

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

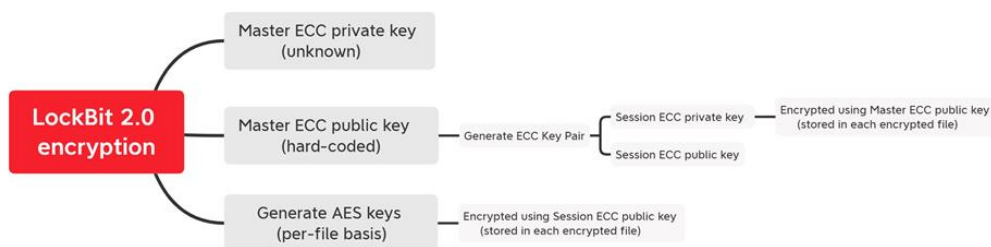
1.212.222.7061

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:

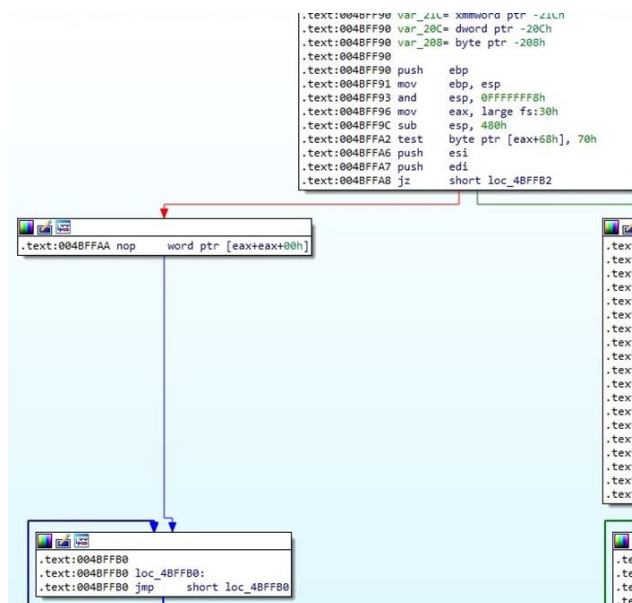


Figure 1

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 → 0048FFB2 C7 84 24 E8 00 00 00 20 00 mov dword ptr ss:[esp+E8],20
0048FFB3 33 F6 xor esi,esi
0048FFB4 C6 84 24 EC 00 00 00 47 mov byte ptr ss:[esp+EC],47
0048FFB5 C6 84 24 ED 00 00 00 44 mov byte ptr ss:[esp+ED],44
0048FFB6 C6 84 24 EE 00 00 00 49 mov byte ptr ss:[esp+EE],49
0048FFB7 C6 84 24 EF 00 00 00 50 mov byte ptr ss:[esp+EF],50
0048FFB8 C6 84 24 F0 00 00 00 4C mov byte ptr ss:[esp+F0],4C
0048FFB9 C6 84 24 F1 00 00 00 55 mov byte ptr ss:[esp+F1],55
0048FFBA C6 84 24 F2 00 00 00 53 mov byte ptr ss:[esp+F2],53
0048FFBB C6 84 24 F3 00 00 00 0E mov byte ptr ss:[esp+F3],E
0048FFBC C6 84 24 F4 00 00 00 44 mov byte ptr ss:[esp+F4],44
0048FFBD C6 84 24 F5 00 00 00 4C mov byte ptr ss:[esp+F5],4C
0048FFBE C6 84 24 F6 00 00 00 4C mov byte ptr ss:[esp+F6],4C
0048FFBF 8A 84 24 EC 00 00 00 00 mov al,byte ptr ss:[esp+EC]
0048FFC0 C6 84 24 F7 00 00 00 00 mov byte ptr ss:[esp+F7],0
0048FFC1 66 66 0F 1F 84 00 00 00 nop word ptr ds:[eax+eax],ax
0048FFC2 8A 94 34 EC 00 00 00 00 mov dl,byte ptr ss:[esp+esi+EC]
0048FFC3 8B 84 24 E8 00 00 00 00 mov eax,dword ptr ss:[esp+E8]
0048FFC4 0F BE C8 movsx ecx,al
0048FFC5 0F BE C2 movsx eax,dl
0048FFC6 33 C8 xor ecx,eax
0048FFC7 8B 8C 34 EC 00 00 00 00 mov byte ptr ss:[esp+esi+EC],cl
0048FFC8 46 inc esi
0048FFC9 83 FE 0B cmp esi,8
0048FFCA 72 DD jb lockbit.4C0030
0048FFCB A1 1C 08 4F 00 00 00 00 mov eax,dword ptr ds:[4F081C]
0048FFCC C6 84 24 F7 00 00 00 00 mov byte ptr ss:[esp+F7],0
0048FFCD 85 C0 test eax,eax
0048FFCE 75 0A jne lockbit.4C006E
0048FFCF E8 A7 19 F5 FF call lockbit.411A10

```

Address	Hex	ASCII
0019F8E0	20 00 00 00 67 64 69 70 6C 75 73 2E 64 6C 6C 00	...gdiplus.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:

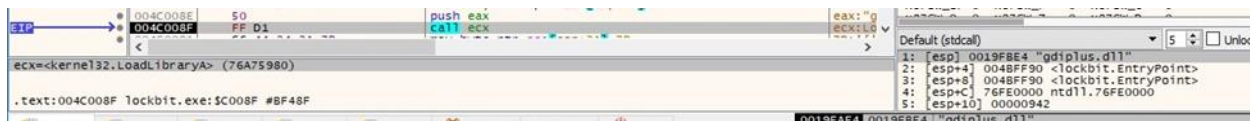


Figure 4

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

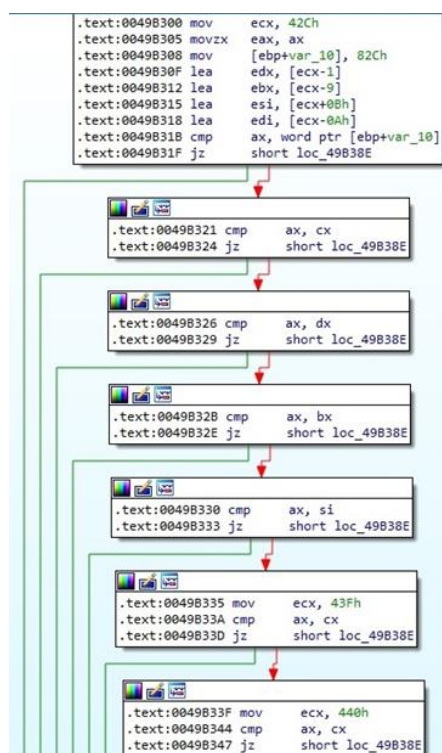


Figure 6

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**):

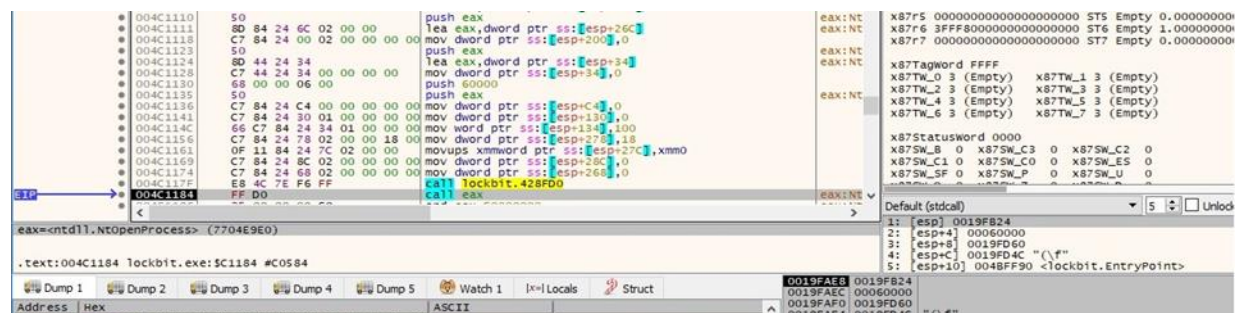
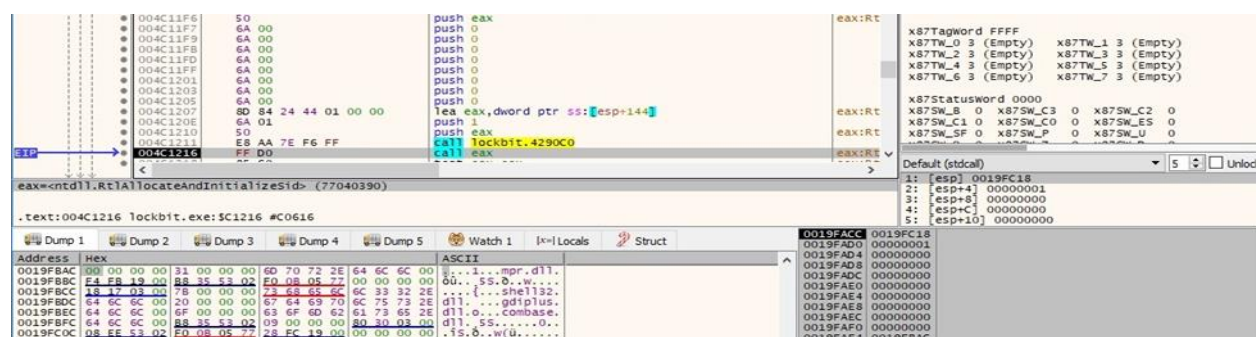


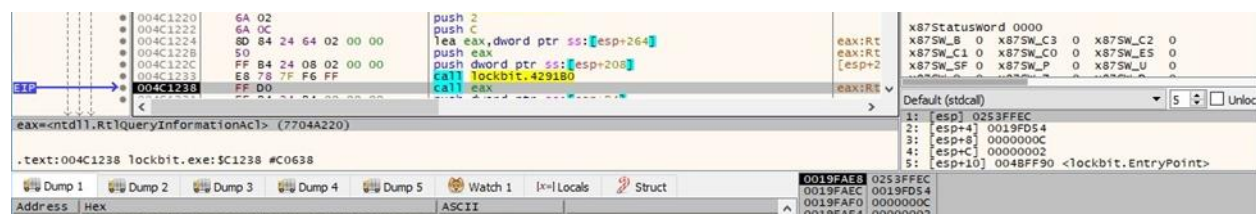
Figure 9

[illegible]

RtlAllocateAndInitializeSid is used to allocate and initialize a SID (security identifier) structure:



The file extracts the ACL size information via a function call to `RtlQueryInformationAcl` (0x2 = **AclSizeInformation**):



The executable allocates memory by calling the `ZwAllocateVirtualMemory` routine (0x3000 = **MEM_COMMIT** | **MEM_RESERVE**, 0x4 = **PAGE_READWRITE**). It's also important to mention that LockBit frees memory previously allocated using `ZwFreeVirtualMemory`:



LIFARS
a SecurityScorecard company

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,MsDsr
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,wdsWfsaf
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	adhip,Culserver,
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,Q8ID
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 4D 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 40 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**):

