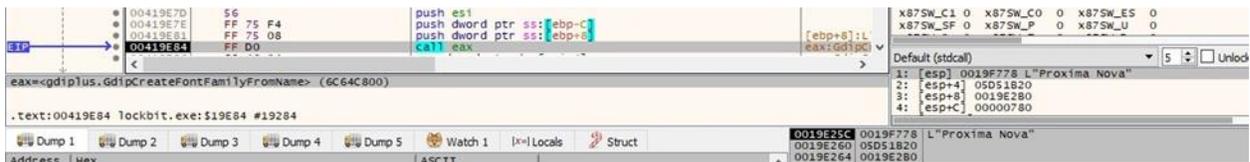


Figure 188

A Font object is created based on the above object via GdiplCreateFont:



Figure 189

The GdiplDrawImageRect function is utilized to draw an image:



Figure 190

The malware measures the extent of the strings that will appear in the wallpaper by calling the GdiplMeasureString API:

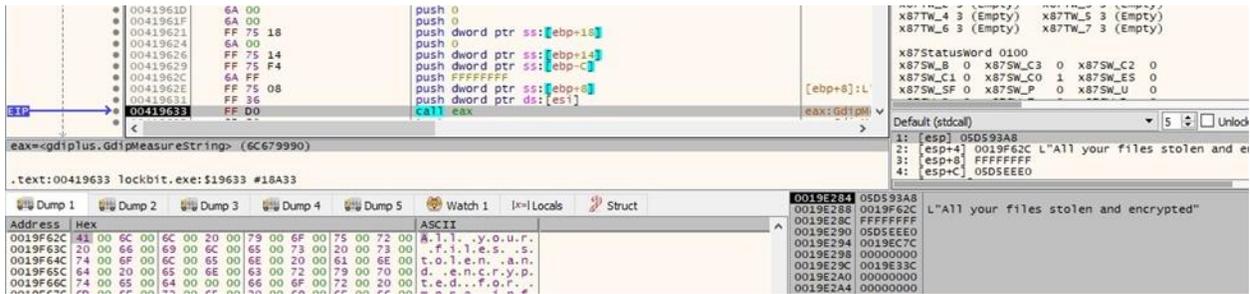


Figure 191

The process draws the strings based on a font, a layout rectangle, and a format via a call to GdiplDrawString:

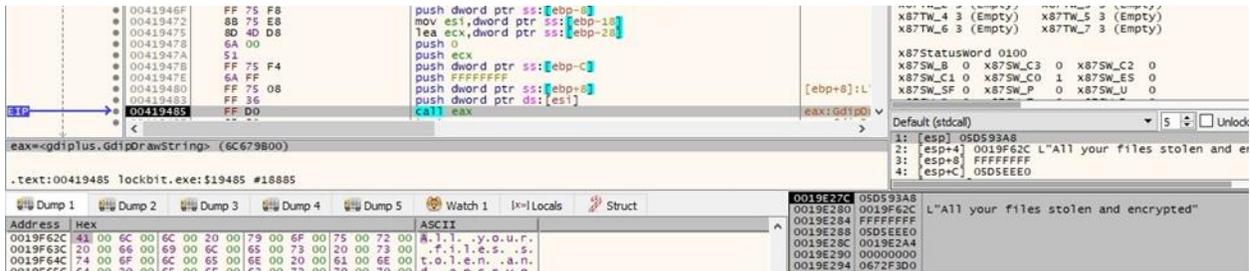


Figure 192

The file extracts the path of the %TEMP% directory:



Figure 193

GetTempFileNameW is utilized to create a temporary file:

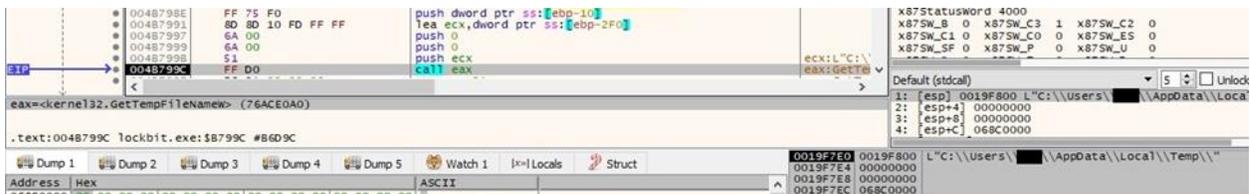


Figure 194

The GdiplGetImageEncoders function is used to retrieve an array of ImageCodecInfo objects containing information about the available image encoders:



Figure 195

The image constructed in memory is saved to the disk in the temporary file created earlier:

