A Detailed Analysis of The LockBit Ransomware

Prepared by: Vlad Pasca, LIFARS, LLC

Date: 02/14/2022

www.LIFARS.com

info@lifars.com

244 Fifth Avenue, Suite 2035,

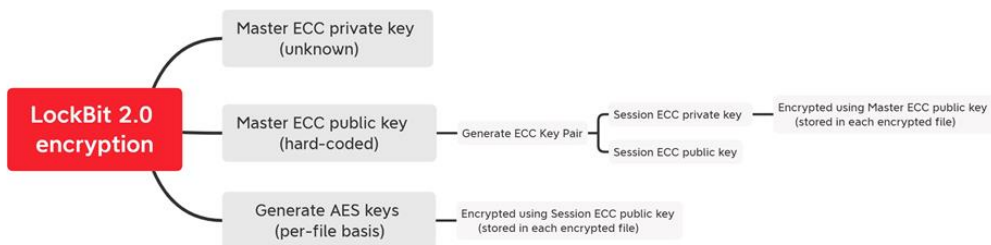
New York, NY 10001

1.212.222.7061

Table of Contents

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:

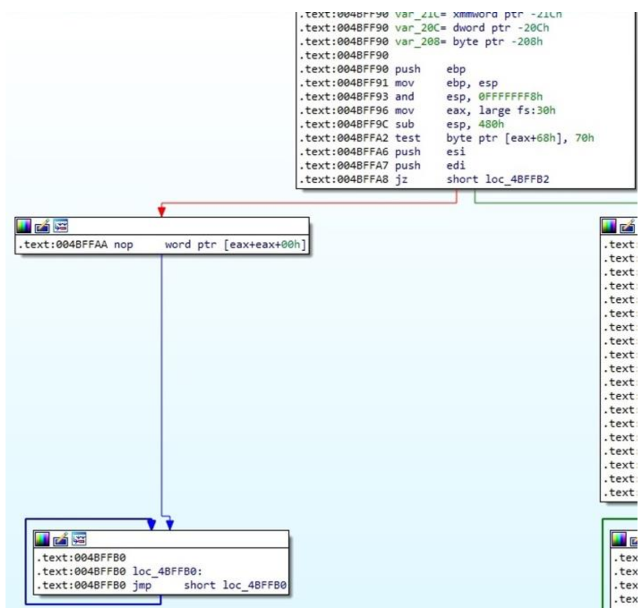


Figure 1

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:

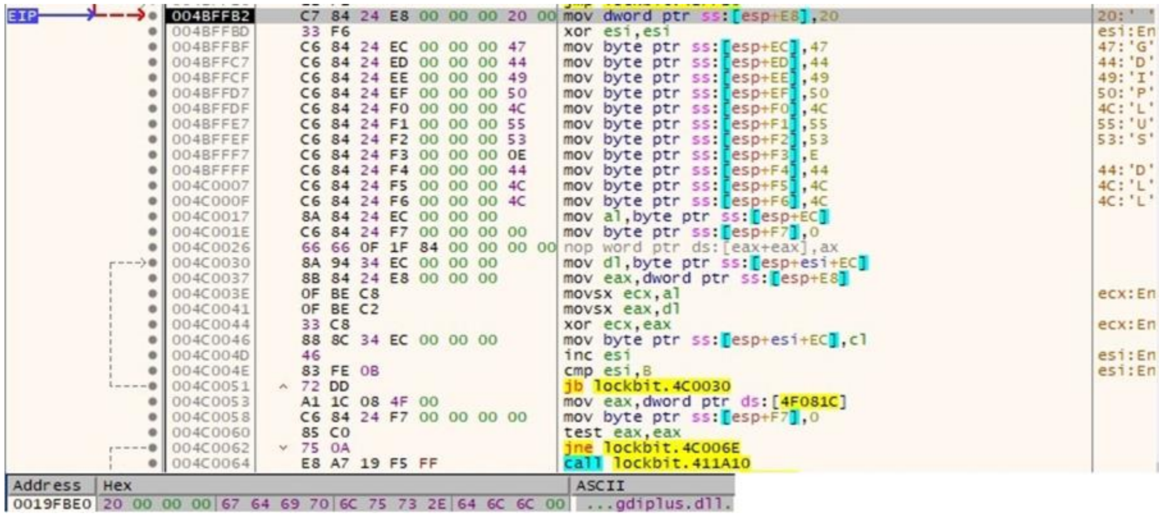


Figure 2

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):

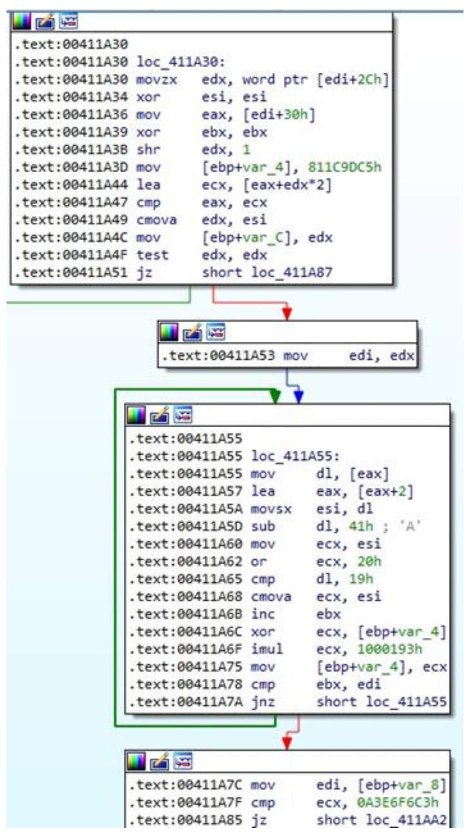


Figure 3

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

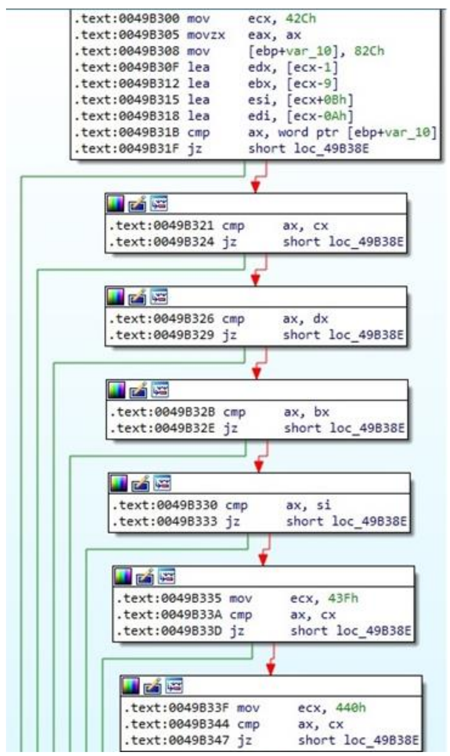


Figure 6

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:

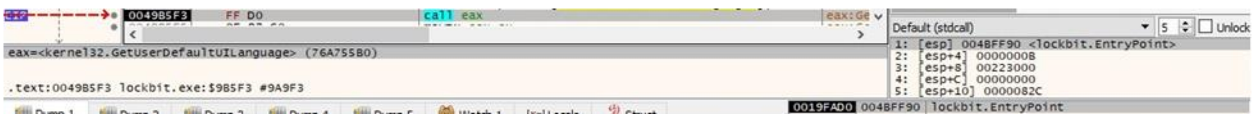


Figure 7

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**):

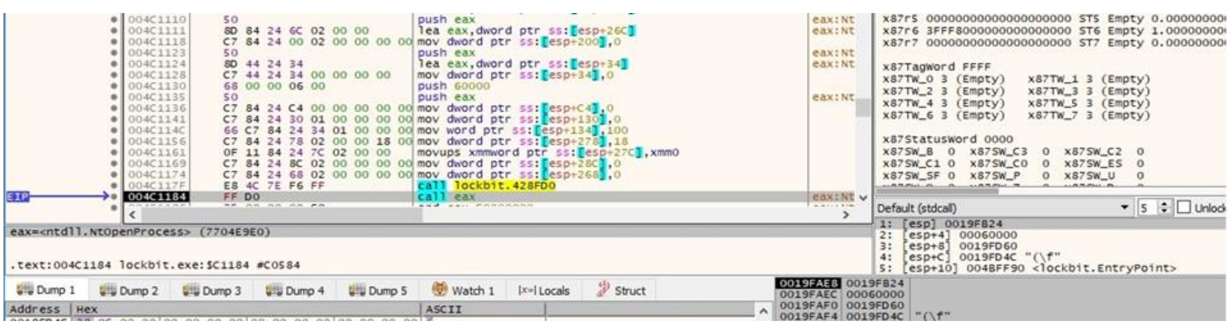


Figure 9

The GetSecurityInfo API is utilized to retrieve a pointer to the DACL in the returned security descriptor (0x6 = **SE_KERNEL_OBJECT**, 0x4 = **DACL_SECURITY_INFORMATION**):

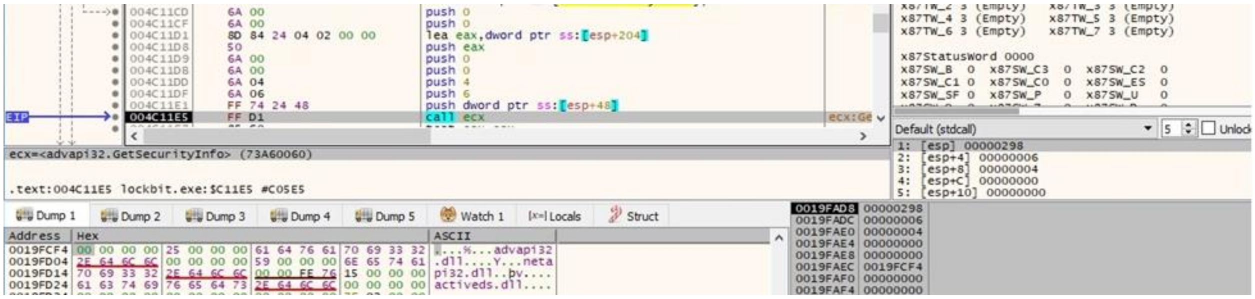


Figure 10

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

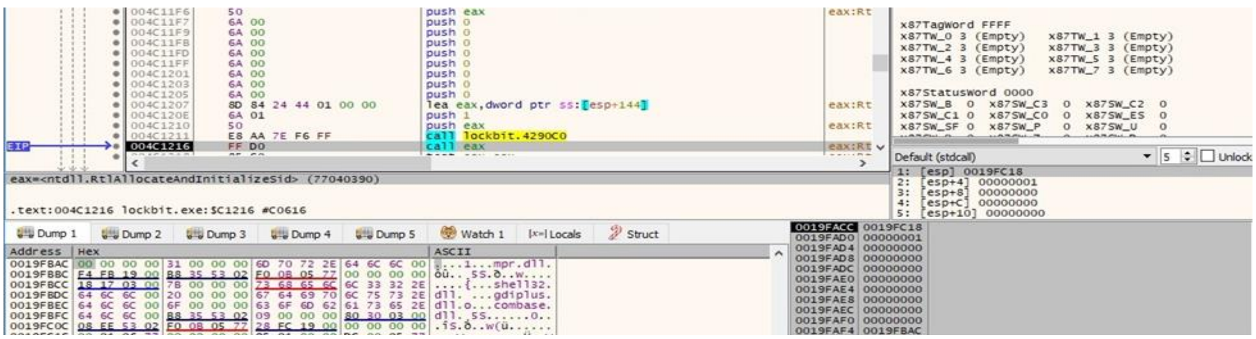


Figure 11

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

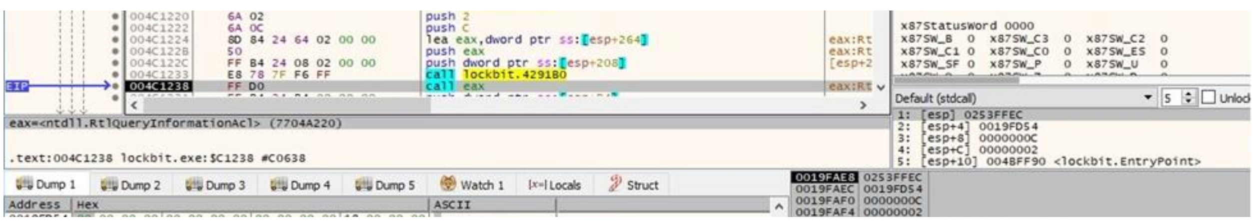


Figure 12

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:

Figure 13

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**):

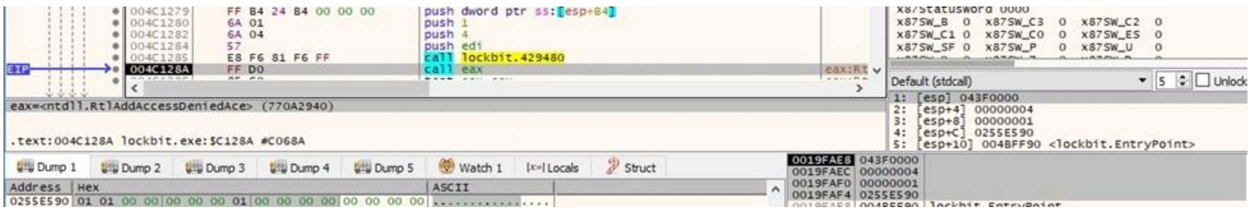


Figure 15

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

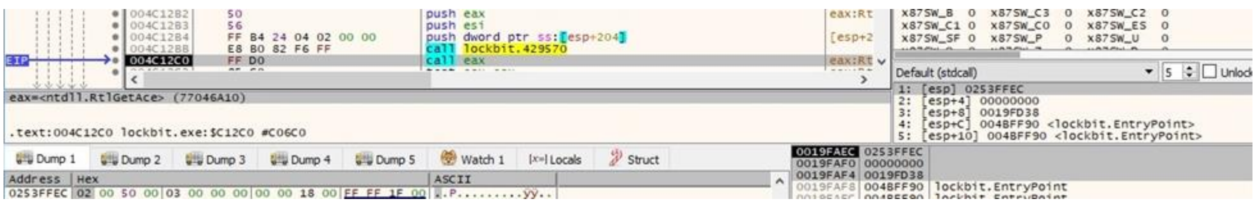


Figure 16

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

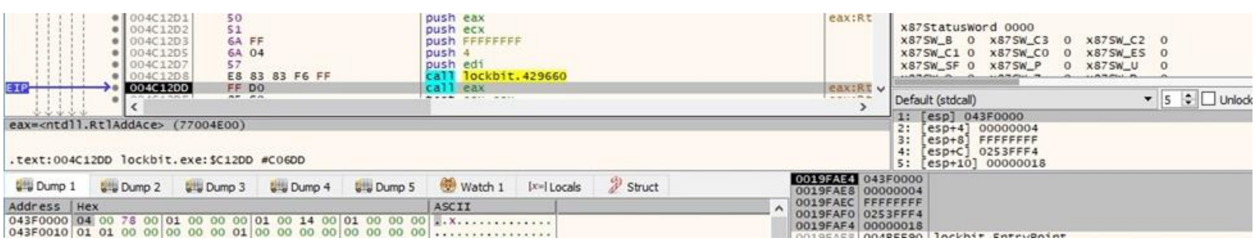


Figure 17

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**):

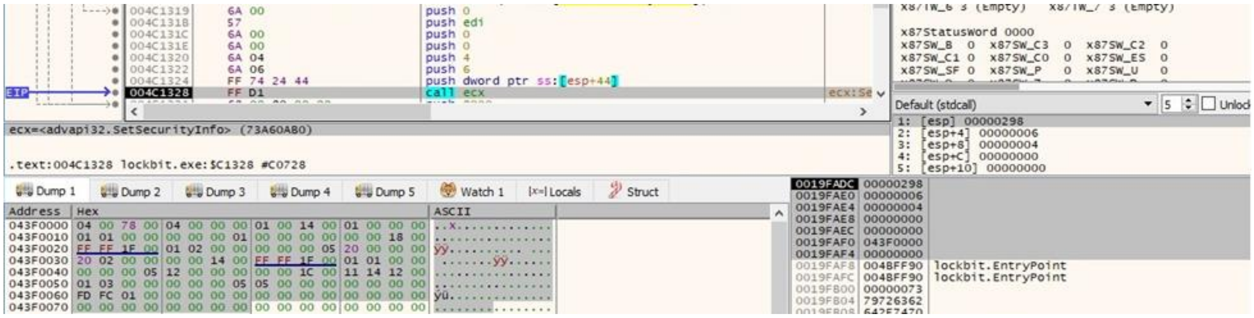


Figure 18

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**):

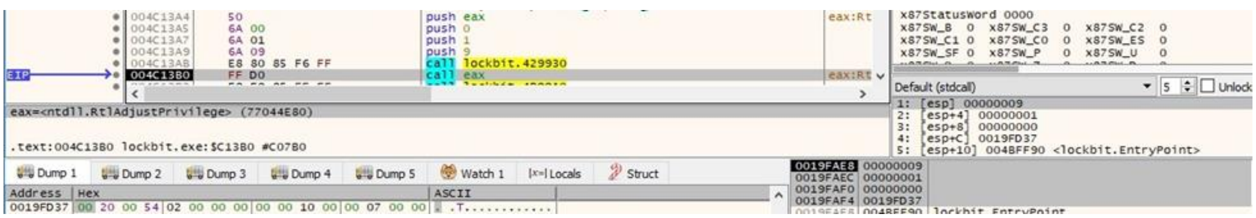


Figure 20

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

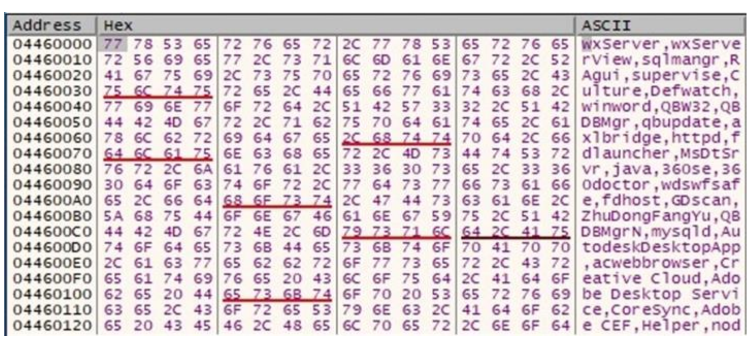


Figure 21

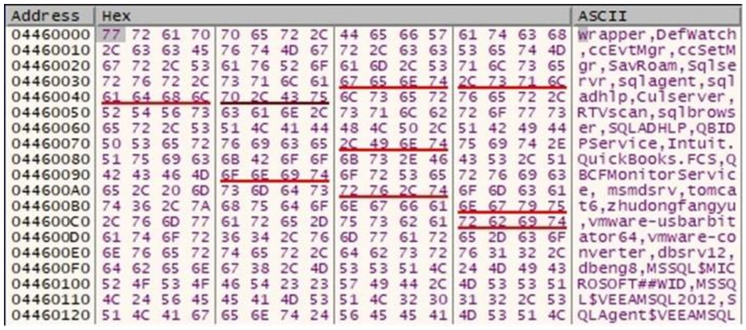


Figure 22

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**):

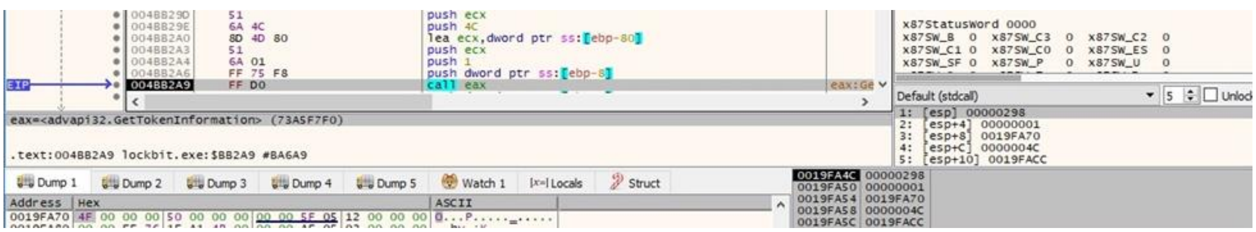


Figure 24

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

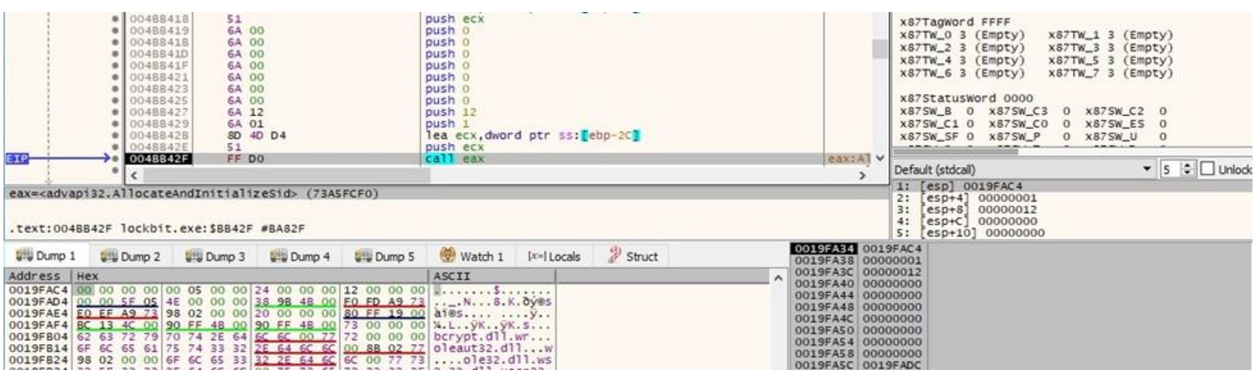


Figure 25

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

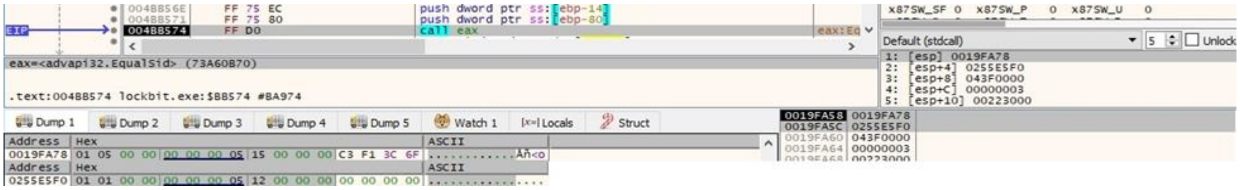


Figure 26

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:

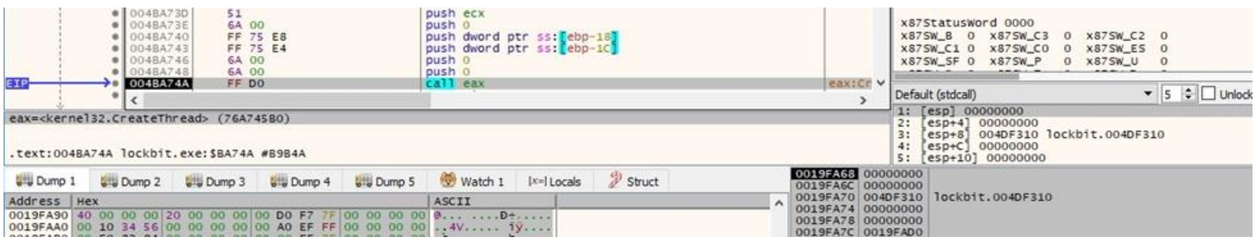


Figure 28

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

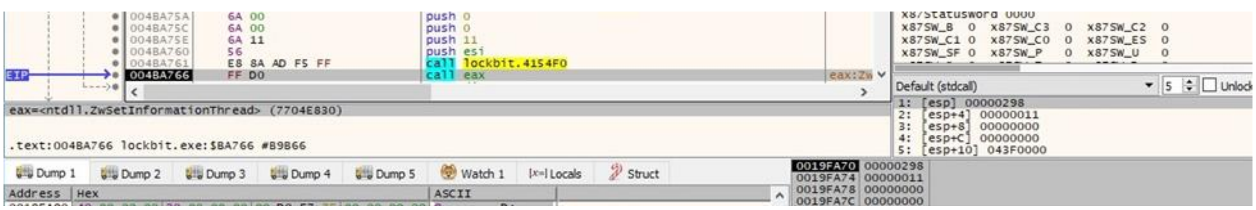


Figure 29

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:

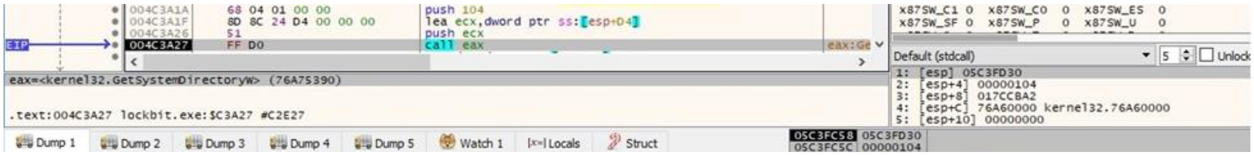


Figure 31

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

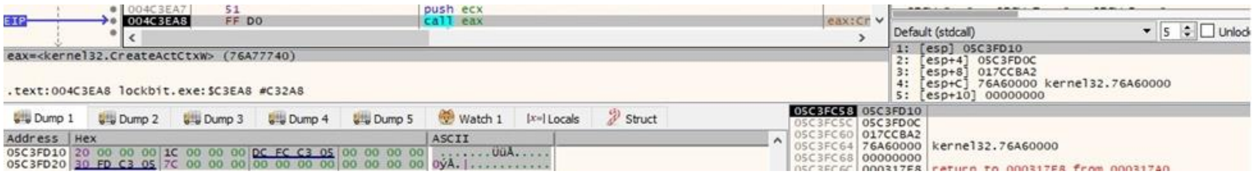


Figure 32

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:

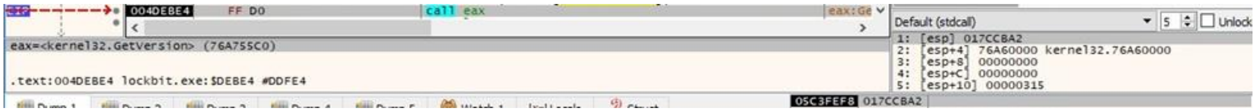


Figure 37

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:

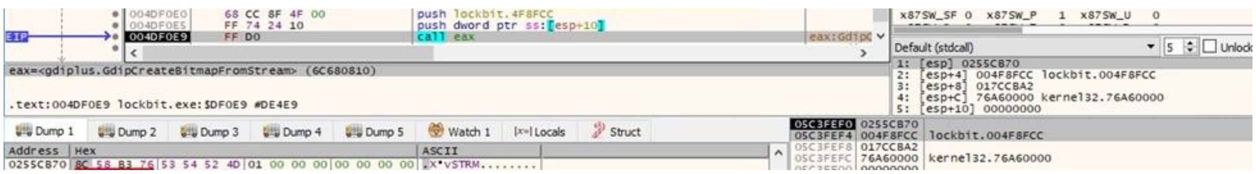


Figure 39

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

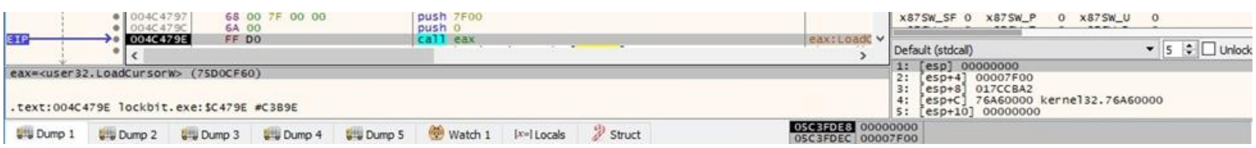


Figure 40

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:

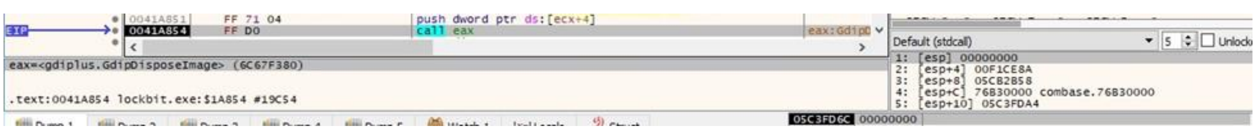


Figure 42

LockBit 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:

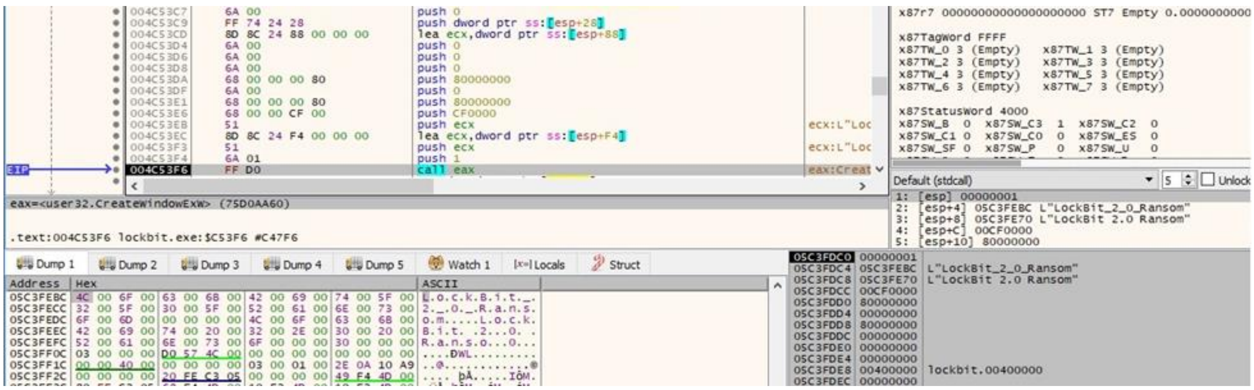


Figure 44

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

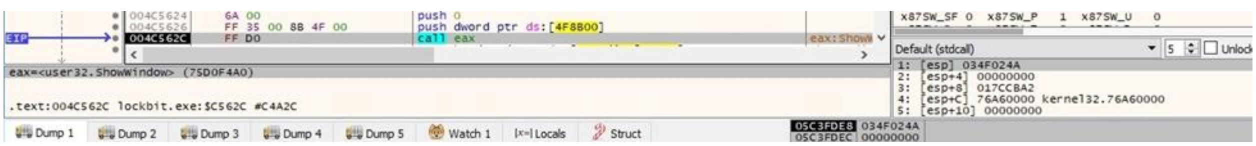


Figure 45

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

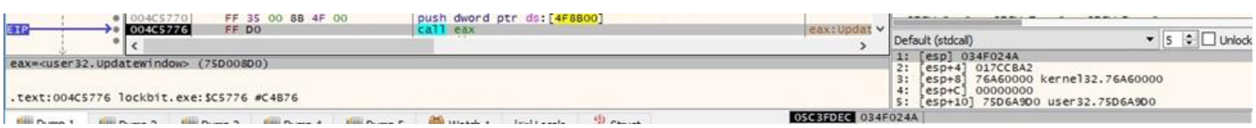


Figure 46

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

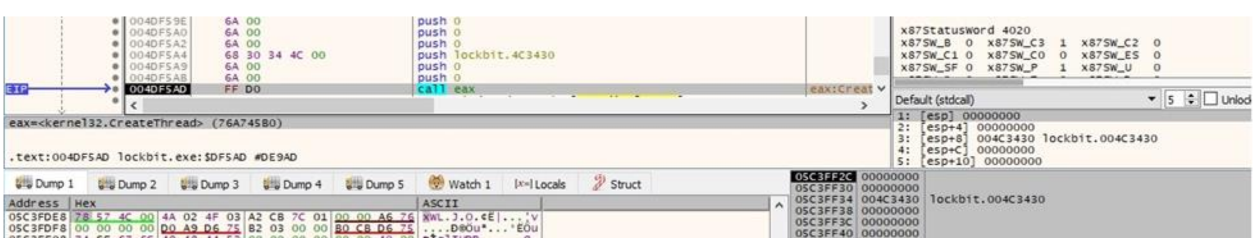


Figure 47

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

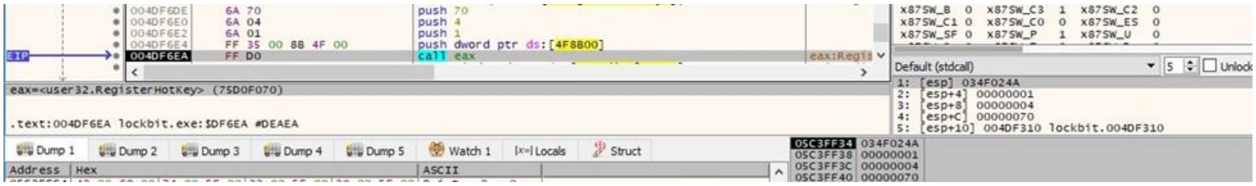


Figure 48

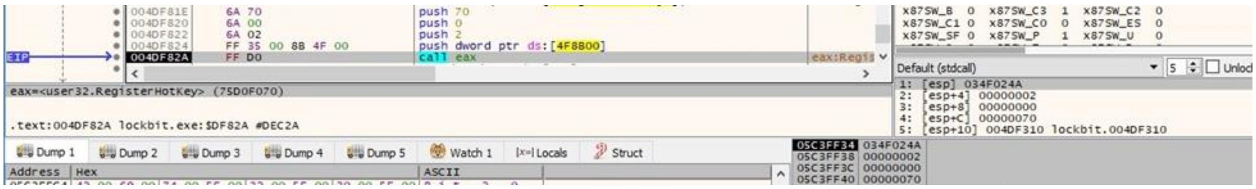


Figure 49

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

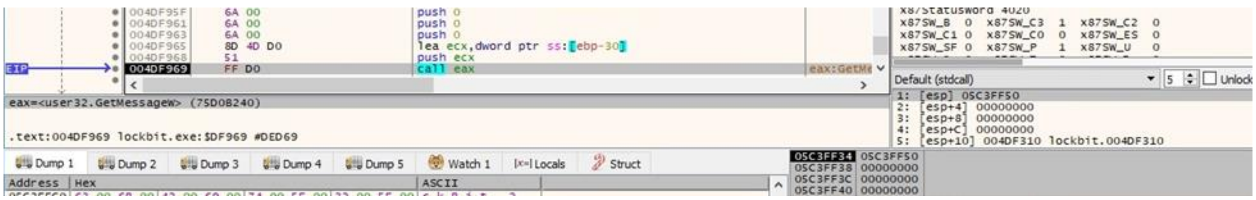


Figure 50

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

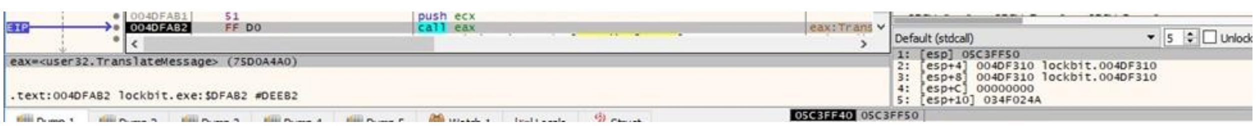


Figure 51

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**):

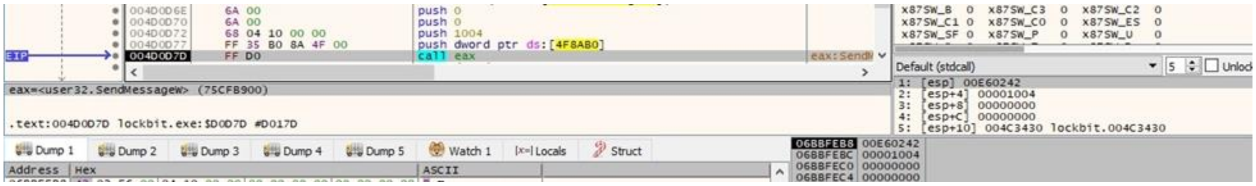


Figure 53

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

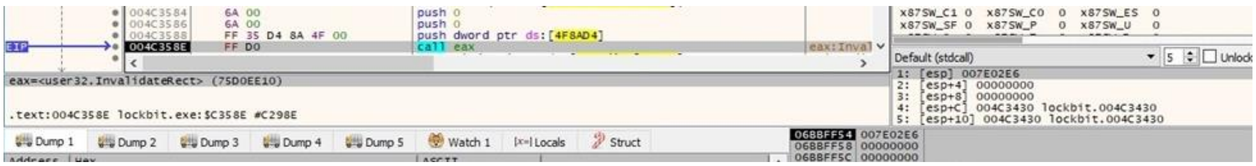


Figure 54

We continue with the analysis of the main thread.