```

0048B156 50          push     eax
0048B157 6A 08       push     8
0048B159 6A FF       push     0xFFFF
0048B15B 66 C7 45 D8 00 05 mov     word ptr ss:[ebp+28],500
0048B161 FF D0       call     lockbit.411130,500
0048B166 FF D0       call     eax

```

Figure 23

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

```

0048B290 51          push     ecx
0048B29E 6A 4C       push     4C
0048B2A0 8D 40 80    lea     ecx,dword ptr ss:[ebp-80]
0048B2A3 51          push     ecx
0048B2A4 6A 01       push     1
0048B2A6 FF 75 F8    push     dword ptr ss:[ebp-8]
0048B2A9 FF D0       call     eax

```

Figure 24

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

```

0048B418 51          push     ecx
0048B419 6A 00       push     0
0048B41B 6A 00       push     0
0048B41D 6A 00       push     0
0048B41F 6A 00       push     0
0048B421 6A 00       push     0
0048B423 6A 00       push     0
0048B425 6A 00       push     0
0048B427 6A 12       push     12
0048B429 6A 01       push     1
0048B42B 8D 4D D4    lea     ecx,dword ptr ss:[ebp-2C]
0048B42F FF D0       call     eax

```

Figure 25

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

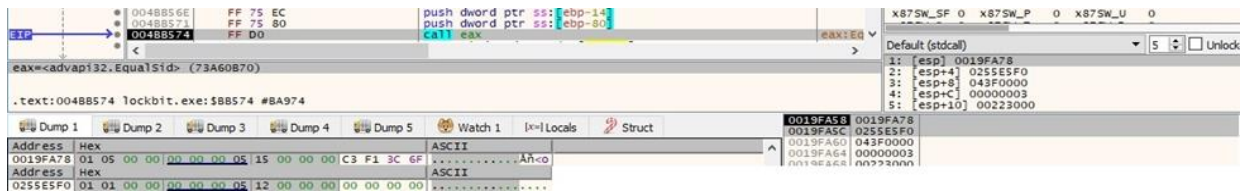


Figure 26

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**):

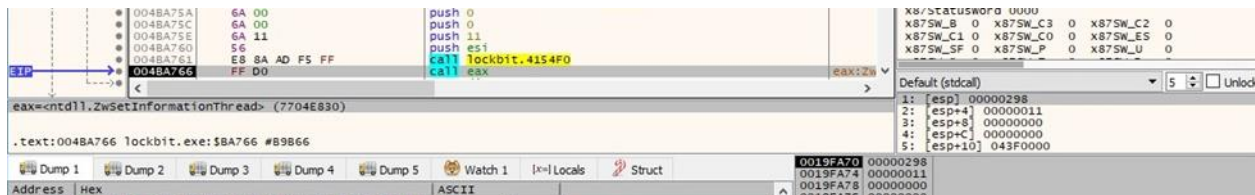


Figure 29

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:

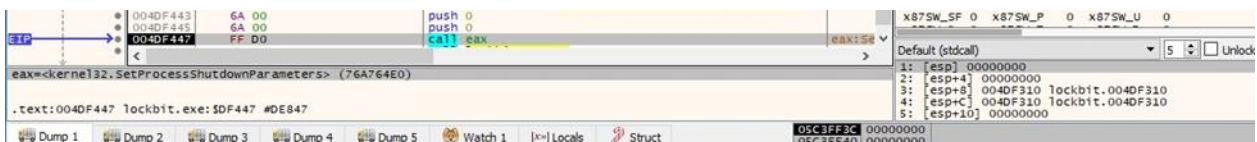


Figure 30

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:

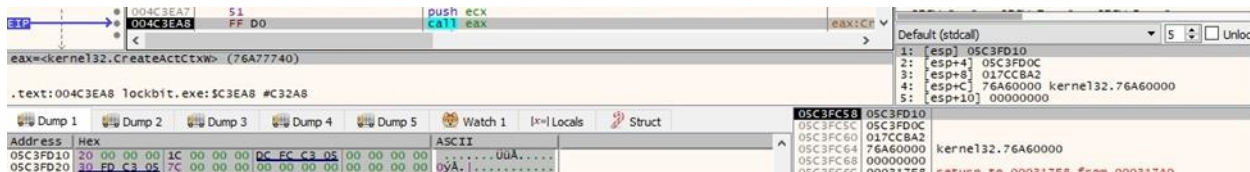


Figure 32



Figure 33

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

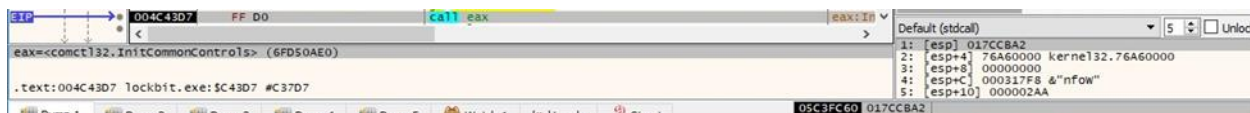


Figure 34

GdiplusStartup is used to initialize Windows GDI+:



Figure 35

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

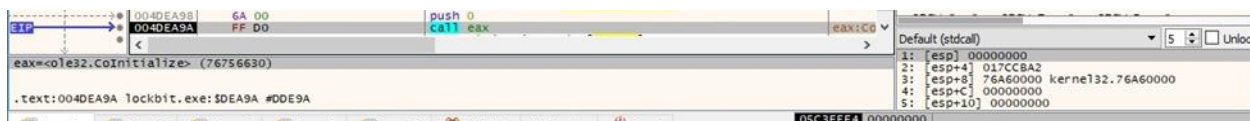


Figure 36

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

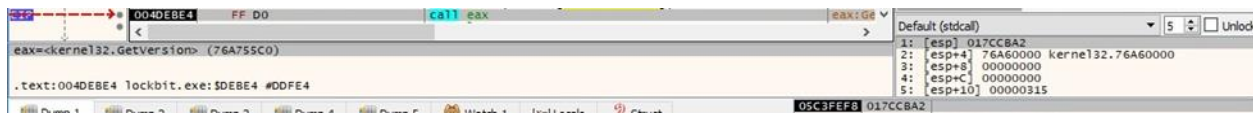


Figure 37

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

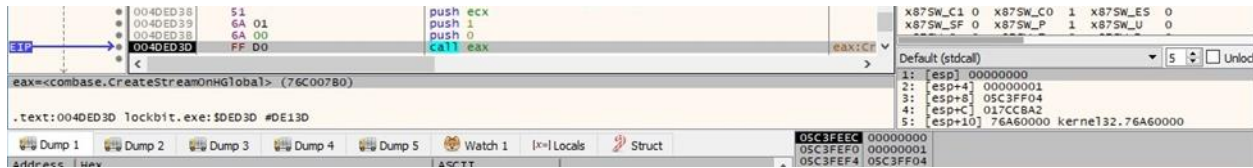


Figure 38

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

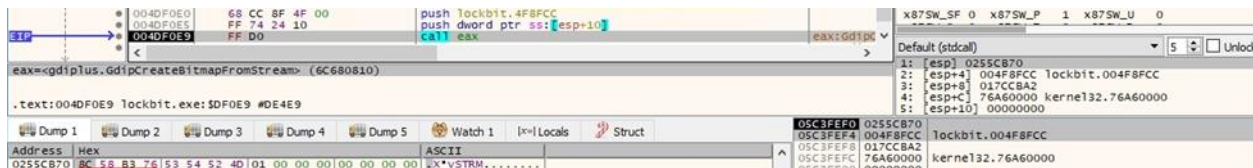


Figure 39

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 registers a window class called "LockBit_2.0_Ransom" using the RegisterClassExW API:

Figure 43

CreateWindowExW is used to create a window called "LockBit 2.0 Ransom" that will track the progress of the ransomware, such as the identified drives and different logs:

Figure 44

The new window is hidden using the ShowWindow routine (0x0 = **SW_HIDE**):

Figure 45

The UpdateWindow function is utilized to update the client area of the specified window by sending a WM_PAINT message to the window:

Figure 46

The process creates a new thread by calling the CreateThread function:

Figure 47

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**):

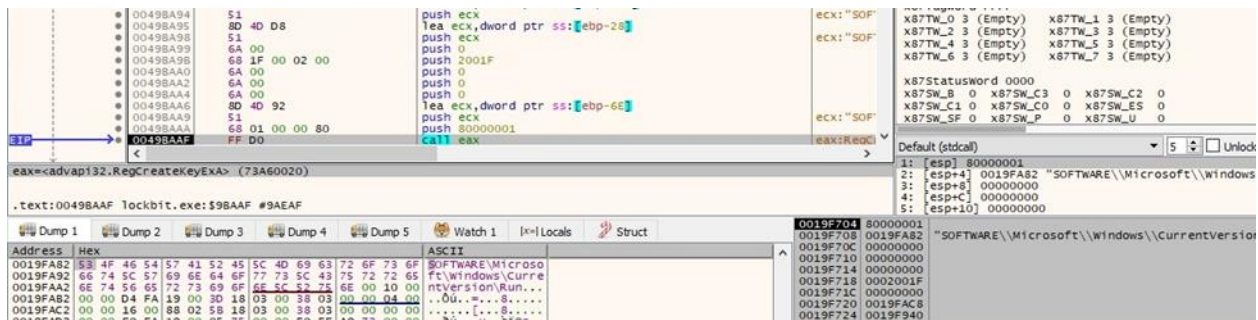


Figure 61

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:

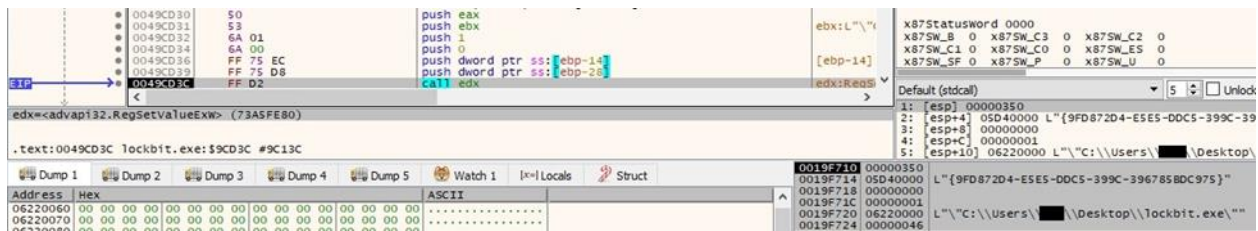


Figure 63



Figure 64

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

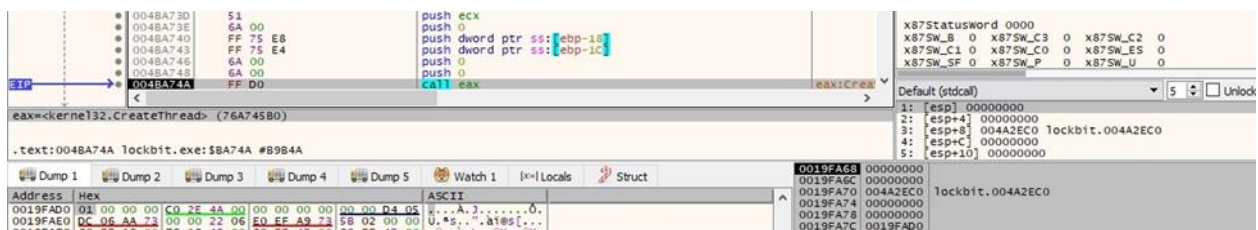


Figure 65

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

The screenshot shows a debugger window with the following sections:

- Assembly View:** Displays assembly instructions with their addresses and hex values. The instruction at address 004C947 is highlighted, showing a lockbit operation. The instruction at 004C948 is also visible.
- Registers:** Shows the state of various registers, including EAX, ECX, EDI, and ESI. The EAX register is highlighted, showing its value as 00000000.
- Memory Dump:** Shows a dump of memory starting at address 0019F348. The dump includes hex values and their corresponding ASCII representations.
- Registers and Memory:** A section showing the state of registers and memory, including the EAX register and the memory address 0019F348.

Thread activity – sub_4A2EC0 function

The screenshot displays the Immunity Debugger interface with three main panels:

- Assembly Window:** Shows assembly instructions for a function named `FindFirstVolumeW`. The instructions are:
 - `push 104`
 - `lea ecx, dword ptr SS:[esp+16C]`
 - `push ecx`
 - `call eax`
- Registers Window:** Shows the state of CPU registers. The `EAX` register is highlighted, containing the address `kernel32.FindFirstVolumeW` (76ACDE0).
- Stack Window:** Shows the current stack frame. The stack pointer `EIP` is at `004A3017`. The stack contains several entries, including `004A3016` and `004A3015`.

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

The screenshot displays the Immunity Debugger interface with the following components:

- Registers Panel:**
 - EAX:** 004A31D9, 68 04 01 00, 1ea ecx, dword ptr ss:[esp+37h]
 - ECX:** 004A31DE, 51, push ecx
 - EDX:** 004A31EE, 80 8C 24 78 01 00 00, 1ea ecx, dword ptr ss:[esp+17h]
 - ESI:** 004A31F6, 51, push ecx
 - EDI:** 004A31FD, 51, push ecx
 - EIP:** 004A31EE, FF D0, call eax
- Disassembly Panel:**
 - Address: 004A31EE, Instruction: FF D0, Comment: call eax
- Stack Panel:**
 - Address: 004A31EE, Instruction: FF D0, Comment: call eax
- Memory Dump Panel:**
 - Address: 004A31EE, Instruction: FF D0, Comment: call eax
- Registers Panel (Right):**
 - X87SW0:** STATUSW0F UUUU
 - X87SW_B:** 0 0 X87SW_C3 0 X87SW_C2 0
 - X87SW_C1:** 0 0 X87SW_C0 0 X87SW_E5 0
 - X87SW_SF:** 0 X87SW_P 0 X87SW_U 0
- Default (stdcall) Panel:**
 - 1: [esp] 0643F168 L"Volume{d7e47829-0000-0000-0000-000000000000}
 - 2: [esp+4] 0643F368
 - 3: [esp+8] 000000104
 - 4: [esp+C] 004A2EC0 lockbit, 004A2EC0
 - 5: [esp+10] 004A2EC0 lockbit, 004A2EC0
- Registers Panel (Bottom):**
 - EAX:** 004A31EE, Instruction: FF D0, Comment: call eax
- Memory Dump Panel (Bottom):**
 - Address: 004A31EE, Instruction: FF D0, Comment: call eax

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

The screenshot displays the Immunity Debugger interface with several panes:

- Registers:** Shows CPU registers like EAX, ECX, EDI, etc. EAX contains 004A337D.
- Disassembly:** Shows assembly instructions:


```

push ecx
push dword ptr ss:[esp+24]
lea ecx,dword ptr ss:[esp+170]
push ebx
push ecx
call eax

```
- Stack:** Shows stack frames. The current frame is at 004A337D. The stack pointer (ESP) is 0643F160.


```

1: [esp] 0643F160 L"\\?\\Volume{d7e47829-0000-0000-0000-0000-0000}
2: [esp+4] 06220000
3: [esp+8] 00000209
4: [esp+C] 0643F018
5: [esp+10] 004A2EC0 lockbit.004A2EC0

```
- Memory Dump:** Shows a hex dump of memory at address 0643F160. The data is:


```

0643F160 5C 00 5C 00 3F 00 5C 00 56 00 6F 00 6C 00 75 00
0643F170 6D 00 65 00 78 00 64 00 37 00 65 00 34 00 37 00
0643F180 38 00 32 00 39 00 30 00 30 00 30 00 30 00 30 00
0643F190 2D 00 20 00 30 00 30 00 2D 00 20 00 30 00 30 00
0643F1A0 30 00 30 00 20 00 31 00 30 00 30 00 30 00 30 00
0643F1B0 30 00 30 00 30 00 30 00 30 00 30 00 7D 00 30 00
0643F1C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

LIFARS
a SecurityScorecard company

The screenshot shows the Immunity Debugger interface. The assembly window displays the following code:

```

004A358B 51 FF D0          push ecx
004A358C          call eax

```

The comment for the first instruction is: `eax=<kernel32.GetDriveType> (76ACDFA0)`. The comment for the second instruction is: `.text:004A358C lockbit.exe:$A358C #A298C`.

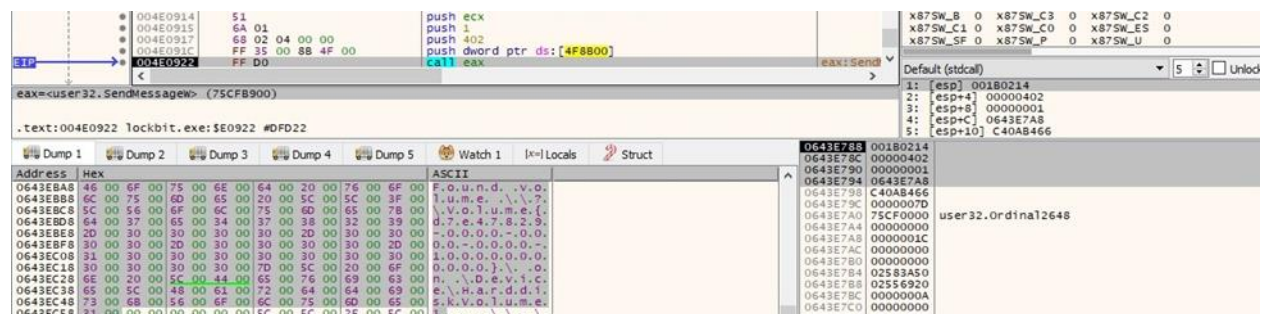
The stack window shows the following dump:

```

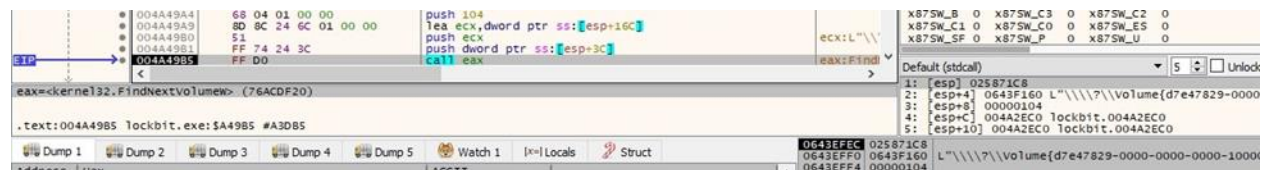
0043EEFA 0043F160 L"\\\\?\\Volume{d7e47829-0000-0000-0000-0000-0000-0000-0000-0000}
0043EEFB 004A2E00 lockbit.004A2E00
0043EEFC 004A2E00 lockbit.004A2E00
0043EEFD 00000000
0043EEFE 00000000

```

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



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



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

[illegible]

An unmounted volume is mounted by calling the SetVolumeMountPointW routine:

```

004A4198 51          push ecx
004A419C 8D 8C 24 40 01 00 00 lea ecx,dword ptr ss:[esp+140]
004A41A3 61          push ecx
004A41A4 FF D0      call eax

```

eax=kernel32.SetVolumeMountPointw> (76AB2200)

.text:004A41A4 lockbit.exe:\$A41A4 #A35A4

Default (stdcall)

1: [esp] 0643F134 L"Z:"

2: [esp+4] 0643F160 L"\\\\?\\volume{d7e47829-0000-0000-0000-0000}00000000"

3: [esp+8] 004A2EC0 lockbit.004A2EC0

4: [esp+C] 004A2EC0 lockbit.004A2EC0

5: [esp+10] 00000000

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.

```

004E0914 51          push ecx
004E0915 6A 01      push 1
004E0917 68 02 04 00 00 push dword ptr ds:[4F8B00]
004E0922 FF D0      call eax

```

eax=user32.SendMessageW> (75CFB900)

.text:004E0922 lockbit.exe:\$E0922 #DFD22

Default (stdcall)

1: [esp] 00180214

2: [esp+4] 00000402

3: [esp+8] 00000001

4: [esp+C] 0643E7A8

5: [esp+10] 62518C49

0643E788 00180214

0643E78C 00000402

0643E790 00000001

0643E794 0643E7A8

0643E798 62518C49

0643E79C 76AECCE1

0643E7A0 75CFB900

0643E7A4 00000000

0643E7A8 00000000

0643E7AC 00000002

0643E7B0 7706201D

0643E7B4 0643E7A8

0643E7B8 0643E7A8

0643E7BC 0643E7A8

return to ntdll.7706201D from ntdll.770508A0

lockbit.0043E7A8

Figure 80

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

```

004ACAE4 6A 00      push 0
004ACAE6 6A 00      push 0
004ACAE8 6A 00      push 0
004ACAE9 68 00 00 00 08 push dword ptr ds:[00000008]
004ACAEF FF D0      call eax

```

eax=shell32.SHChangeNotify> (743BA9C0)

.text:004ACAEF lockbit.exe:\$ACAEF #ABEEF

Default (stdcall)