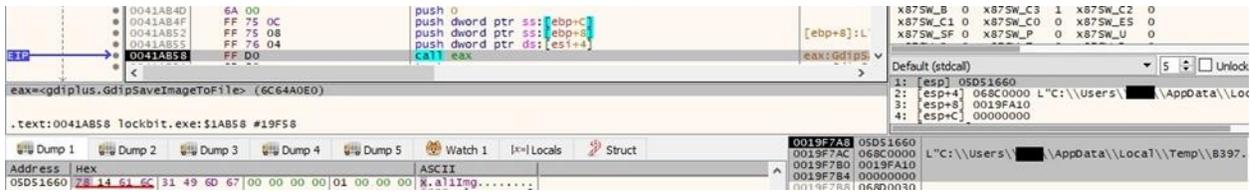


Figure 196

Figure 197 shows the wallpaper that will be set:



Figure 197

The RegOpenKeyA API is utilized to open the "Control Panel\Desktop" registry key (0x80000001 = HKEY_CURRENT_USER):

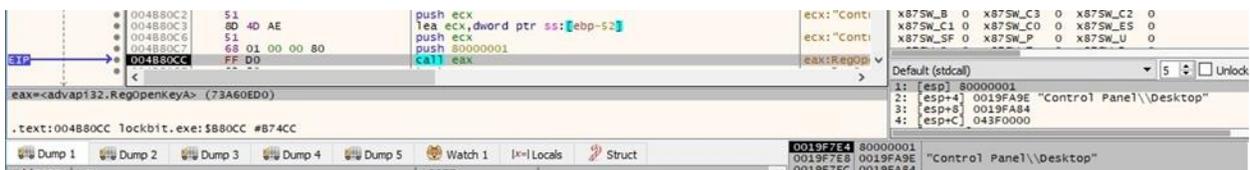


Figure 198

The "WallpaperStyle" registry value is set to 2, and the "TileWallpaper" value is set to 0 by calling the RegSetValueExA routine (0x1 = REG_SZ):

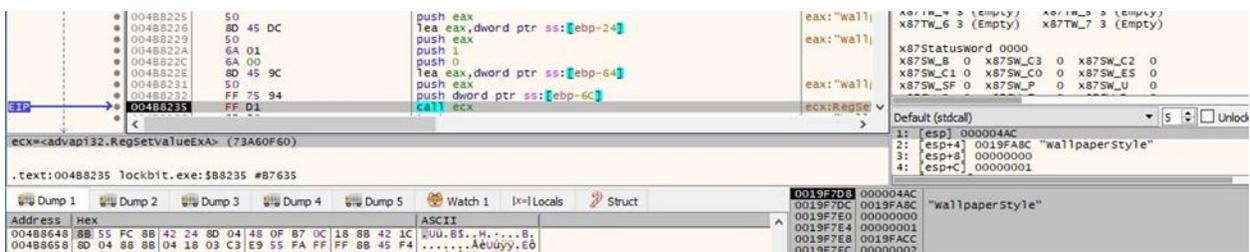


Figure 199

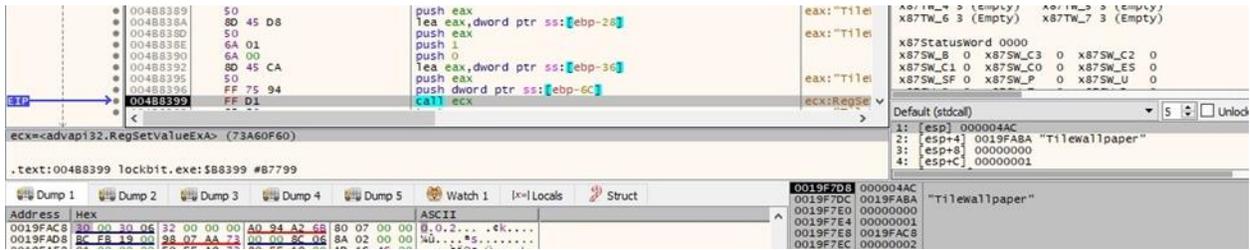


Figure 200

The Desktop wallpaper is set by calling the SystemParametersInfoW function (0x14 = **SPI_SETDESKWALLPAPER**, 0x3 = **SPIF_UPDATEINIFILE** | **SPIF_SENDCHANGE**):



Figure 201

As we can see in the next picture, the registry values were successfully modified:

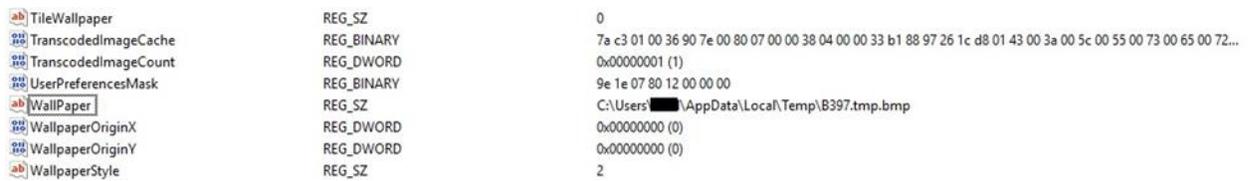


Figure 202

Extract and save the HTA ransom note to Desktop

LockBit sends the "[+] Extract *.hta file" message to the hidden window. The HTA ransom note is stored in an encrypted form in the executable. It is decrypted using the XOR operator (key = 0x38).

The malicious binary creates a file called "LockBit_Ransomware.hta" on the user Desktop (0x40000000 = **GENERIC_WRITE**, 0x2 = **CREATE_ALWAYS**, 0x80 = **FILE_ATTRIBUTE_NORMAL**):



Figure 203

The WriteFile API is used to populate the HTA file:



Figure 204

The ZwCreateKey API is utilized to open the “HKCR\lockbit” registry key (0x2000000 = **MAXIMUM_ALLOWED**):

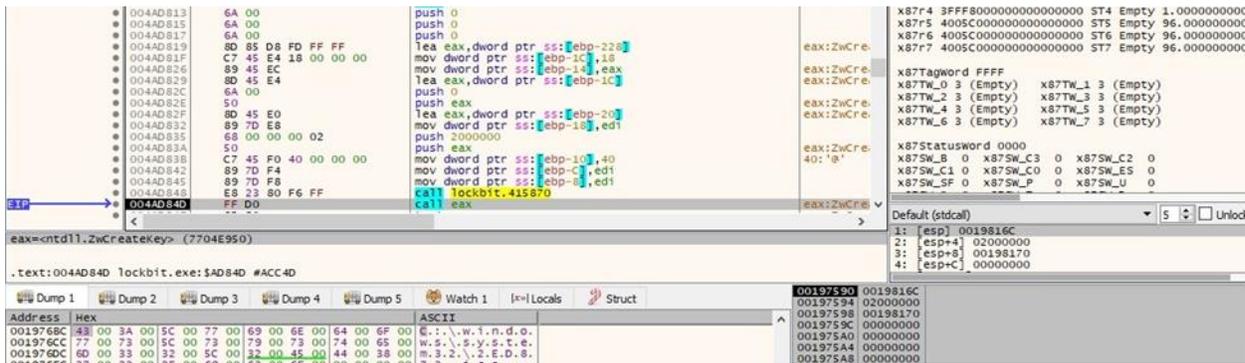


Figure 205

The (Default) registry value is set to "LockBit" by calling the ZwSetValueKey function (0x1 = **REG_SZ**):



Figure 206

The malware creates the “HKCR\Lockbit” registry key by calling the ZwCreateKey API (0x2000000 = **MAXIMUM_ALLOWED**):

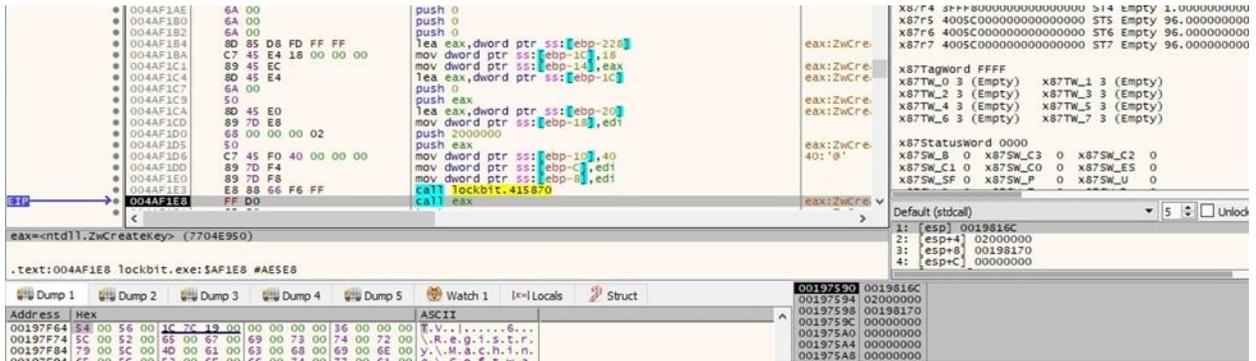


Figure 207

The DefaultIcon registry value is set to "C:\windows\SysWow64\2ED873.ico" using ZwSetValueKey (0x1 = **REG_SZ**):