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

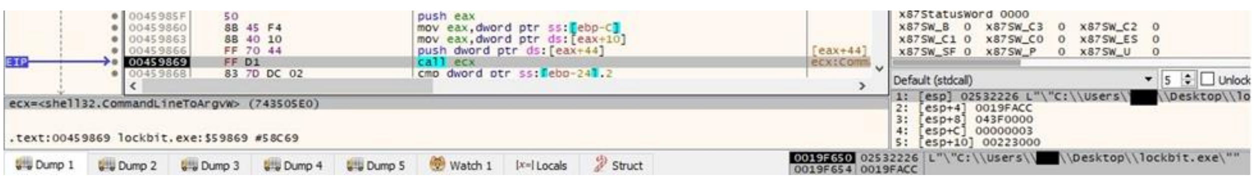


Figure 55

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

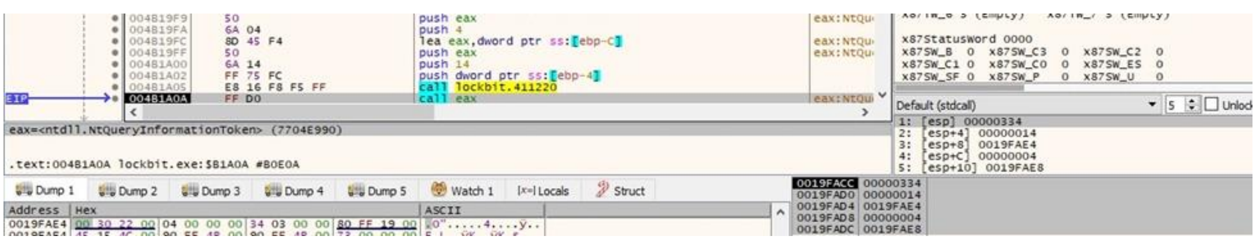


Figure 56

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:

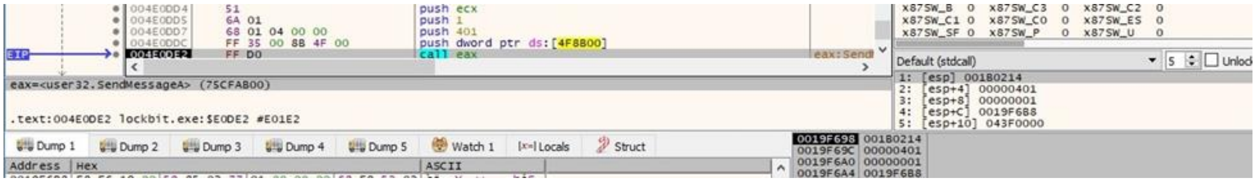


Figure 57

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**):

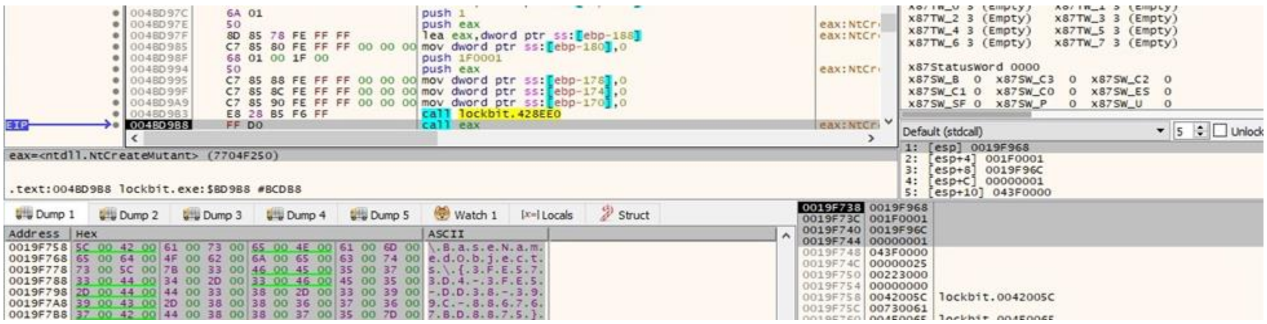


Figure 58

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}":

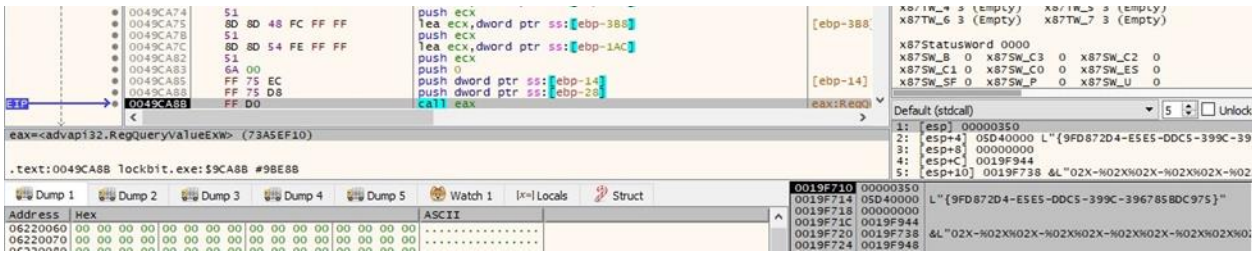


Figure 62

The malware establishes persistence by creating the above registry value:

Figure 63

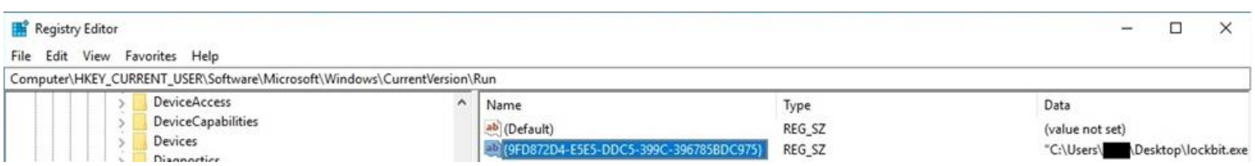


Figure 64

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

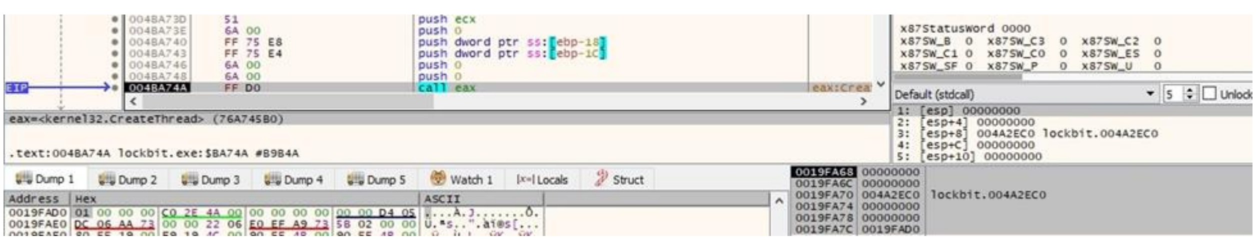


Figure 65

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**):

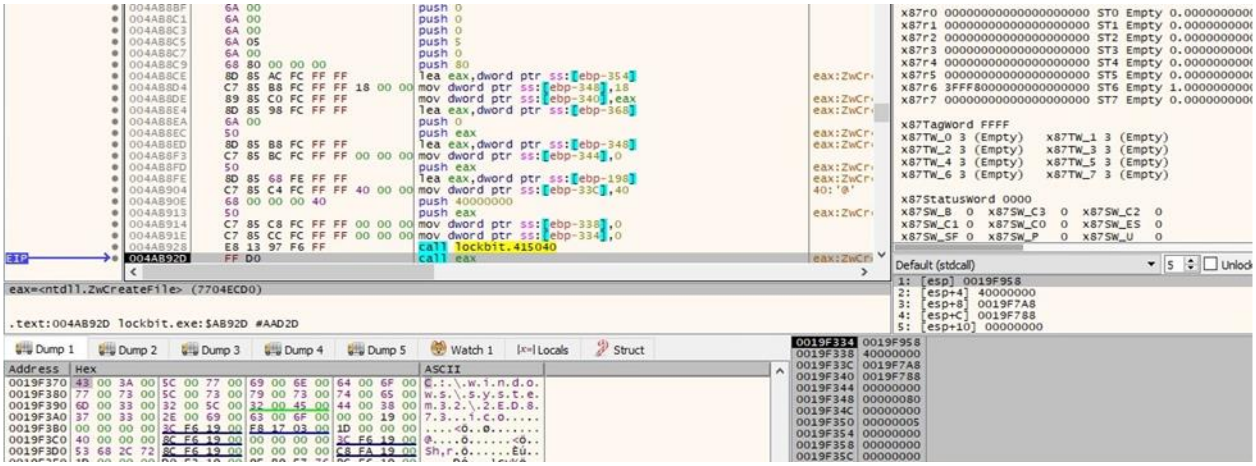


Figure 66

The ICO file is populated using the ZwWriteFile routine:

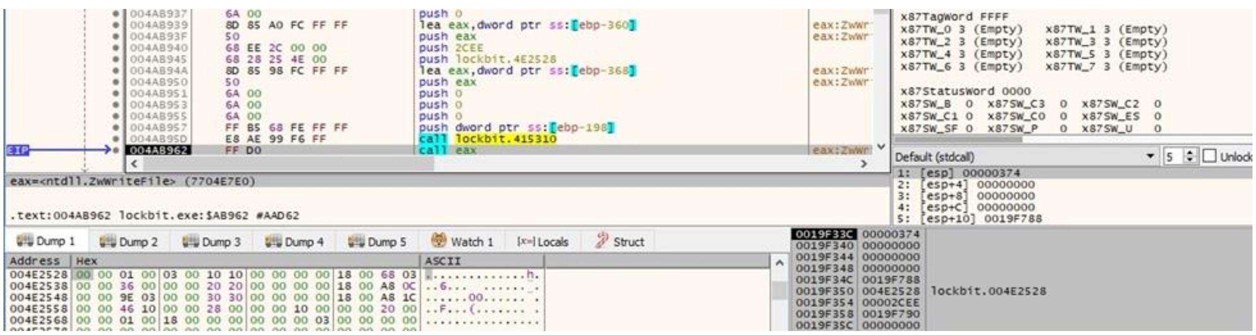


Figure 67

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

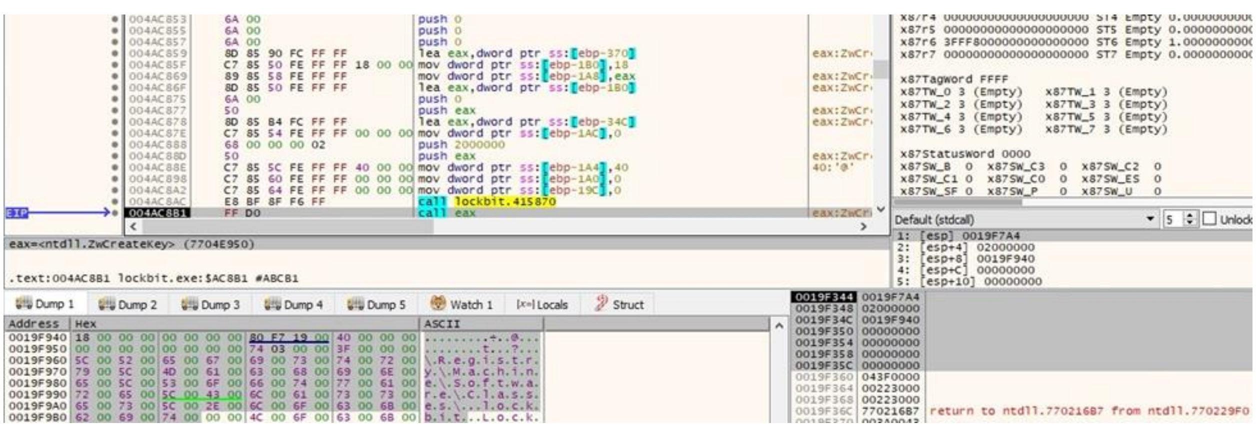


Figure 68

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:

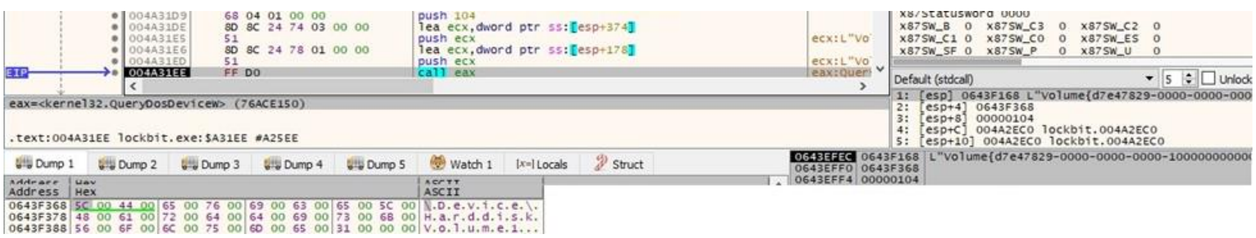


Figure 72

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

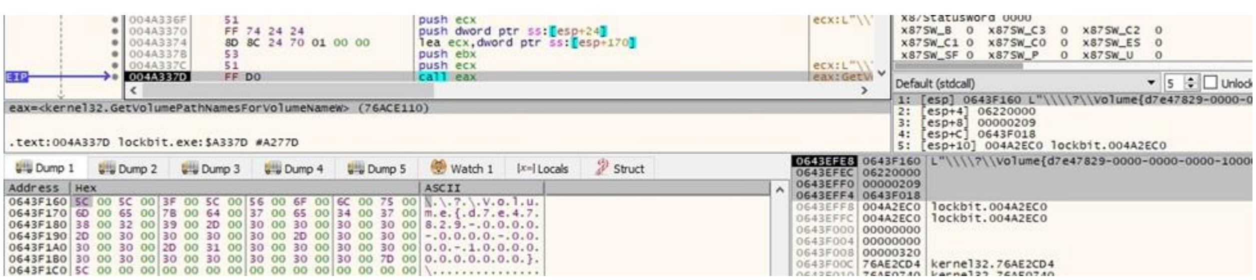


Figure 73

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.

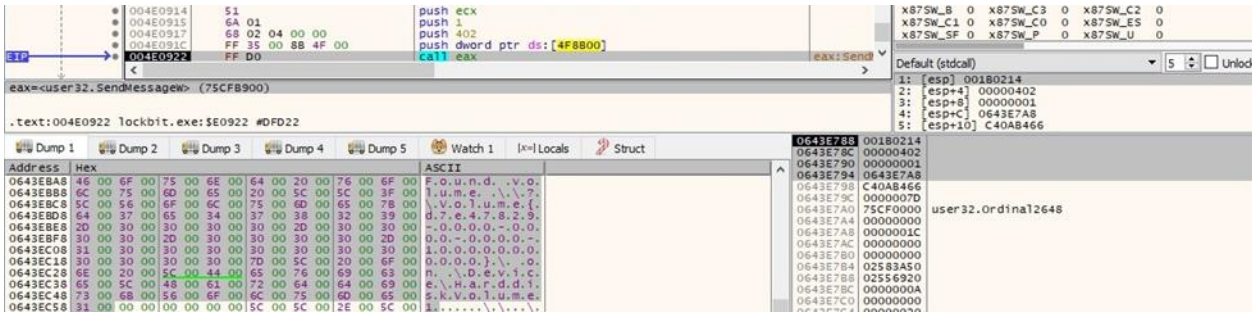


Figure 75

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

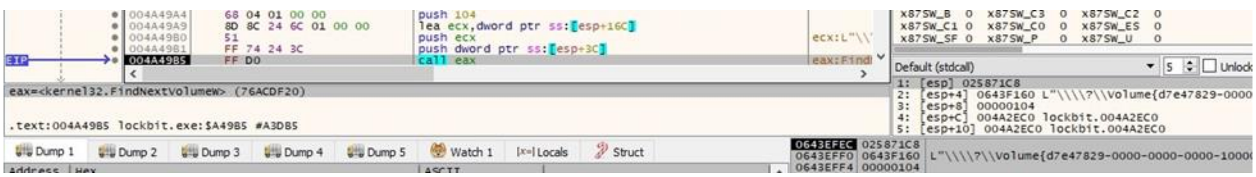


Figure 76

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**):

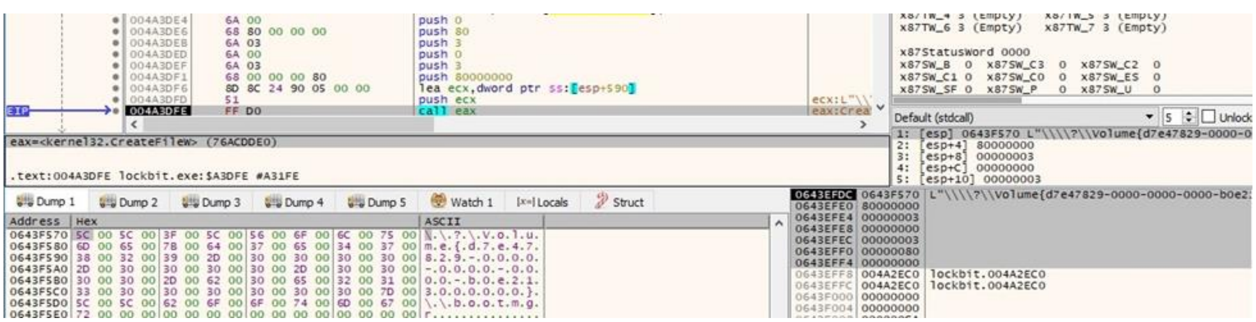


Figure 77

An unmounted volume is mounted by calling the SetVolumeMountPointW routine:

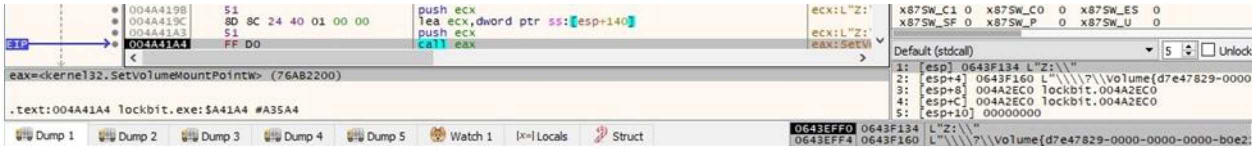


Figure 78

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.

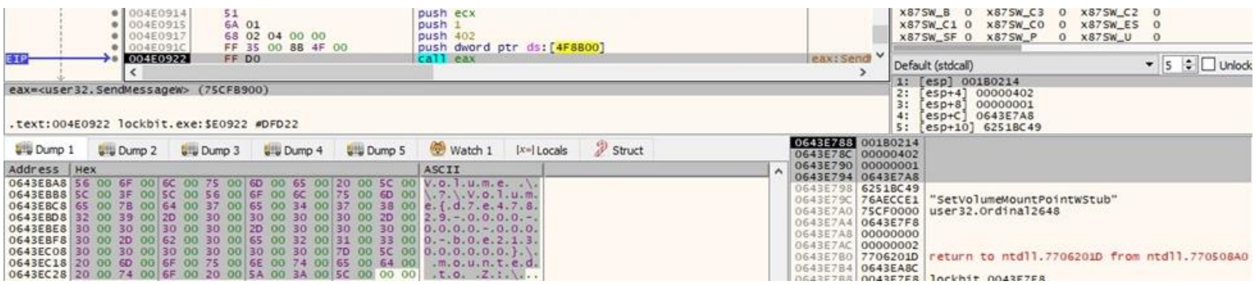


Figure 80

The binary calls the SHChangeNotify API with the SHCNE_ASSOCCHANGED parameter (0x8000000 = **SHCNE_ASSOCCHANGED**):

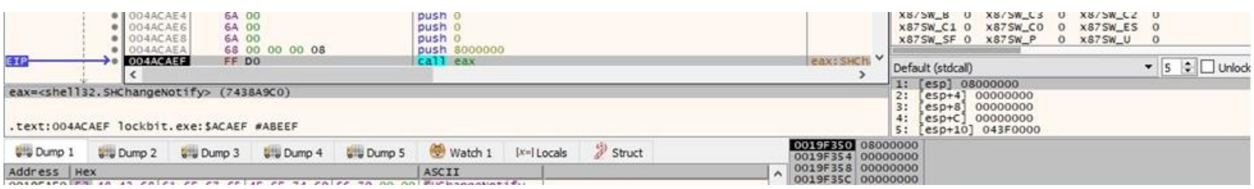


Figure 81

A new thread is created by the malware using CreateThread:

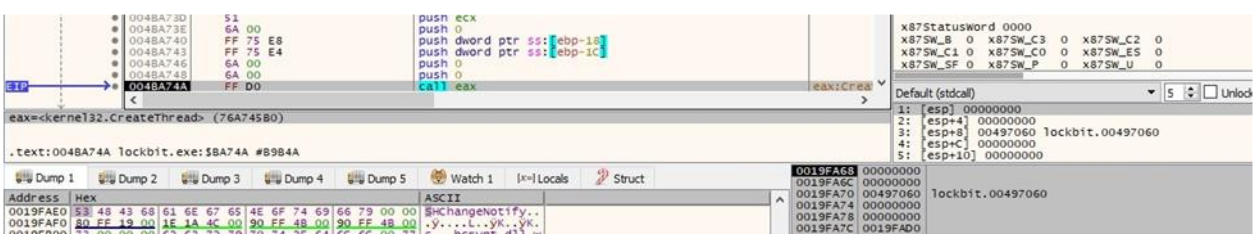


Figure 82

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:

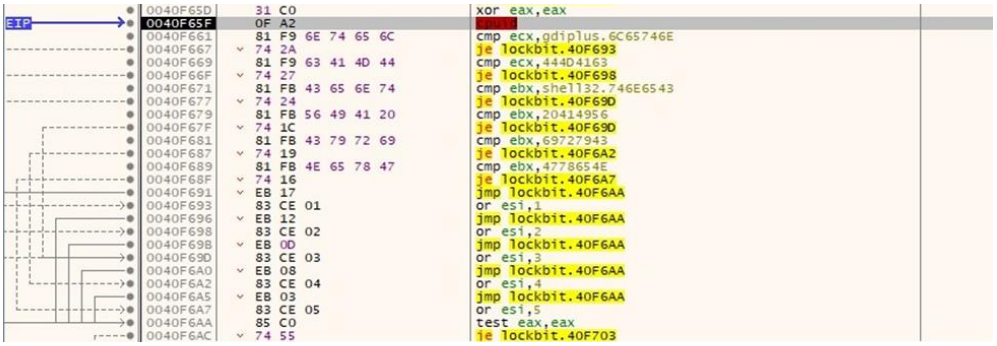


Figure 83

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:

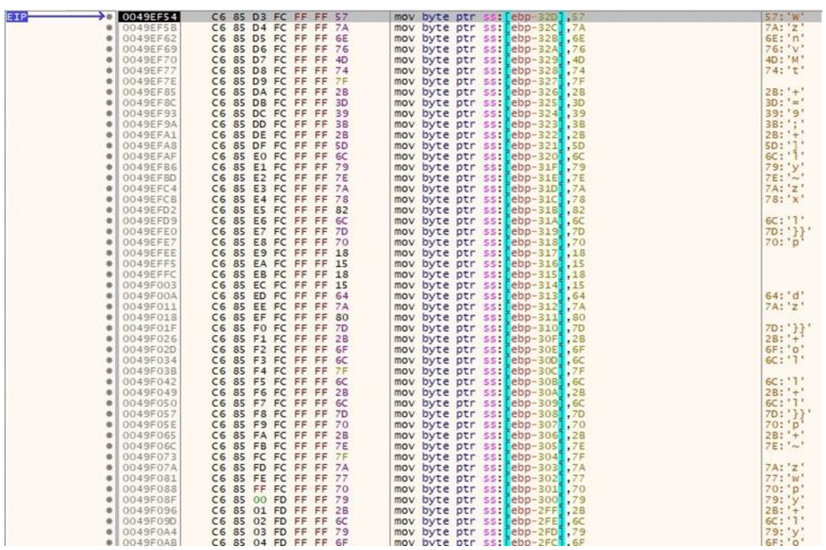


Figure 85

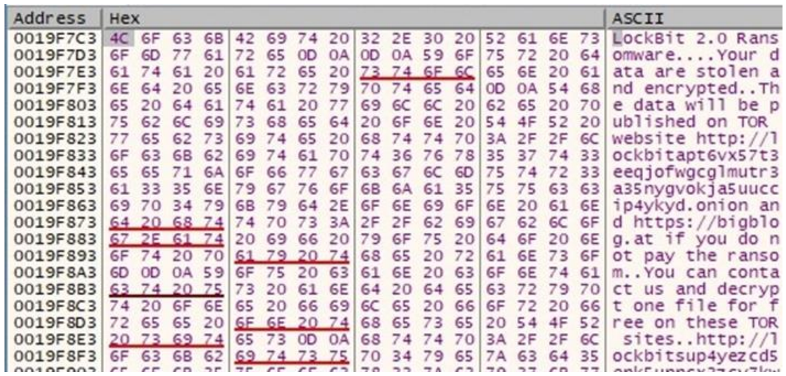


Figure 86

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:

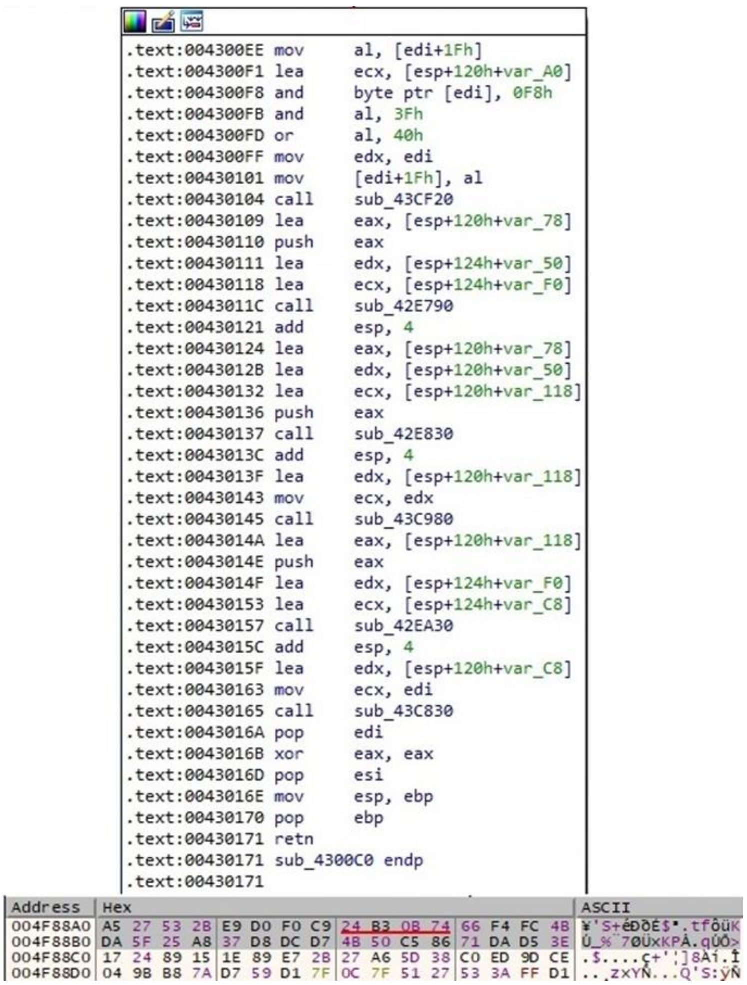


Figure 91

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:

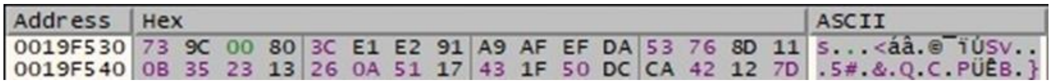


Figure 93

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):

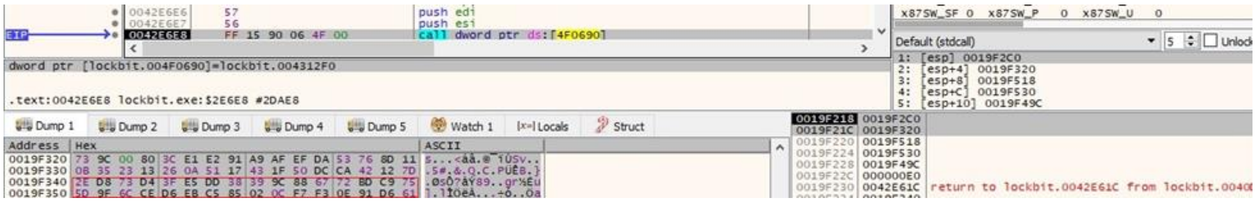


Figure 94

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

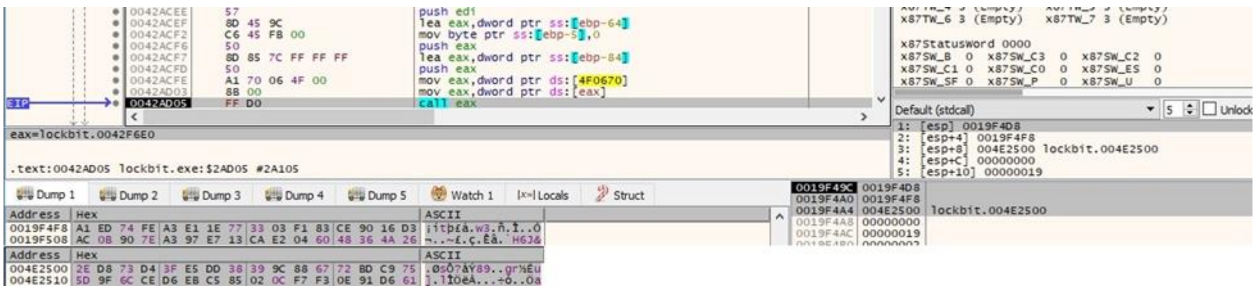


Figure 95

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     dimiter.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:

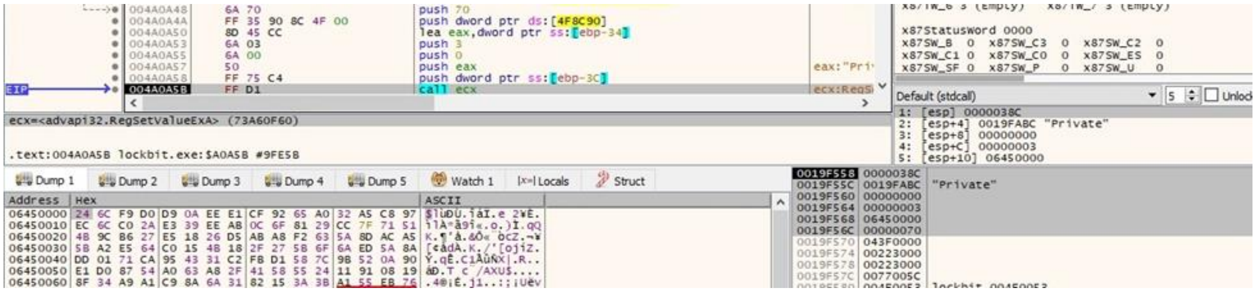


Figure 97

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

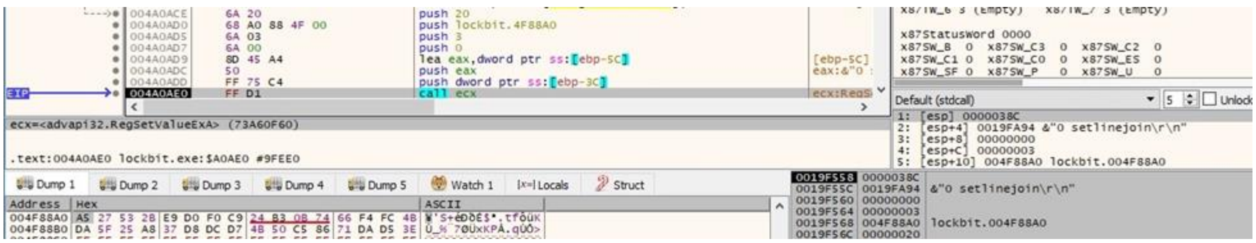


Figure 98

Figure 99 reveals both registry values with their content:

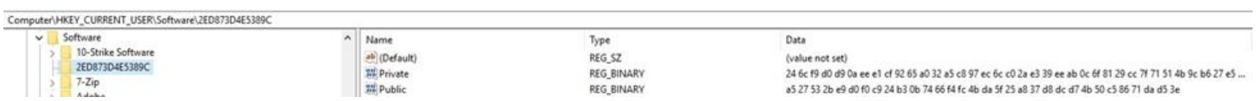


Figure 99

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**):

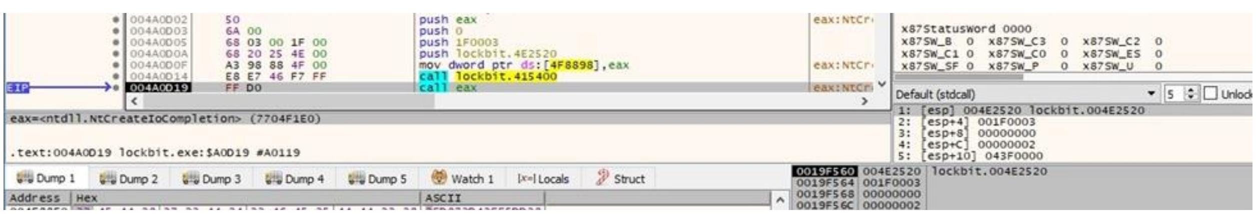


Figure 100

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

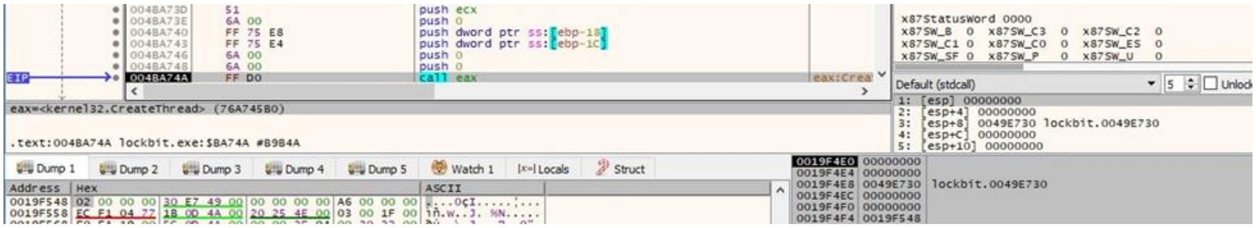


Figure 101

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

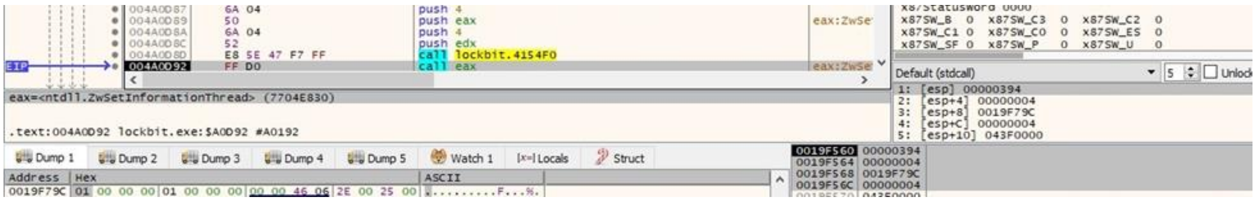


Figure 102

GetLogicalDrives is used to retrieve the available disk drives:

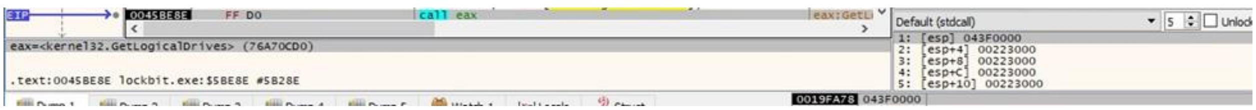


Figure 103

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:

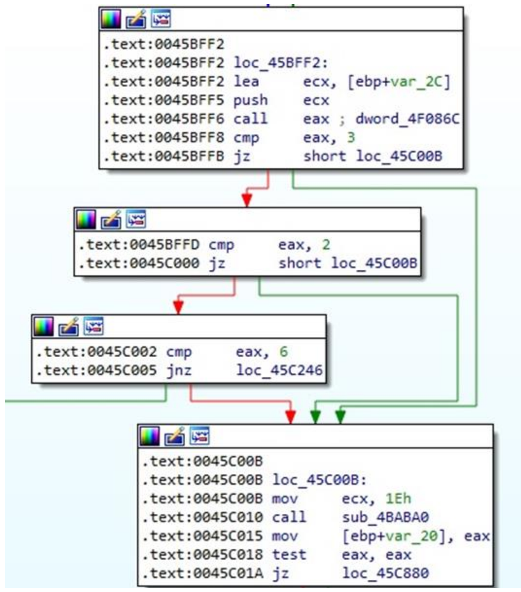


Figure 105

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

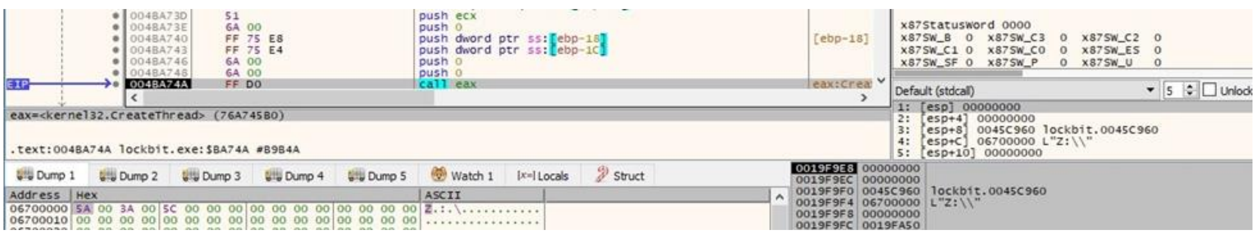


Figure 106

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

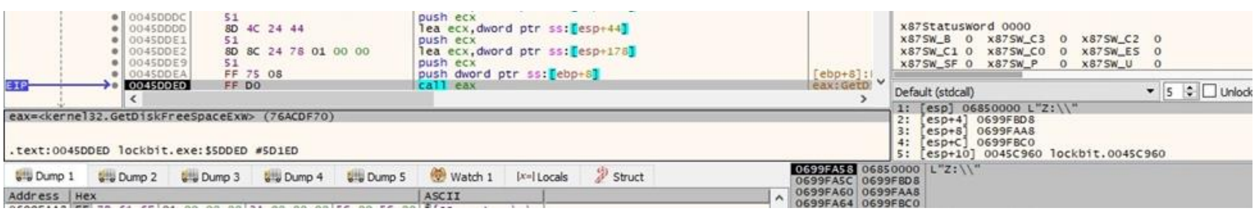


Figure 111

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:

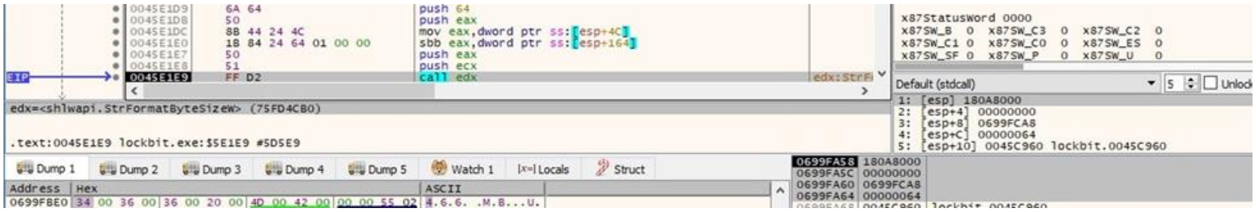


Figure 113

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 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**):

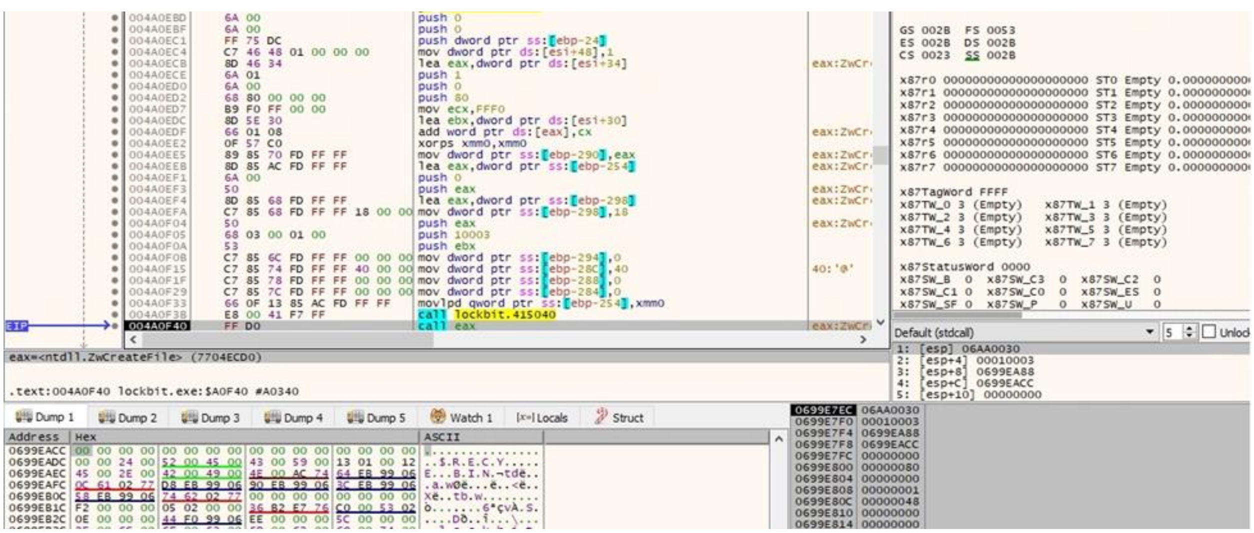


Figure 118

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

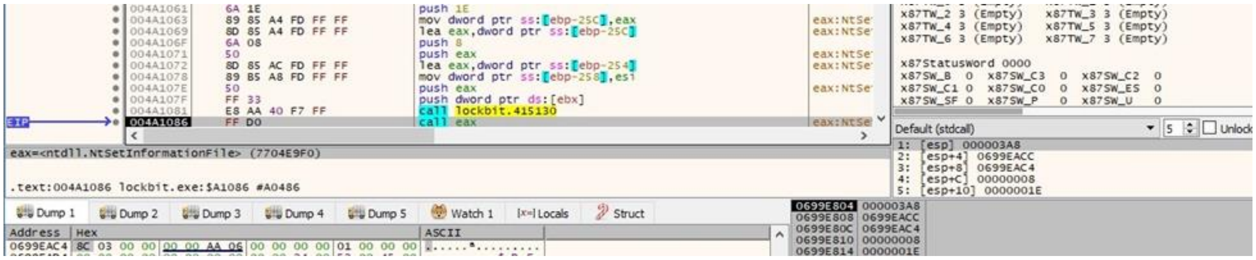


Figure 119

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**):

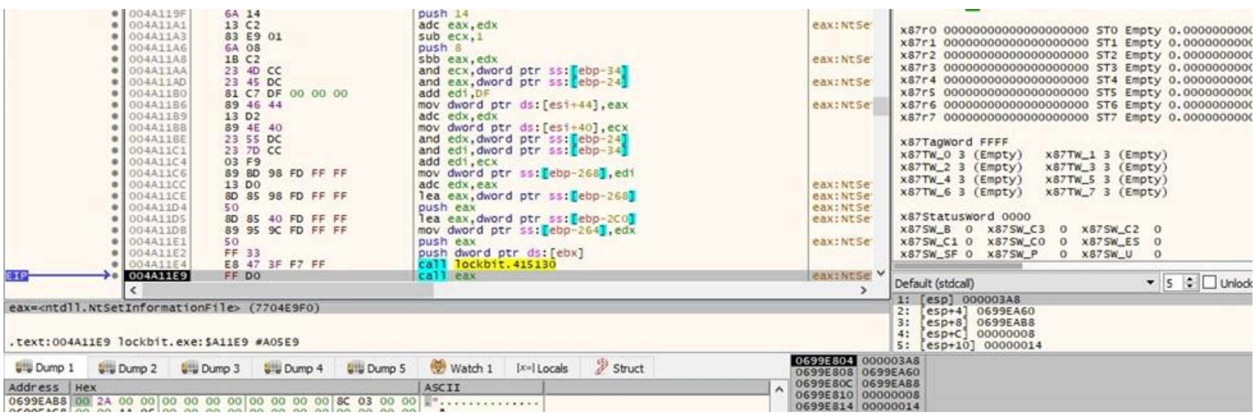


Figure 121

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" ".dacpac" ".zipx" ".lzma"

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

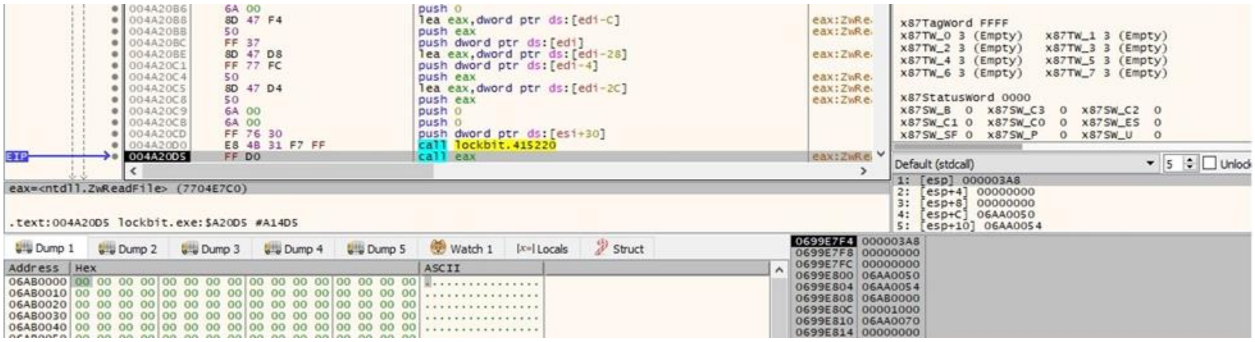


Figure 122

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**):