1: [esp] 08000000

2: [esp+4] 00000000

3: [esp+8] 00000000

4: [esp+C] 00000000

5: [esp+10] 043F0000

0019F350 08000000

0019F354 00000000

0019F358 00000000

0019F35C 00000000

Figure 81

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

```

004BA73D 51          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=kernel32.CreateThread> (76A745B0)

.text:004BA74A lockbit.exe:\$BA74A #B9B4A

Default (stdcall)

1: [esp] 00000000

2: [esp+4] 00000000

3: [esp+8] 00497060 lockbit.00497060

4: [esp+C] 00000000

5: [esp+10] 00000000

0019FA68 00000000

0019FA6C 00000000

0019FA70 00497060

0019FA74 00000000

0019FA78 00000000

0019FA7C 0019FA70

lockbit.00497060

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:

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**):

Figure 84

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

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	eeqjofwgcglmutr3
0019F853	61 33 35 6E 79 67 76 6F 68 6A 61 35 75 75 63 63	a35nygvokja5uucc
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 6F 6E 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):

004A04D3	51	push ecx	ecx:L"SOI	x87TW_0 3 (Empty)	x87TW_1 3 (Empty)
004A04D4	8D 4D C4	lea ecx,dword ptr ss:[ebp-3C]	ecx:L"SOI	x87TW_2 3 (Empty)	x87TW_3 3 (Empty)
004A04D7	51	push ecx		x87TW_4 3 (Empty)	x87TW_5 3 (Empty)
004A04D8	6A 00	push 0		x87TW_6 3 (Empty)	x87TW_7 3 (Empty)
004A04DA	68 3F 00 0F 00	push F003F			
004A04DF	6A 00	push 0			
004A04E1	6A 00	push 0			
004A04E3	6A 00	push 0			
004A04E5	8D 80 90 FA FF FF	lea ecx,dword ptr ss:[ebp-570]			
004A04EB	51	push ecx	ecx:L"SOI		
004A04EC	68 01 00 00 80	push 80000001			
004A04F1	FF D0	call eax	eax:REGQ		

eax=cadvap132.RegCreateKeyExW (73A5FE50)		.text:004A04F1 lockbit.exe:SA04F1 #9F8F1	
Address	Hex	ASCII	
0019F580	53 00 4F 00	46 00 54 00	52 00 41 00
0019F590	5C 00 32 00	45 00 44 00	38 00 37 00
0019F5A0	34 00 45 00	35 00 33 00	38 00 39 00
0019F5B0	63 00 6F 00	00 00 19 00	50 6A 05 77
0019F5C0	FE FF FF FF	80 FE 19 00	EA A3 07 76

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:

004A058F	50	push eax	eax:"Pri	x87TW_4 3 (Empty)	x87TW_5 3 (Empty)
004A0593	FF 35 90 BC 4F 00	push dword ptr ds:[4FBC90]		x87TW_6 3 (Empty)	x87TW_7 3 (Empty)
004A0596	8D 85 C8 FC FF FF	lea eax,dword ptr ss:[ebp-338]			
004A059C	50	push eax	eax:"Pri		
004A059D	50	push 0			
004A059F	8D 45 BC	lea eax,dword ptr ss:[ebp-44]			
004A05A2	50	push eax	eax:"Pri		
004A05A3	FF 75 C4	push dword ptr ss:[ebp-3C]			
004A05A6	FF D1	call ecx	ecx:REQQ		

ecx=cadvap132.RegQueryValueExA (73A5FA00)		.text:004A05A6 lockbit.exe:SA05A6 #9FA6	
Address	Hex	ASCII	
0019F788	03 00 00 00	70 00 00 00	3C 00 3E 42
0019F7C8	69 74 20 32	2E 30 20 52	61 6E 73 6F

Figure 88

004A0635	50	push eax	eax:&0	x87TW_6 3 (Empty)	x87TW_7 3 (Empty)
004A0636	56	push esi			
004A0637	8D 85 C8 FC FF FF	lea eax,dword ptr ss:[ebp-338]			
004A063D	50	push eax	eax:&0		
004A063E	6A 00	push 0			
004A0640	8D 45 B0	lea eax,dword ptr ss:[ebp-50]			
004A0643	50	push eax	[ebp-50]		
004A0644	FF 75 C4	push dword ptr ss:[ebp-3C]	eax:&0		
004A0647	FF D1	call ecx	ecx:REQQ		

ecx=cadvap132.RegQueryValueExA (73A5FA00)		.text:004A0647 lockbit.exe:SA0647 #9FA47	
Address	Hex	ASCII	
0019FA00	58 75 62 6C	69 63 00 00	10 00 00 00
0019FA0B	61 74 65 00	8C 03 00 00	00 00 FE 7E

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:

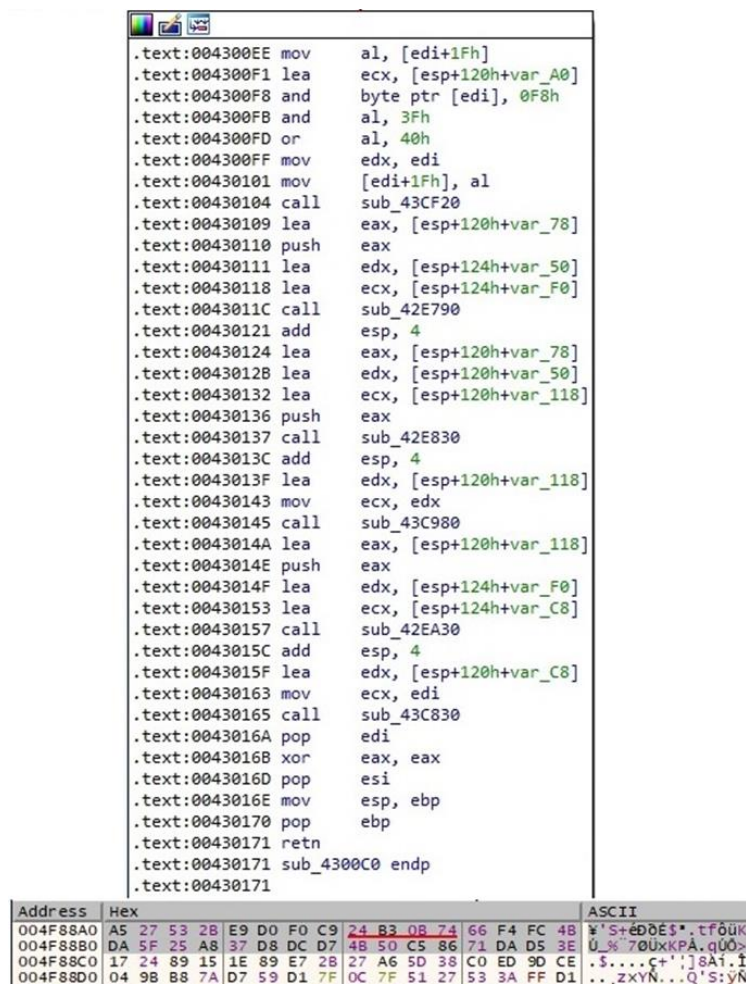


Figure 91

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.® 10SV..
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):

0042E6E6	57	push edi	x87SW_SF 0 x87SW_P 0 x87SW_U 0
0042E6E7	56	push esi	
0042E6E8	FF 15 90 06 4F 00	call dword ptr ds:[4F0690]	
dword ptr [lockbit.004F0690]=lockbit.004312F0			
.text:0042E6E8 lockbit.exe:\$2E6E8 #2DAE8			
Address	Hex	ASCII	
0019F320	73 9C 00 80 3C E1 E2 91 A9 AF EF DA 53 76 8D 11	S...<aa.® 10SV..	
0019F330	0B 35 23 13 26 0A 51 17 43 1F 50 DC CA 42 12 7D	.5#.&.Q.C.PUEB.}	
0019F340	2E D8 73 D4 3F E5 D0 38 39 9C 88 67 72 8D C9 75	.0s074y89..grHEU	
0019F350	50 9F 6C CE D6 E8 C5 85 02 0C F7 F3 0E 91 D6 61	.110eA...0..08	
0019F218	0019F2C0		
0019F21C	0019F320		
0019F220	0019F518		
0019F224	0019F530		
0019F228	0019F49C		
0019F22C	000000E0		
0019F230	0042E61C	return to lockbit.0042E61C from lockbit.0040	

Figure 94

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

0042ACEE	57	push edi	x87TW_6 3 (Empty) x87TW_7 3 (Empty)
0042ACF0	8D 45 9C	lea eax,dword ptr ss:[ebp-64]	x87StatusWord 0000
0042ACF2	C6 45 FB 00	mov byte ptr ss:[ebp-5],0	x87SW_L 0 x87SW_C3 0 x87SW_C2 0
0042ACF6	50	push eax	x87SW_L1 0 x87SW_C0 0 x87SW_ES 0
0042ACF7	8D 85 7C FF FF FF	lea eax,dword ptr ss:[ebp-84]	x87SW_SF 0 x87SW_P 0 x87SW_U 0
0042ACFD	50	push eax	
0042ACFE	A1 70 06 4F 00	mov eax,dword ptr ds:[4F0670]	
0042AD03	8B 00	mov eax,dword ptr ds:[eax]	
0042AD05	FF D0	call eax	
eax=lockbit.0042F6E0			
.text:0042AD05 lockbit.exe:\$2AD05 #2A105			
Address	Hex	ASCII	
0019F4F8	A1 ED 74 FE A3 E1 1E 77 33 03 F1 83 CE 90 16 D3	ittpEa.w3.ñ.1..0	
0019F508	AC 0B 90 7E A3 97 E7 13 CA E2 04 60 48 36 4A 26	...~f.ç.Êa. H636	
Address	Hex	ASCII	
0042E200	2E D8 73 D4 3F E5 D0 38 39 9C 88 67 72 8D C9 75	.0s074y89..grHEU	
0042E210	50 9F 6C CE D6 E8 C5 85 02 0C F7 F3 0E 91 D6 61	.110eA...0..08	
0019F49C	0019F4D8		
0019F4A0	0019F4F8		
0019F4A4	004E2500	lockbit.004E2500	
0019F4A8	00000000		
0019F4AC	00000019		

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:

```

004A0A48 6A 70          push 70
004A0A4A FF 35 8C 4F 00 push dword ptr ds:[4F8C90]
004A0A50 8D 45 CC       lea eax,dword ptr ss:[ebp-34]
004A0A53 6A 03          push 3
004A0A55 6A 00          push 0
004A0A57 50            push eax
004A0A58 FF 75 C4       push dword ptr ss:[ebp-3C]
004A0A5B FF D1         call ecx

```

Default (stdcall) 5 Unload

1: [esp] 0000038C
2: [esp+4] 0019FABC "Private"
3: [esp+8] 00000000
4: [esp+C] 00000003
5: [esp+10] 06450000

Figure 97

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