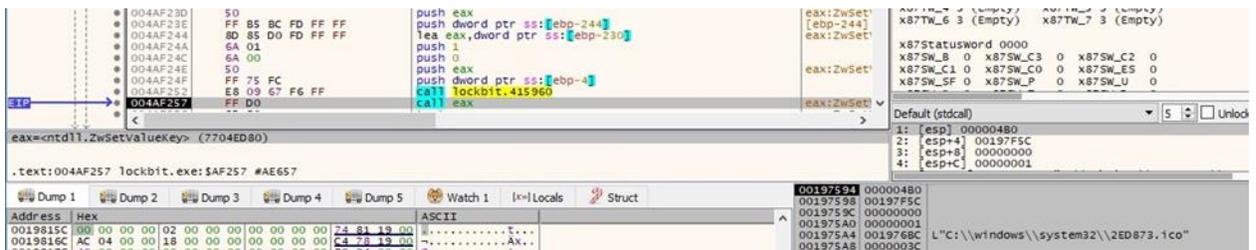


Figure 208

The process creates the following registry subkeys: "shell", "Open", and "Command". The (Default) value is set to "LockBit Class" using ZwSetValueKey (0x1 = **REG_SZ**):

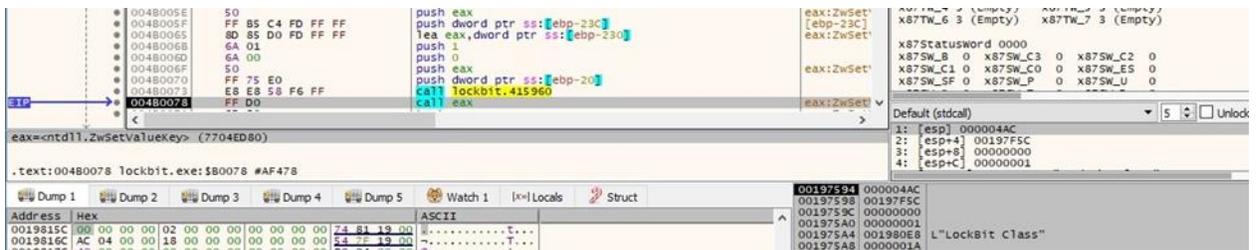


Figure 209

The (Default) registry value under the Command key is set to open the HTA ransom note:

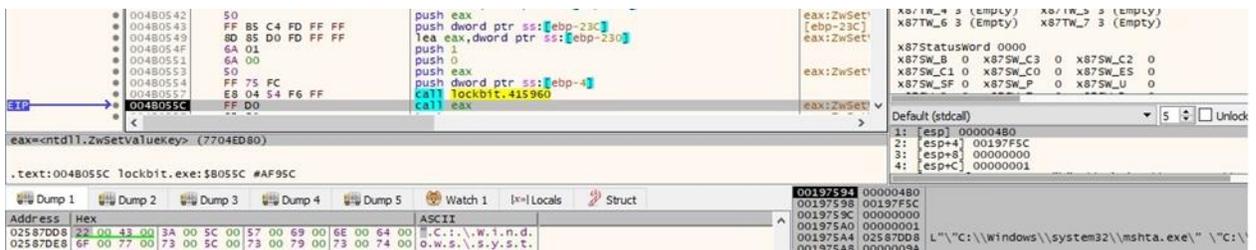


Figure 210

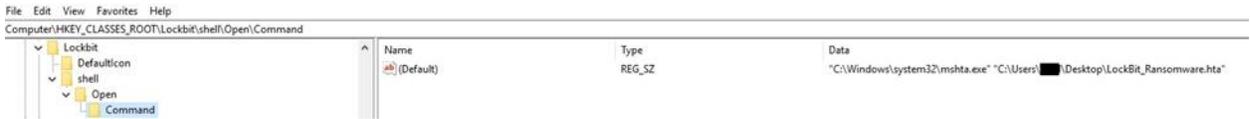


Figure 211

The NtOpenKey routine is utilized to open the “HKCR\hta” registry key (0x2000000 = **MAXIMUM_ALLOWED**):



Figure 212

The malicious binary retrieves the (Default) registry value via a function call to NtQueryValueKey (0x2 = **KeyValuePartialInformation**):

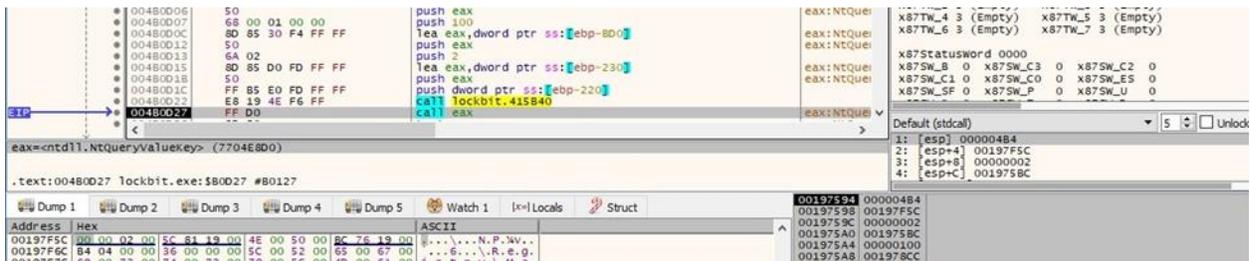


Figure 213

NtOpenKey is used to open the “HKCR\htafile” key (0x2000000 = **MAXIMUM_ALLOWED**):

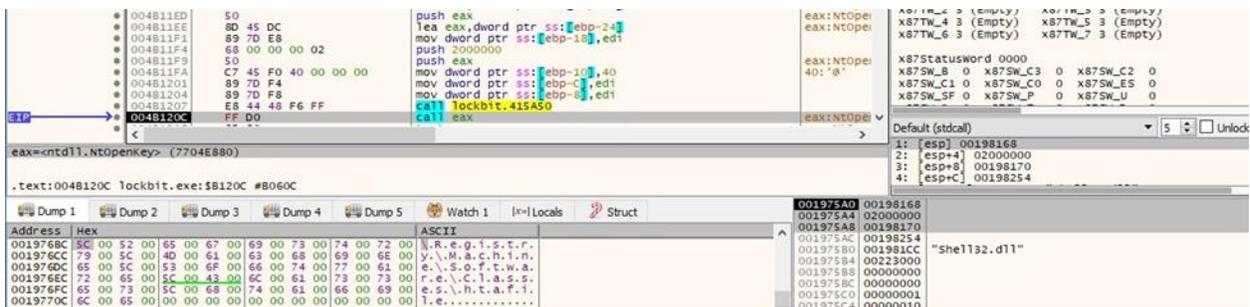


Figure 214

The DefaultIcon registry value is set to “C:\windows\SysWow64\2ED873.ico” (0x1 = **REG_SZ**):

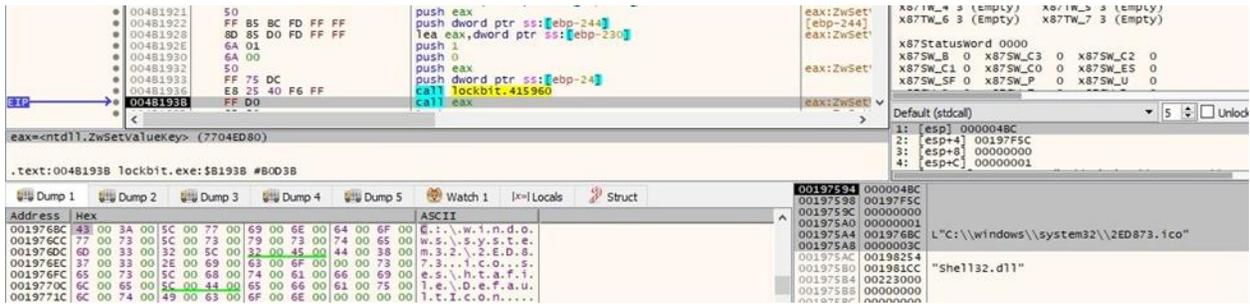


Figure 215

The file opens the Run registry key using RegCreateKeyExW (0x80000001 = HKEY_CURRENT_USER, 0x2001F = KEY_READ | KEY_WRITE):

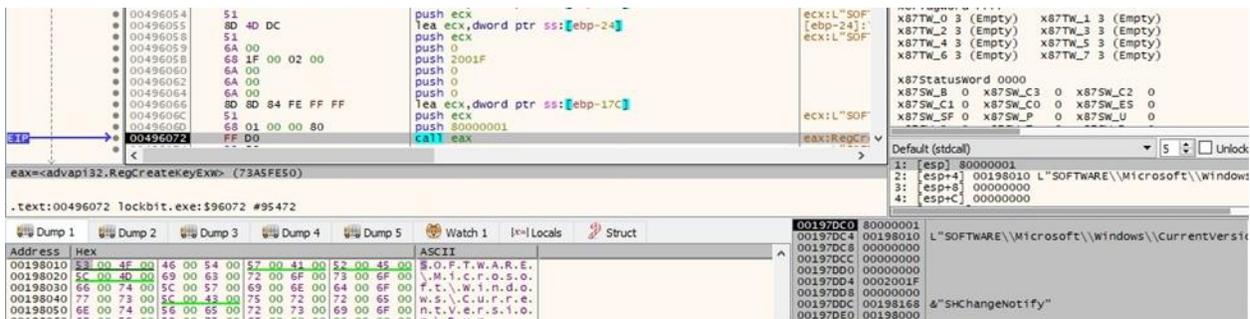


Figure 216

The ransomware creates a value called "{2C5F9FCC-F266-43F6-BFD7-838DAE269E11}", which contains the path to the HTA note (0x1 = REG_SZ):