```

004A0A48 6A 20          push 20
004A0A4A 68 A0 88 4F 00 push lockbit.4F88A0
004A0A50 6A 03          push 3
004A0A53 6A 00          push 0
004A0A55 6A 00          push 0
004A0A57 50            push eax
004A0A58 FF 75 C4       push dword ptr ss:[ebp-3C]
004A0A5B FF D1         call ecx

```

Default (stdcall) 5 Unload

1: [esp] 0000038C
2: [esp+4] 0019FA94 &"o setlinejoin\r\n"
3: [esp+8] 00000000
4: [esp+C] 00000003
5: [esp+10] 004F88A0 lockbit.004F88A0

Figure 98

Figure 99 reveals both registry values with their content:

Name	Type	Data
(Default)	REG_SZ	(value not set)
Private	REG_BINARY	24 6c f9 d0 d9 0a ee e1 cf 52 65 a0 32 a5 c8 97 ec fc 0c e3 39 ee ab 0c 6f 81 29 cc 7f 71 51 4b 9c b6 27 e5 ...
Public	REG_BINARY	a3 27 53 2b e9 d0 10 c9 24 b3 0b 74 66 14 fc 4b da 5f 25 a3 37 d8 dc d7 4b 50 c5 b6 71 da d5 3e

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**):

```

004A0D02 50            push eax
004A0D03 6A 00          push 0
004A0D05 68 03 00 1F 00 push 1F0003
004A0D0A 68 20 25 4E 00 push lockbit.4E2520
004A0D0F A3 98 88 4F 00 mov dword ptr ds:[4F8898],eax
004A0D14 68 E7 46 F7 FF call lockbit.415400
004A0D19 FF D0         call ecx

```

Default (stdcall) 5 Unload

1: [esp] 004E2520 lockbit.004E2520
2: [esp+4] 001F0003
3: [esp+8] 00000000
4: [esp+C] 00000002
5: [esp+10] 043F0000

Figure 100

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

Figure 101

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

Figure 102

GetLogicalDrives is used to retrieve the available disk drives:

Figure 103

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

Figure 104

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

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 115

File extensions are extracted using the PathFindExtensionW routine:



Figure 116

The binary is looking for a ".lockbit" file that would suggest the targeted file has already been encrypted:

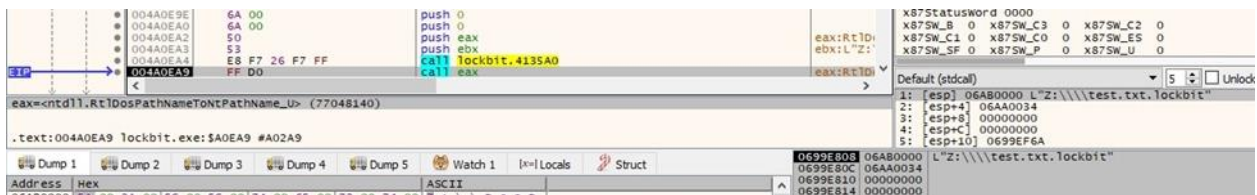


Figure 117

ZwCreateFile is utilized to open the targeted file (0x10003 = **FILE_READ_DATA | FILE_WRITE_DATA | DELETE**, 0x80 = **FILE_ATTRIBUTE_NORMAL**, 0x1 = **FILE_OPEN**, 0x48 = **FILE_NON_DIRECTORY_FILE | FILE_NO_INTERMEDIATE_BUFFERING**):

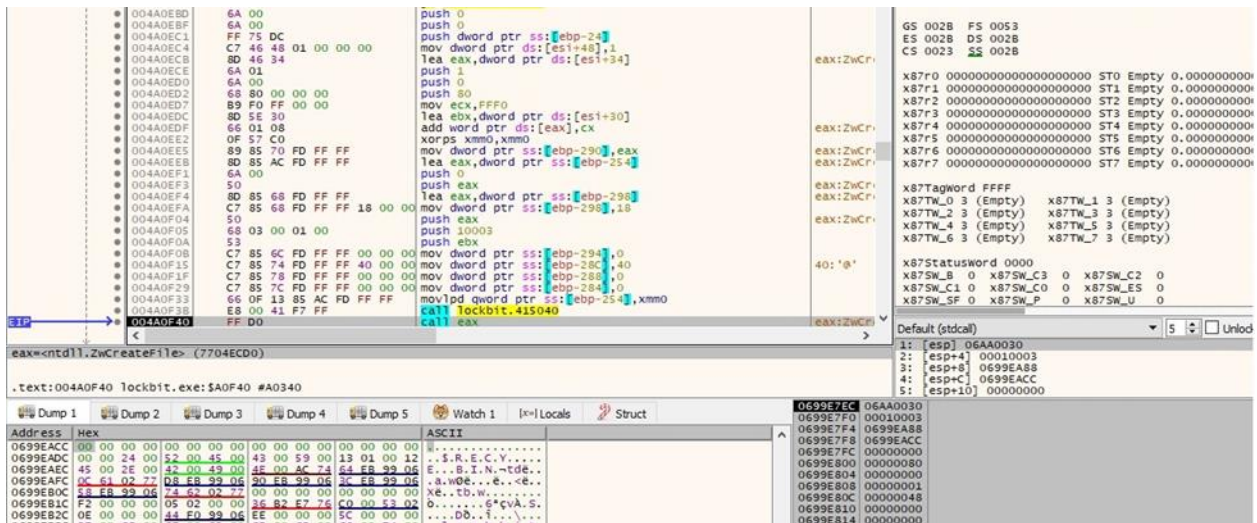


Figure 118

The targeted file is bound to the I/O completion port created earlier via a function call to NtSetInformationFile (0x1E = **FileCompletionInformation**):

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:

```

004A2AB8 6A 00          push 0
004A2ABA C7 46 2C 00 00 00 00 mov dword ptr ds:[esi+2C],0
004A2AC1 51           push ecx
004A2AC2 89 42 2C     mov dword ptr ds:[edx+2C],eax
004A2AC5 A1 78 8C 4F 00 mov eax,dword ptr ds:[4F8C78]
004A2ACA 89 42 28     mov dword ptr ds:[edx+28],eax
004A2ACD C7 02 03 00 00 00 00 mov dword ptr ds:[edx],3
004A2AD3 C7 01 00 00 00 00 00 mov dword ptr ds:[ecx],0
004A2AD9 C7 41 04 00 00 00 00 mov dword ptr ds:[ecx+4],0
004A2AE0 FF 72 2C     push dword ptr ds:[edx+2C]
004A2AE3 50           push eax
004A2AE4 8D 42 04     lea eax,dword ptr ds:[edx+4]
004A2AE7 50           push eax
004A2AE8 52           push edx
004A2AE9 6A 00       push 0
004A2AEB 6A 00       push 0
004A2AED FF 33       push word ptr ds:[ebx]
004A2AEF E8 1C 26 F7 FF call lockbit.415310
004A2AF4 FF D0       call eax
  
```

Figure 126

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

```

0045E60F 51           push ecx
0045E610 FF D0       call eax
  
```

Figure 127

The content of the ransom note is displayed below:

```

1 LockBit 2.0 Ransomware
2
3 Your data are stolen and encrypted
4 The data will be published on TOR website http://lockbitapt6vx57c3eeqjofwgoglmtr3a35nygvokja5uuoocip4kykd.onion and https://bigblog.at if you do not pay the ransom
5 You can contact us and decrypt one file for free on these TOR sites
6 http://lockbitsup4yezcd5enk5unnx3zcy7kw6willygmihvanj352jayid.onion
7 http://lockbitsap2oaqhoun3syvbt6n5nrt7fqsc6jdlmsfleu3ka9k2did.onion
8 OR
9 https://decoding.at
10
11 Decryption ID: 2ED873D43FE5DD38A527532BE9D0F0C9
  
```

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:

```

0048ECD1 51           push ecx
0048ECD2 8D 4D BF     lea ecx,dword ptr ss:[ebp-41]
0048ECD5 51           push ecx
0048ECD6 6A 00       push 0
0048ECD8 FF D0       call eax
  
```

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:

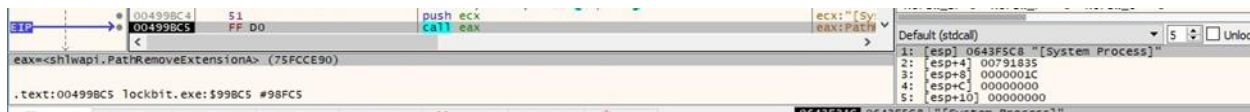


Figure 139

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_OLEIIDDE**):

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.\$uC.

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:

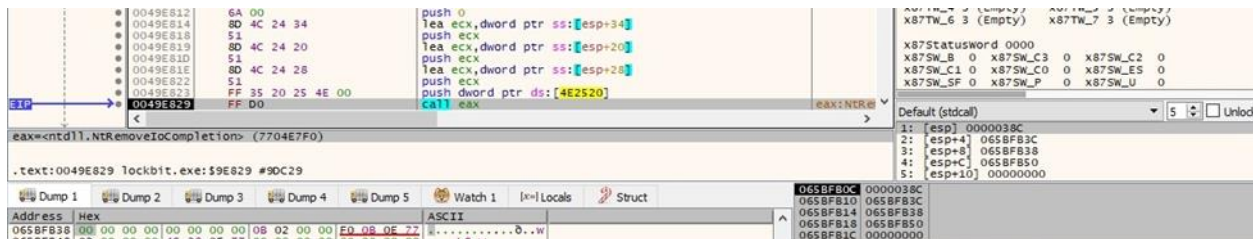


Figure 149

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 .wpv .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:0043D9E8 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	%wC./6Ac;0k..7U
0658FE50	4D ED F3 1F 62 19 51 DF 01 22 A2 84 84 35 95 48	Mi0.b.QB."c'.5.H
0658FE60	D9 C7 A1 40 8B DE F0 9F BA FC 52 28 3E C9 C7 63	Ucj@»pD.%UR+>EcC
0658FE70	00 01 5A F2 BB DF AA 6D 01 23 F8 46 3F EA 3F 25	..Z0»8*m.#0F?e?%
0658FE80	8F 74 65 87 34 AB CF EA 35 88 37 AC 0A 62 08 89	.te.4«Ie5.7~.b..
0658FE90	35 44 C2 E0 01 EF 0D 0A 34 67 3A A6 3E 05 32 2F	SDAa.T..4g:/.>.2/
0658FEA0	7E 67 D7 52 7F 88 DA 58 48 EF E0 FE 75 EA D2 D1	~gxR..UXK1apu0N
0658FEB0	89 D2 E9 CF C6 5A 33 97 8D B5 D3 69 F8 5F 01 88	"0eIAZ3...p010..
0658FEC0	F6 AE 85 8E 30 F4 B6 19 BD 41 65 70 45 1E 64 C8	09...00%.%AepE.dE
0658FED0	9F ED 6D E0 AF 19 DB F9 12 58 BE 89 57 46 DA 41	.ima".0U.X%.WFOA
0658FEE0	F3 BA EE 88 5C A3 35 42 4E FB 8B CB 19 BD 51 8A	0~i»\£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 aesenc  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
06A80000	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80010	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80020	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80030	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80040	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80050	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80060	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80070	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80080	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80090	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A800A0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A800B0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A800C0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A800D0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A800E0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A800F0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80100	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80110	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80120	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
06A80130	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA

Figure 153

Address	Hex	ASCII
06A80000	C8 89 88 60 3E F8 64 12 B3 F1 7E D7 2A 1A 8A EE	E...>od.*h~x*..i
06A80010	6C 08 52 F5 6D FD 6A 3A 80 EF D5 99 C1 D7 24 F2	l.R6myj:~t0.Ax5b
06A80020	72 3E B1 2C 60 EA 4C 71 A1 16 D8 FF 85 A0 29 67	r>±,eLqi,0y, .ig
06A80030	09 A0 97 A2 C1 1E E0 8A 5D 47 9D 6E 5E E5 C5 76	.>.eA,a,]G,n^0Av
06A80040	94 64 92 12 FE D0 7F F2 A2 82 87 6F 03 D2 C8 A2	.d, .pD,0s*~o,0E±
06A80050	80 56 22 43 D3 D6 F6 85 3D 38 8C 14 31 A1 E1 E3	.v^CO00u~,.1iaa
06A80060	75 66 92 5A E0 68 95 C0 B6 00 94 93 D4 4C 7D B0	uf,Zak,Aq...0Lj'
06A80070	C1 CD 8F 13 52 E8 A3 F2 05 70 61 2D 35 00 BD 8D	Ai..Reib.pa-5.%s
06A80080	30 03 88 8E C8 06 D0 BC 99 E0 31 26 D9 82 08 88	O...E.D%.a1&U...
06A80090	39 E9 4C 53 A4 5F 43 8D BF FF 05 9B 41 72 E3 BE	9eLS~C.0Y..Ar&±
06A800A0	4C 4E 1D AA 0E 75 43 A1 0D AF 12 DF D2 2A 25 70	L'.±.uCi..B0~Sp
06A800B0	29 CA EE DF F5 9F D0 17 A1 57 01 C5 2E F4 D7 C5)E1B6.D.iW.A.0xA
06A800C0	06 50 B5 43 59 C8 44 A8 86 E6 00 A4 AA 31 93 C8	.PµCYED±.±.R±1.E
06A800D0	19 2F 71 B1 43 D1 1A 41 17 DA 6D B1 D0 9B F1 8F	./q±CN.A.Um±Y.h.
06A800E0	E5 4E 35 E7 53 2E 7C 71 20 7F A7 69 CA 55 79 BE	±N5CS.iq.±iEuy%
06A800F0	91 6C 28 88 EA E6 3E 13 C7 0D 22 95 AC 2D 9F F3	.1(.e±>.C."~-.0
06A80100	E9 47 BC 77 70 8D DE C9 20 3F 80 44 0F B8 69 C2	eG4wP.DE ?..D..iA
06A80110	64 38 44 02 D0 F8 F6 AF C4 C5 38 56 BC 99 31 AD	d8D.D00.AA8V%.1.
06A80120	9A 3A 40 0C 66 68 AF 58 CC A9 9E 36 D9 82 87 C4	±.0.fk'±i0.6U..A
06A80130	24 6E E0 A7 FC F2 0E D3 98 92 65 B2 33 2D 44 5E	\$na§ub.0...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:

0049EBD2	6A 00	push 0			
0049EBD4	89 74 24 18	mov dword ptr ss:[esp+18],esi			
0049EBD8	8D 41 20	lea eax,dword ptr ds:[ecx+20]	eax:ZwWri	x87tagword FFFF	
0049EBDC	50	push eax	eax:ZwWri	x87TW_0 3 (Empty)	x87TW_1 3 (Empty)
0049EBE0	71 2C	push dword ptr ds:[ecx+2C]		x87TW_2 3 (Empty)	x87TW_3 3 (Empty)
0049EBE4	8D 41 04	lea eax,dword ptr ds:[ecx+4]	eax:ZwWri	x87TW_4 3 (Empty)	x87TW_5 3 (Empty)
0049EBE8	FF 71 28	push dword ptr ds:[ecx+28]		x87TW_6 3 (Empty)	x87TW_7 3 (Empty)
0049EBEC	50	push eax	eax:ZwWri		
0049EBF0	51	push ecx		x87StatusWord 0000	
0049EBF4	6A 00	push 0		x87SW_8 0	x87SW_C3 0
0049EBF8	6A 00	push 0		x87SW_8 0	x87SW_C3 0
0049EBFC	76 30	push dword ptr ds:[esi+30]		x87SW_C1 0	x87SW_C0 0
0049EB00	E8 1D 67 F7 FF	call lockbit.415310		x87SW_SF 0	x87SW_P 0
0049EB04	FF D0	call eax	eax:ZwWri		

eax=<ntdll.ZwWriteFile> (7704E7E0)		Default (stdcal)	
.text:0049EBF3 lockbit.exe:\$9EBF3 #90FF3		1: [esp] 000003A8	
		2: [esp+4] 00000000	
		3: [esp+8] 00000000	
		4: [esp+C] 06AA0050	
		5: [esp+10] 06AA0054	

Dump 1		Dump 2		Dump 3		Dump 4		Dump 5		Watch 1		[x] Locals		Struct	
Address	Hex	Address	Hex	Address	Hex	Address	Hex	Address	Hex	Address	Hex	Address	Hex	Address	Hex
06A80000	C8 89 88 60 3E F8 64 12 B3 F1 7E D7 2A 1A 8A EE	06A80000	C8 89 88 60 3E F8 64 12 B3 F1 7E D7 2A 1A 8A EE	06A80000	C8 89 88 60 3E F8 64 12 B3 F1 7E D7 2A 1A 8A EE	06A80000	C8 89 88 60 3E F8 64 12 B3 F1 7E D7 2A 1A 8A EE	06A80000	C8 89 88 60 3E F8 64 12 B3 F1 7E D7 2A 1A 8A EE	06A80000	C8 89 88 60 3E F8 64 12 B3 F1 7E D7 2A 1A 8A EE	06A80000	C8 89 88 60 3E F8 64 12 B3 F1 7E D7 2A 1A 8A EE	06A80000	C8 89 88 60 3E F8 64 12 B3 F1 7E D7 2A 1A 8A EE
06A80010	6C 08 52 F5 6D FD 6A 3A 80 EF D5 99 C1 D7 24 F2	06A80010	6C 08 52 F5 6D FD 6A 3A 80 EF D5 99 C1 D7 24 F2	06A80010	6C 08 52 F5 6D FD 6A 3A 80 EF D5 99 C1 D7 24 F2	06A80010	6C 08 52 F5 6D FD 6A 3A 80 EF D5 99 C1 D7 24 F2	06A80010	6C 08 52 F5 6D FD 6A 3A 80 EF D5 99 C1 D7 24 F2	06A80010	6C 08 52 F5 6D FD 6A 3A 80 EF D5 99 C1 D7 24 F2	06A80010	6C 08 52 F5 6D FD 6A 3A 80 EF D5 99 C1 D7 24 F2	06A80010	6C 08 52 F5 6D FD 6A 3A 80 EF D5 99 C1 D7 24 F2
06A80020	72 3E B1 2C 60 EA 4C 71 A1 16 D8 FF 85 A0 29 67	06A80020	72 3E B1 2C 60 EA 4C 71 A1 16 D8 FF 85 A0 29 67	06A80020	72 3E B1 2C 60 EA 4C 71 A1 16 D8 FF 85 A0 29 67	06A80020	72 3E B1 2C 60 EA 4C 71 A1 16 D8 FF 85 A0 29 67	06A80020	72 3E B1 2C 60 EA 4C 71 A1 16 D8 FF 85 A0 29 67	06A80020	72 3E B1 2C 60 EA 4C 71 A1 16 D8 FF 85 A0 29 67	06A80020	72 3E B1 2C 60 EA 4C 71 A1 16 D8 FF 85 A0 29 67	06A80020	72 3E B1 2C 60 EA 4C 71 A1 16 D8 FF 85 A0 29 67
06A80030	09 A0 97 A2 C1 1E E0 8A 5D 47 9D 6E 5E E5 C5 76	06A80030	09 A0 97 A2 C1 1E E0 8A 5D 47 9D 6E 5E E5 C5 76	06A80030	09 A0 97 A2 C1 1E E0 8A 5D 47 9D 6E 5E E5 C5 76	06A80030	09 A0 97 A2 C1 1E E0 8A 5D 47 9D 6E 5E E5 C5 76	06A80030	09 A0 97 A2 C1 1E E0 8A 5D 47 9D 6E 5E E5 C5 76	06A80030	09 A0 97 A2 C1 1E E0 8A 5D 47 9D 6E 5E E5 C5 76	06A80030	09 A0 97 A2 C1 1E E0 8A 5D 47 9D 6E 5E E5 C5 76	06A80030	09 A0 97 A2 C1 1E E0 8A 5D 47 9D 6E 5E E5 C5 76
06A80040	94 64 92 12 FE D0 7F F2 A2 82 87 6F 03 D2 C8 A2	06A80040	94 64 92 12 FE D0 7F F2 A2 82 87 6F 03 D2 C8 A2	06A80040	94 64 92 12 FE D0 7F F2 A2 82 87 6F 03 D2 C8 A2	06A80040	94 64 92 12 FE D0 7F F2 A2 82 87 6F 03 D2 C8 A2	06A80040	94 64 92 12 FE D0 7F F2 A2 82 87 6F 03 D2 C8 A2	06A80040	94 64 92 12 FE D0 7F F2 A2 82 87 6F 03 D2 C8 A2	06A80040	94 64 92 12 FE D0 7F F2 A2 82 87 6F 03 D2 C8 A2	06A80040	94 64 92 12 FE D0 7F F2 A2 82 87 6F 03 D2 C8 A2

Figure 155

The BcryptGenRandom routine is utilized to generate 32 random bytes:

0048E1AA	6A 02	push 2			
0048E1AC	FF 75 0C	push dword ptr ss:[ebp+C]			
0048E1AE	FF 75 08	push dword ptr ss:[ebp+8]			
0048E1B0	6A 00	push 0			
0048E1B4	FF D0	call eax	eax:BcryptG		

eax=<bcrypt.BCryptGenRandom> (73817DE0)		Default (stdcal)	
.text:0048E1B4 lockbit.exe:\$8E1B4 #B05B4		1: [esp] 00000000	
		2: [esp+4] 06BFAA8	
		3: [esp+8] 00000020	
		4: [esp+C] 00000002	
		5: [esp+10] 06AA0050	

Dump 1		Dump 2		Dump 3		Dump 4		Dump 5		Watch 1		[x] Locals		Struct	
Address	Hex	Address	Hex	Address	Hex	Address	Hex	Address	Hex	Address	Hex	Address	Hex	Address	Hex
06BFAA8	54 1F 11 C0 C0 0F F1 28 11 48 11 51 42 56 D7 E2	06BFAA8	54 1F 11 C0 C0 0F F1 28 11 48 11 51 42 56 D7 E2	06BFAA8	54 1F 11 C0 C0 0F F1 28 11 48 11 51 42 56 D7 E2	06BFAA8	54 1F 11 C0 C0 0F F1 28 11 48 11 51 42 56 D7 E2	06BFAA8	54 1F 11 C0 C0 0F F1 28 11 48 11 51 42 56 D7 E2	06BFAA8	54 1F 11 C0 C0 0F F1 28 11 48 11 51 42 56 D7 E2	06BFAA8	54 1F 11 C0 C0 0F F1 28 11 48 11 51 42 56 D7 E2	06BFAA8	54 1F 11 C0 C0 0F F1 28 11 48 11 51 42 56 D7 E2
06BFA88	02 A6 F5 B4 30 5E BC 72 FC 01 5A BF E5 F8 01 C3	06BFA88	02 A6 F5 B4 30 5E BC 72 FC 01 5A BF E5 F8 01 C3	06BFA88	02 A6 F5 B4 30 5E BC 72 FC 01 5A BF E5 F8 01 C3	06BFA88	02 A6 F5 B4 30 5E BC 72 FC 01 5A BF E5 F8 01 C3	06BFA88	02 A6 F5 B4 30 5E BC 72 FC 01 5A BF E5 F8 01 C3	06BFA88	02 A6 F5 B4 30 5E BC 72 FC 01 5A BF E5 F8 01 C3	06BFA88	02 A6 F5 B4 30 5E BC 72 FC 01 5A BF E5 F8 01 C3	06BFA88	02 A6 F5 B4 30 5E BC 72 FC 01 5A BF E5 F8 01 C3

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#.ä.÷%.C..÷.V
065BF8F0	05 C5 A3 C4 28 56 41 85 EA D0 CD 2E F1 83 D0 76	.ÄfA(VAUëDÎ.ñ•Dv
065BF900	A5 27 53 28 E9 D0 F0 C9 24 B3 08 74 66 F4 FC 4B	¥'S+ëD0E\$\$.tfôüK
065BF910	DA 5F 25 A8 37 D8 DC D7 4B 50 C5 86 71 DA D5 3E	U.%70üxKpA,qü0x

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:

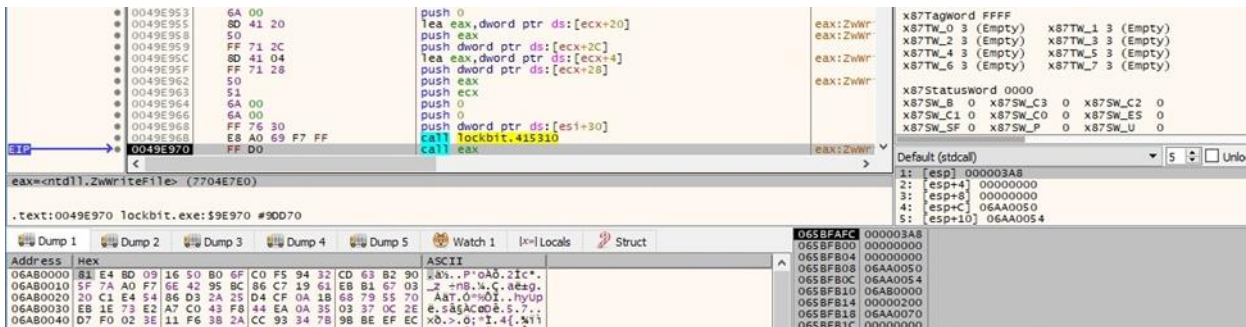


Figure 159

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

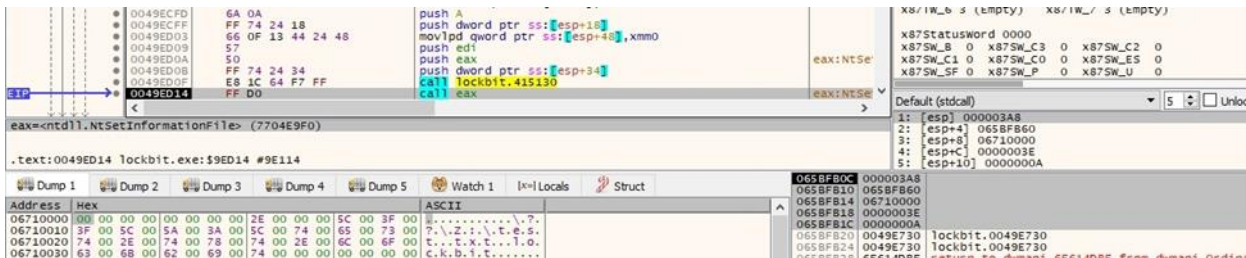


Figure 160

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:

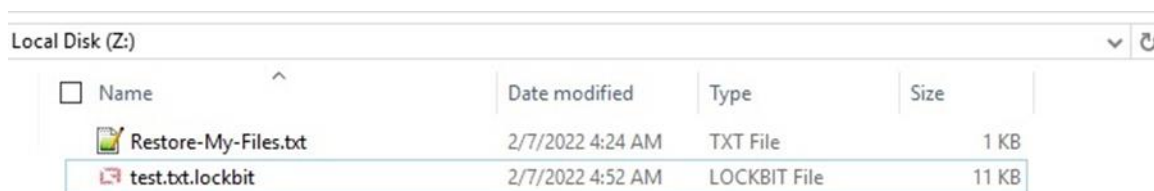


Figure 163

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):

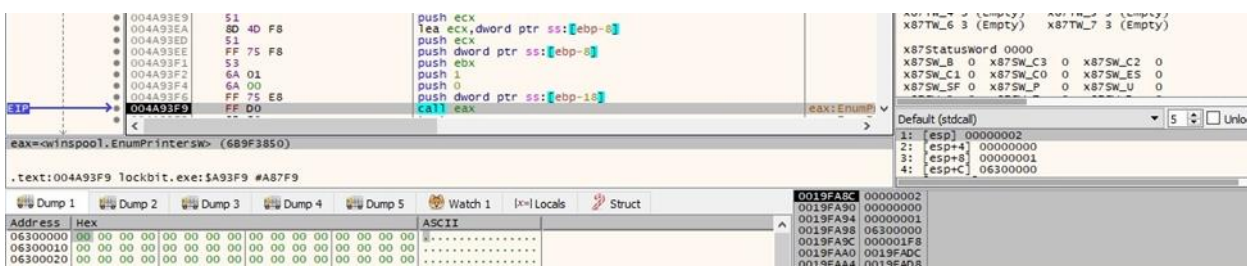


Figure 164

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:

```
.text:004AA7ED
.text:004AA7ED loc_4AA7ED:
.text:004AA7ED push [ebp+var_24]
.text:004AA7F0 call eax ; dword_4F8D3C
.text:004AA7F2 mov eax, [ebp+var_1C]
.text:004AA7F5 inc eax
.text:004AA7F6 mov [ebp+var_1C], eax
.text:004AA7F9 cmp eax, 9999
.text:004AA7FE jnl loc_4AA410
```

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 GdipAddPathArc calls in order to add elliptical arcs to the current figure of the path:

Figure 187

The ransomware performs function calls such as GdipFillPath and GdipDrawPath 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 GdipCreateFont:

Figure 189

The GdipDrawImageRect 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 GdipMeasureString API:



Figure 191

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

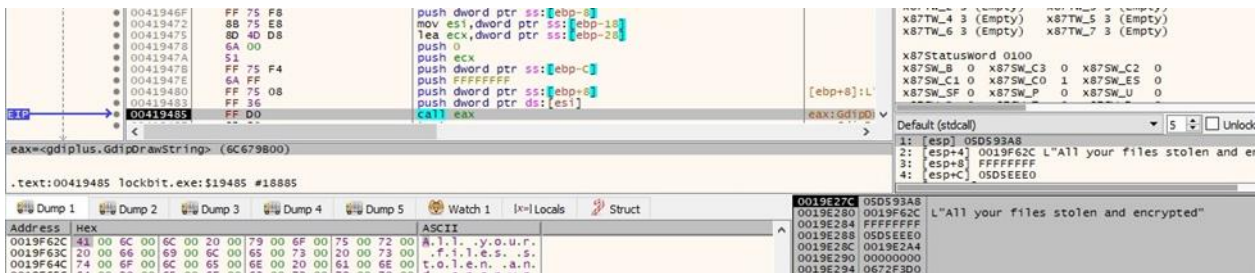


Figure 192

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



Figure 193

GetTempFileNameW is utilized to create a temporary file:



Figure 194

The GdiGetImageEncoders 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**):

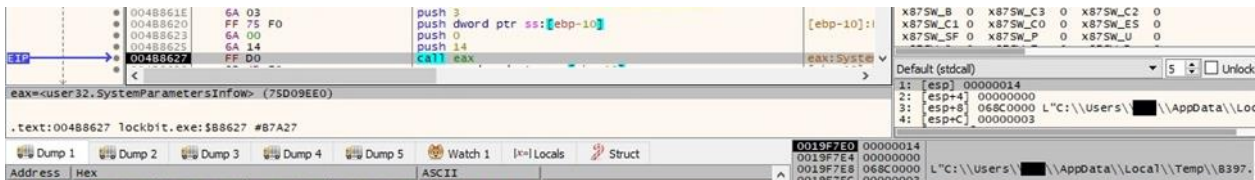


Figure 201

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

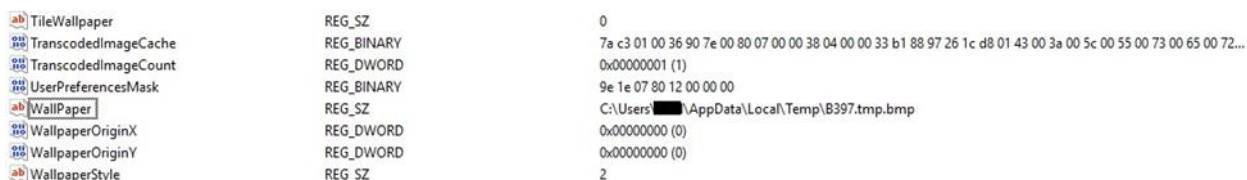


Figure 202

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