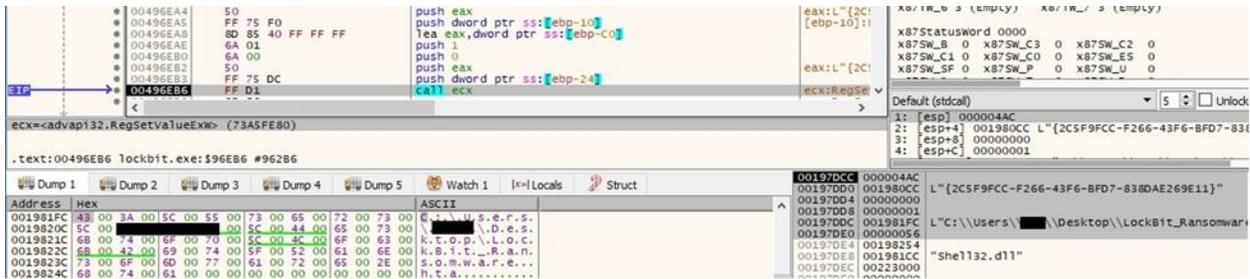


Figure 217

ShellExecuteW is utilized to open and display the above ransom note:

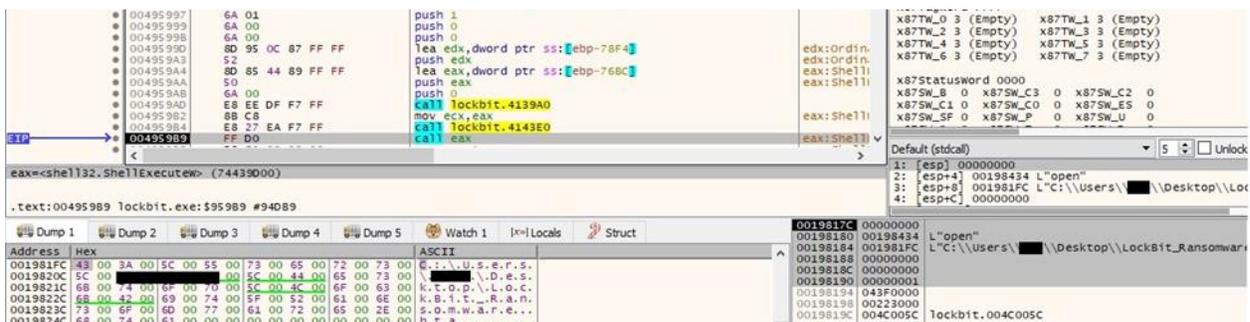


Figure 218

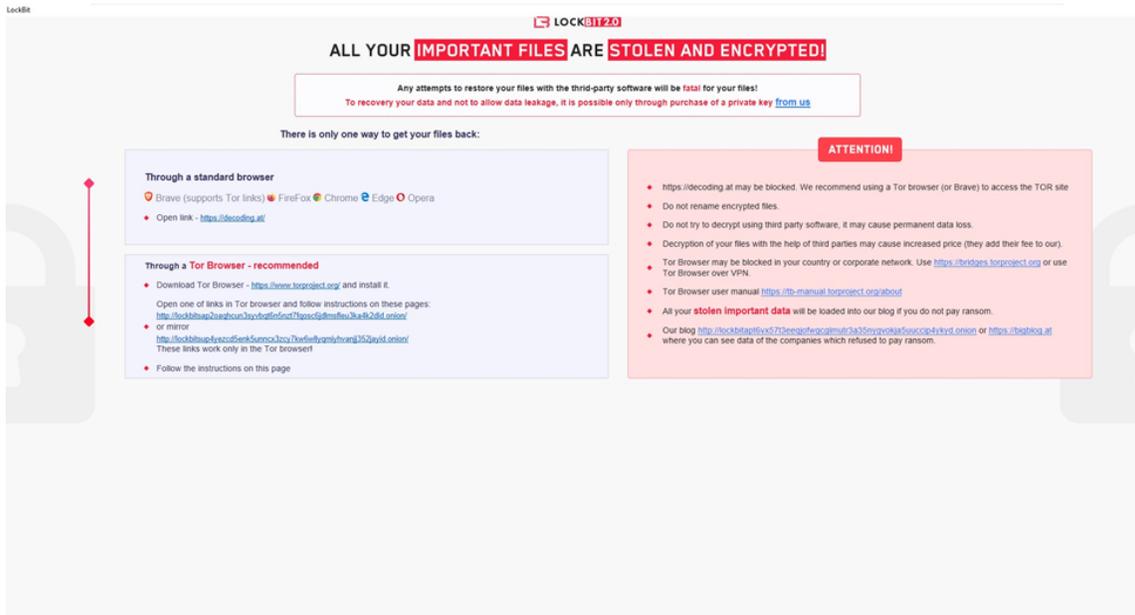


Figure 219

LockBit deletes the registry value used for persistence named "[9FD872D4-E5E5-DDC5-399C-396785BDC975]". We believe this value was created to resume the encryption process in the case of a reboot:



Figure 220

The executable sends the "[+] Removed autorun key" message to the hidden window using SendMessageA. There is a call to ZwSetIoCompletion afterward:



Figure 221

The malware deletes itself when the system restarts by calling the MoveFileExW function (0x4 = MOVEFILE_DELAY_UNTIL_REBOOT):

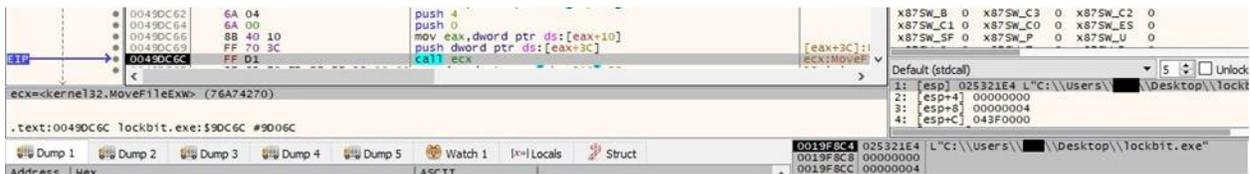


Figure 222

There is also a second process that will handle the executable deletion:

```
"cmd.exe /C ping 127.0.0.7 -n 3 > Nul & fsutil file setZeroData offset=0 length=524288
"C:\\Users\\<User>\\Desktop\\lockbit.exe" & Del /f /q "C:\\Users\\<User>\\Desktop\\lockbit.exe"
```

By pressing Shift+F1, we can access the hidden window:



Figure 223



Figure 224

Indicators of Compromise

Registry Keys

Key: HKEY_CLASSES_ROOT\Lockbit\shell\Open\Command

Data: "C:\Windows\system32\mshta.exe" "C:\Users\<User>\Desktop\LockBit_Ransomware.hta"

Key: HKEY_CLASSES_ROOT\Lockbit\DefaultIcon

Key: HKEY_CLASSES_ROOT\.lockbit\DefaultIcon

Key: HKEY_CLASSES_ROOT\htafile\DefaultIcon

Data: C:\windows\SysWow64\2ED873.ico

Key: SOFTWARE\Microsoft\Windows\CurrentVersion\Run\{2C5F9FCC-F266-43F6-BFD7-838DAE269E11}

Data: C:\Users\<User>\Desktop\LockBit_Ransomware.hta

Key: SOFTWARE\Microsoft\Windows\CurrentVersion\Run\{9FD872D4-E5E5-DDC5-399C-396785BDC975}

Data: <LockBit 2.0 file path>

Key: HKCU\Software\2ED873D4E5389C\Private

Key: HKCU\Software\2ED873D4E5389C\Public

Key: HKCU\Control Panel\Desktop

Data: Wallpaper = %AppData%\Local\Temp\<wallpaper>.tmp.bmp

Data: TileWallpaper = 0

Data: WallpaperStyle = 2

Files Created

C:\Users\<User>\Desktop\LockBit_Ransomware.hta

C:\windows\SysWow64\2ED873.ico

C:\Users\<User>\AppData\Local\Temp\<wallpaper>.tmp.bmp

C:\2ED873D4.lock (or any drive)