```
004953AE 6A 00 push 0
```

[illegible]

Figure 204

004AD813	6A 00	push 0	x87r4 3FFF80000000000000000000 ST4 Empty 1.0000000000
----------	-------	--------	---

```

004AD813 6A 00      push 0
004AD815 6A 00      push 0
004AD817 6A 00      push 0
004AD819 8B 85 D8 F0 FF FF  lea eax,dword ptr ss:[ebp-22h]
004AD81F C7 45 04 18 00 00 00  mov dword ptr ss:[ebp-1Ch],18
004AD826 89 45 EC    mov dword ptr ss:[ebp-14],eax
004AD829 8D 45 04    lea eax,dword ptr ss:[ebp-1Ch]
004AD82C 6A 00      push 0
004AD82E 50         push eax
004AD82F 8B 45 E8    lea eax,dword ptr ss:[ebp-20h]
004AD832 89 7D E0    mov dword ptr ss:[ebp-18h],edi
004AD835 68 00 00 00 02      push 2000000
004AD83A 50         push eax
004AD83B C7 45 F0 40 00 00 00  mov dword ptr ss:[ebp-10h],40
004AD842 89 7D F4    mov dword ptr ss:[ebp-Ch],edi
004AD845 8B 7D F8    mov dword ptr ss:[ebp-8h],edi
004AD848 E8 23 80 F6 FF      call 7C90106B
004AD84D FF D0      call eax

```

```

X87r4 3FFF8000000000000000 ST4 Empty 1.000000000000
X87r5 4005C000000000000000 ST5 Empty 96.000000000000
X87r6 4005C000000000000000 ST6 Empty 96.000000000000
X87r7 4005C000000000000000 ST7 Empty 96.000000000000

X87TagWord FFFF
X87TW_0 3 (Empty) X87TW_1 3 (Empty)
X87TW_2 3 (Empty) X87TW_3 3 (Empty)
X87TW_4 3 (Empty) X87TW_5 3 (Empty)
X87TW_6 3 (Empty) X87TW_7 3 (Empty)

X87StatusWord 0000
X87SW_B 0 X87SW_C3 0 X87SW_C2 0
X87SW_C1 0 X87SW_C0 0 X87SW_E5 0
X87SW_SF 0 X87SW_P 0 X87SW_U 0

Default (f8dc)
1: [esp+1] 0019816C
2: [esp+4] 02000000
3: [esp+8] 00198170
4: [esp+C] 00000000

```

```

004AD84D FF D0      call eax

```

```

X87TW_0 3 (Empty) X87TW_1 3 (Empty)
X87TW_2 3 (Empty) X87TW_3 3 (Empty)
X87TW_4 3 (Empty) X87TW_5 3 (Empty)
X87TW_6 3 (Empty) X87TW_7 3 (Empty)

X87StatusWord 0000
X87SW_B 0 X87SW_C3 0 X87SW_C2 0
X87SW_C1 0 X87SW_C0 0 X87SW_E5 0
X87SW_SF 0 X87SW_P 0 X87SW_U 0

```

```

004AD84D FF D0      call eax

```

```

X87TW_0 3 (Empty) X87TW_1 3 (Empty)
X87TW_2 3 (Empty) X87TW_3 3 (Empty)
X87TW_4 3 (Empty) X87TW_5 3 (Empty)
X87TW_6 3 (Empty) X87TW_7 3 (Empty)

X87StatusWord 0000
X87SW_B 0 X87SW_C3 0 X87SW_C2 0
X87SW_C1 0 X87SW_C0 0 X87SW_E5 0
X87SW_SF 0 X87SW_P 0 X87SW_U 0

```

Figure 205

004D978	50	push eax	eax:ZwSet'	API: ZwSet' (ZwSet')	API: ZwSet' (ZwSet')
---------	----	----------	------------	----------------------	----------------------

The screenshot displays a debugger window with the following components:

- Assembly View:**
 - 004AD978: 50 push eax
 - 004AD979: FF B5 34 F7 FF push dword ptr ss:[ebp-8CC]
 - 004AD97F: 80 85 D0 FD FF lea eax, dword ptr ss:[ebp-230]
 - 004AD985: 6A 01 push 01
 - 004AD987: 50 push eax
 - 004AD989: 50 push eax
 - 004AD98A: FF 75 E0 push dword ptr ss:[ebp-20]
 - 004AD98D: E8 CE 7F F6 call lockbit.415960
 - 004AD992: FF D0 call eax
- Registers:**
 - eax:ZwSet'
 - x87TW_6 3 (Empty)
 - x87TW_7 3 (Empty)
- Stack:**
 - x87StatusWord 0000
 - x87SW_B 0 x87SW_C3 0 x87SW_C2 0
 - x87SW_C1 0 x87SW_C0 0 x87SW_E5 0
 - x87SW_SF 0 x87SW_P 0 x87SW_U 0
- Dump:**
 - 00197F5C: 00 00 02 00 64 81 19 00 56 00 58 00 AD 7A 19 00
 - 00197F6C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- ASCII:**
 - ...d...V..X..Z...

Figure 206

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

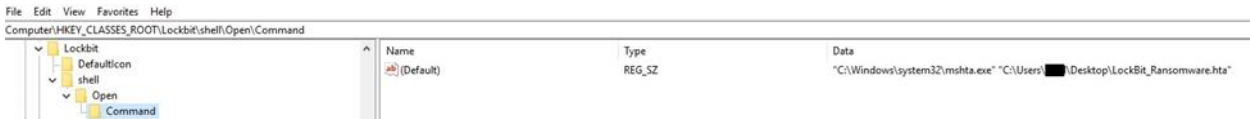


Figure 211

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

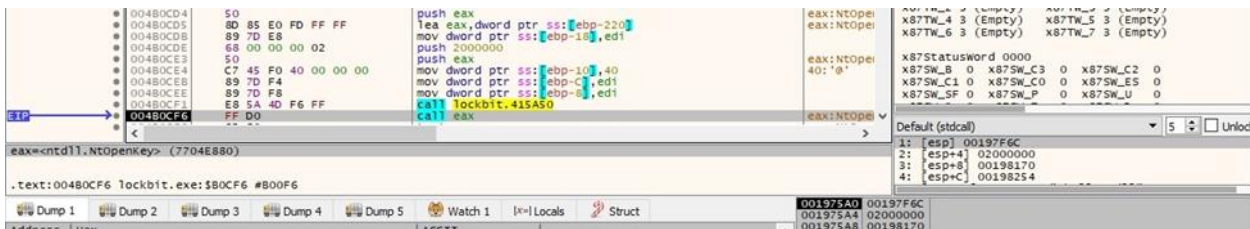


Figure 212

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

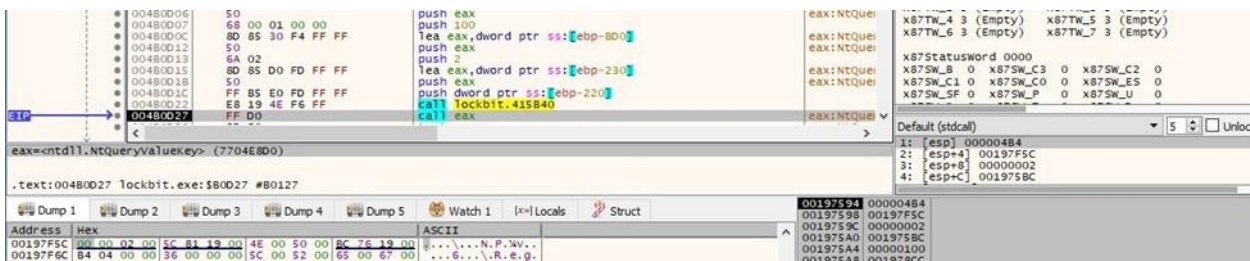


Figure 213

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

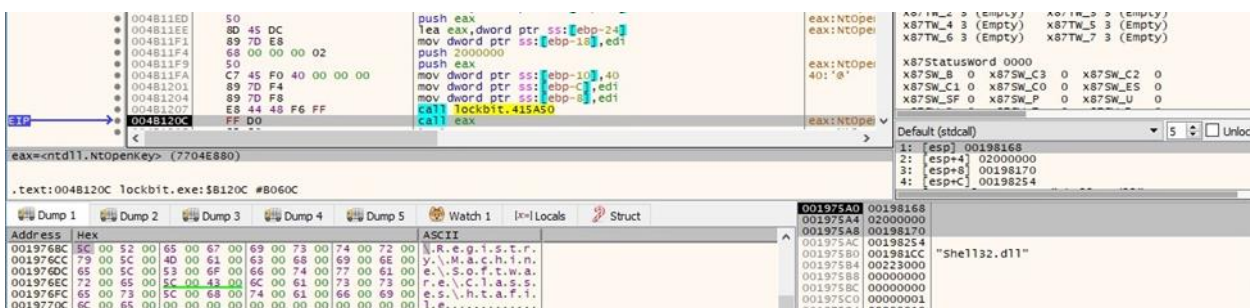


Figure 214

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 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

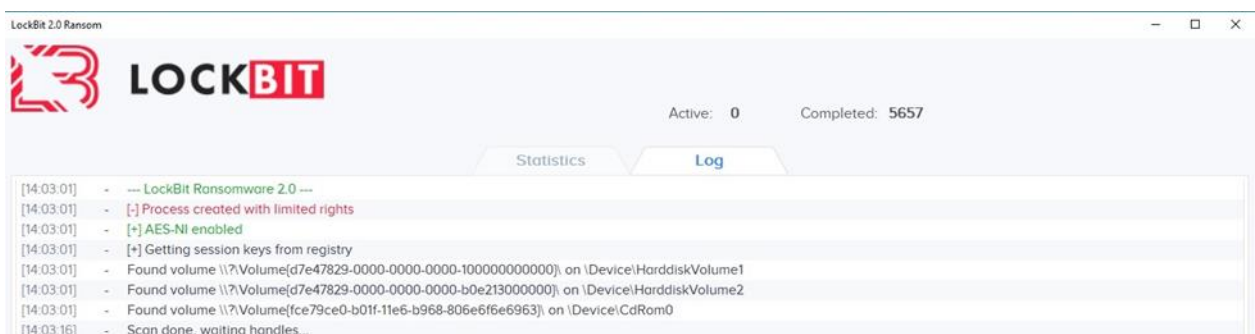


Figure 224

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 wdsfwfsafe 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 ocssd dbsnmp 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 pvlsrv 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 MVAarmor MVAarmor64 stc_raw_agent VSNAPVSS VeeamTransportSvc VeeamDeploymentService VeeamNFSSvc AcronisAgent ARSM AcrSch2Svc CASAD2DWebSvc CAARCUupdateSvc WSBExchange MExchange MExchange\$