Processes spawned

cmd.exe /c vssadmin Delete Shadows /All /Quiet

cmd.exe /c bcdedit /set {default} recoveryenabled No

```
cmd.exe /c bcdedit /set {default} bootstatuspolicy ignoreallfailures
```

```
cmd.exe /c wmic SHADOWCOPY /nointeractive
```

```
cmd.exe /c wevtutil cl security
```

```
cmd.exe /c wevtutil cl system
```

```
cmd.exe /c wevtutil cl application
```

```
cmd.exe /c vssadmin delete shadows /all /quiet & wmic shadowcopy delete & bcdedit /set {default} bootstatuspolicy ignoreallfailures & bcdedit /set {default} recoveryenabled no
```

```
cmd.exe /C ping 127.0.0.7 -n 3 > Nul & fsutil file setZeroData offset=0 length=524288  
"C:\Users\<User>\Desktop\lockbit.exe" & Del /f /q "C:\Users\<User>\Desktop\lockbit.exe"
```

Mutex

```
\BaseNamedObjects\{3FE573D4-3FE5-DD38-399C-886767BD8875}
```

LockBit 2.0 Extension

```
.lockbit
```

LockBit 2.0 Ransom Note

```
Restore-My-Files.txt
```

```
LockBit_Ransomware.hta
```

Appendix

List of processes to be killed

wxServer wxServerView sqlmangr RAGui supervise Culture Defwatch winword QBW32 QBDBMgr qbupdate axlbridge httpd fdlauncher MsDtSrvr java 360se 360doctor wdswwsafe fdhost GDscan ZhuDongFangYu QBDBMgrN mysqld AutodeskDesktopApp acwebbrowser Creative Cloud Adobe Desktop Service CoreSync Adobe CEF Helper node AdobePCBroker sync-taskbar sync-worker InputPersonalization AdobeCollabSync BrCtrlCntr BrCcUxSys SimplyConnectionManager Simply.SystemTrayIcon fbguard fbserver ONENOTEM wsa_service koaly-exp-engine-service TeamViewer_Service TeamViewer tv_w32 tv_x64 TitanV Ssms notepad RdrCEF sam oracle ocspd dbnmp synctime agntsvc isqlplussvc xfssvcon mydesktopservice ocautoupds encsvc tbirdconfig mydesktopqos ocomm dbeng50 sqbcoreservice excel infopath msaccess mspub onenote outlook powerpnt steam thebat thunderbird visio wordpad bedbh vxmon benetns bengien pvlsvr beserver raw_agent_svc vsnapvss CagService DellSystemDetect EnterpriseClient ProcessHacker Procexp64 Procexp GlassWire GWCtlSrv WireShark dumpcap j0gnjko1 Autoruns Autoruns64 Autoruns64a Autorunsc Autorunsc64 Autorunsc64a Sysmon Sysmon64 procexp64a procmon procmon64 procmon64a ADEplorer ADEplorer64 ADEplorer64a tcpview tcpview64 tcpview64a avz tdsskiller RaccineElevatedCfg RaccineSettings Raccine_x86 Raccine Sqlservr RTVscan sqlbrowser tomcat6 QBIDPService notepad++ SystemExplorer SystemExplorerService SystemExplorerService64 Totalcmd Totalcmd64 VeeamDeploymentSvc

List of services to be stopped

wrapper DefWatch ccEvtMgr ccSetMgr SavRoam Sqlservr sqlagent sqladhlp Culserver RTVscan sqlbrowser SQLADHLP QBIDPService Intuit.QuickBooks.FCS QBCFMonitorService msmdsrv tomcat6 zhudongfangyu vmware-usbarbitator64 vmware-converter dbsrv12 dbeng8 MSSQL\$MICROSOFT##WID MSSQL\$VEEAMSQL2012 SQLAgent\$VEEAMSQL2012 SQLBrowser SQLWriter FishbowlMySQL MSSQL\$MICROSOFT##WID MySQL57 MSSQL\$KAV_CS_ADMIN_KIT MSSQLServerADHelper100 SQLAgent\$KAV_CS_ADMIN_KIT msftesql-Exchange MSSQL\$MICROSOFT##SSEE MSSQL\$SBSMONITORING MSSQL\$SHAREPOINT MSSQLFDLauncher\$SBSMONITORING MSSQLFDLauncher\$SHAREPOINT SQLAgent\$SBSMONITORING SQLAgent\$SHAREPOINT QBFCService QBVSS YooBackup YooIT vss sql svc\$ MSSQL MSSQL\$ memtas mepocs sophos veeam backup bedbg PDVFSService BackupExecVSSProvider BackupExecAgentAccelerator BackupExecAgentBrowser BackupExecDiveciMediaService BackupExecJobEngine BackupExecManagementService BackupExecRPCService MVArmor MVArmor64 stc_raw_agent VSNAPVSS VeeamTransportSvc VeeamDeploymentService VeeamNFSSvc AcronisAgent ARSM AcrSch2Svc CASAD2DWebSvc CAARCUupdateSvc WSBExchange MExchange MExchange\$