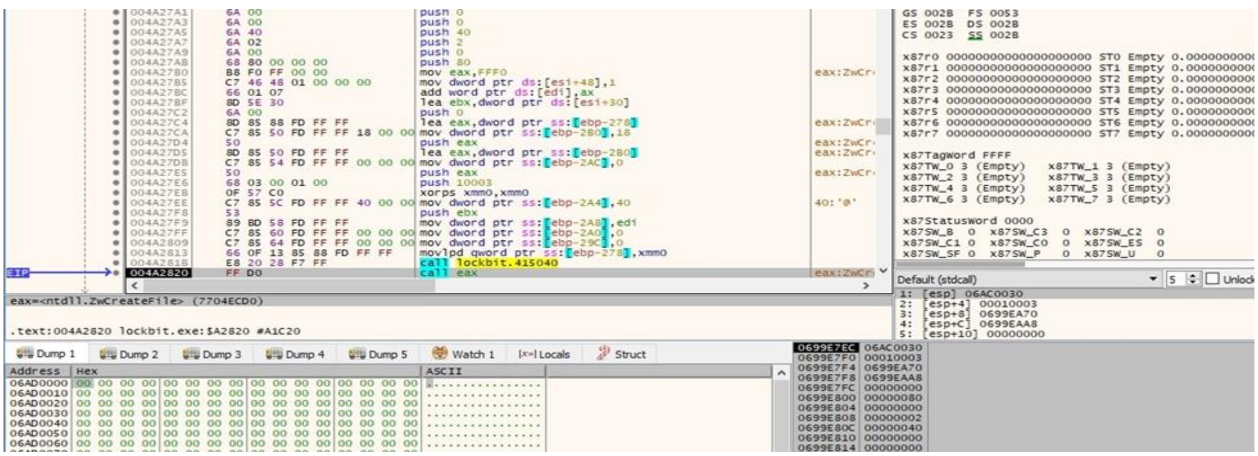


Figure 124

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:

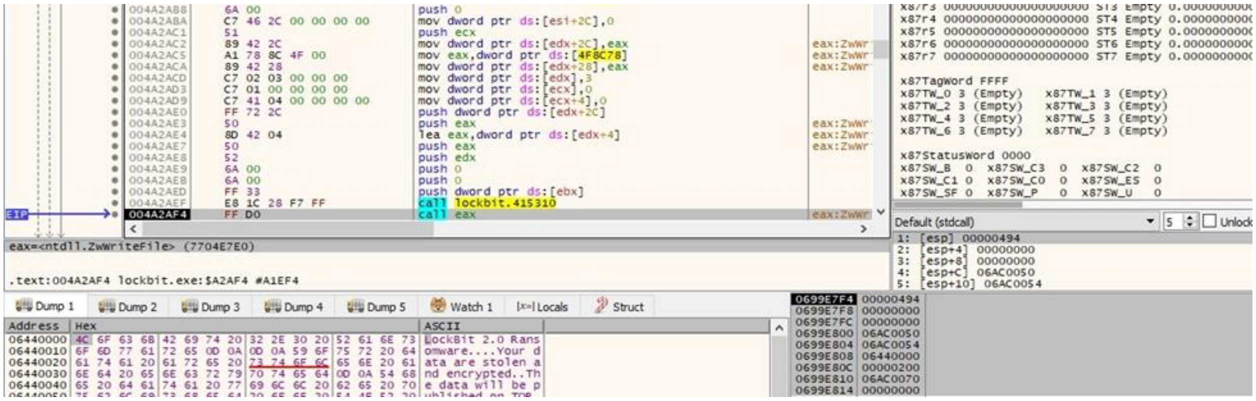


Figure 126

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

Figure 127

The content of the ransom note is displayed below:

```
LockBit 2.0 Ransomware

Your data are stolen and encrypted
The data will be published on TOR website http://lockbitapt6vx57t3eeqjofwgcglmutr3a35nygvokja5uuccip4ykyd.onion and https://bigblog.at if you do not pay the ransom
You can contact us and decrypt one file for free on these TOR sites
http://lockbitsup4yezcd5enk5unncx3zcy7kw6wllyqmiyhvanjj352jayid.onion
http://lockbitsap2oaqhcun3syvbqt6n5nzt7fqosc6jdlmsfleu3ka4k2did.onion
OR
https://decoding.at

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:

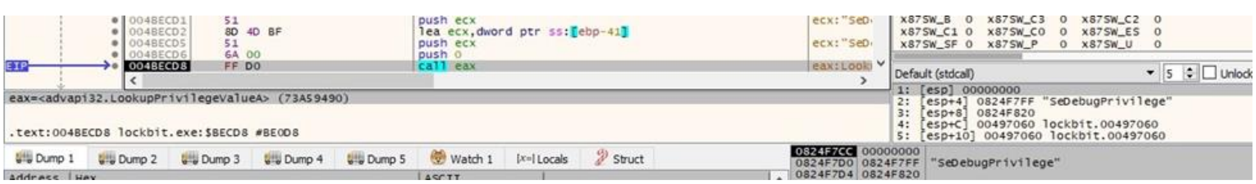


Figure 129

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**):

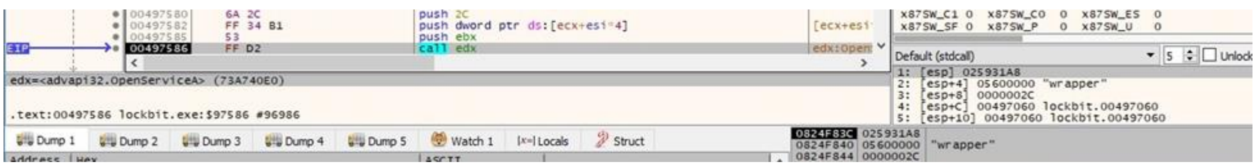


Figure 132

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**):

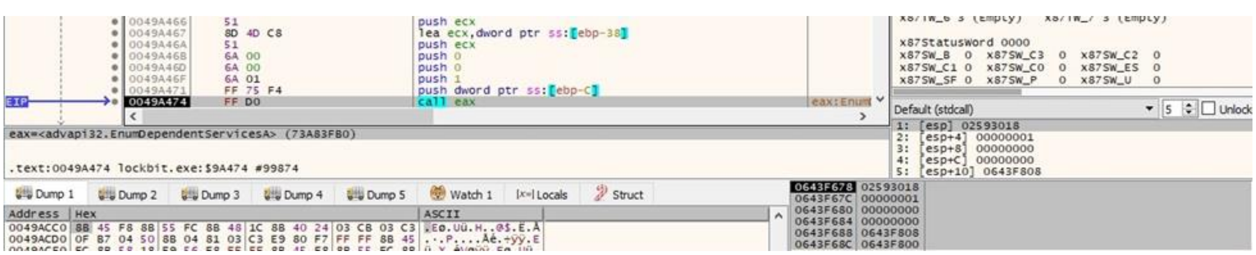


Figure 134

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:

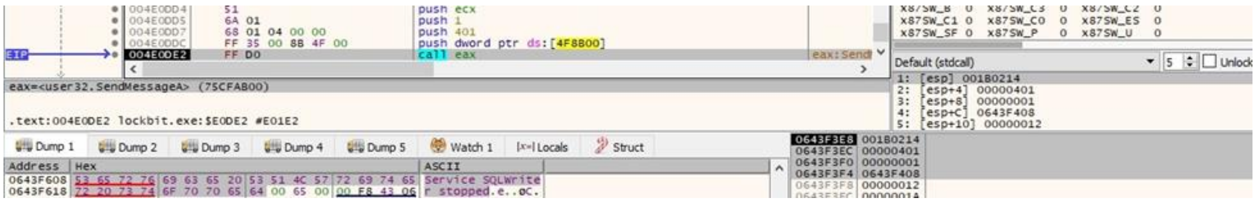


Figure 136

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:

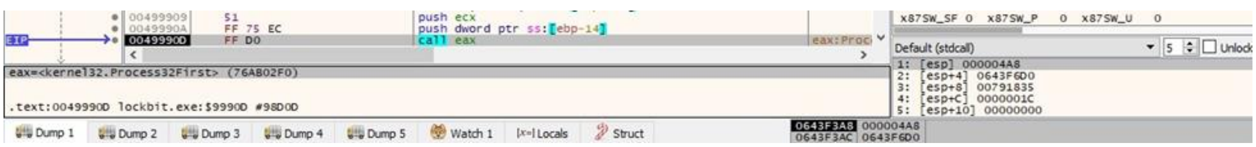


Figure 138

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.

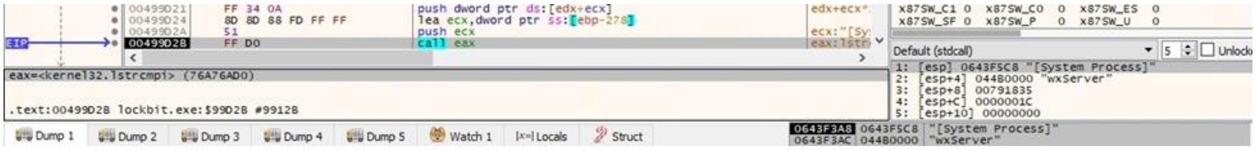


Figure 140

The process enumeration continues by calling the Process32Next routine:

Figure 141

OpenProcess is used to open a targeted process (0x1FFFFF = **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_OLE1DDE**):

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

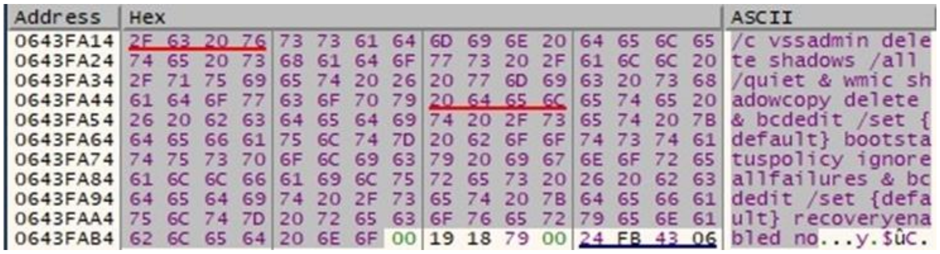


Figure 146

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**):

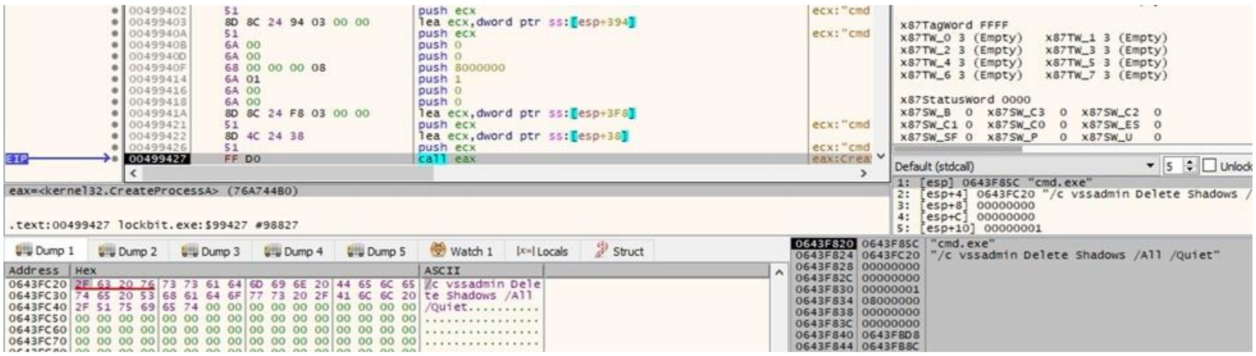


Figure 147

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:

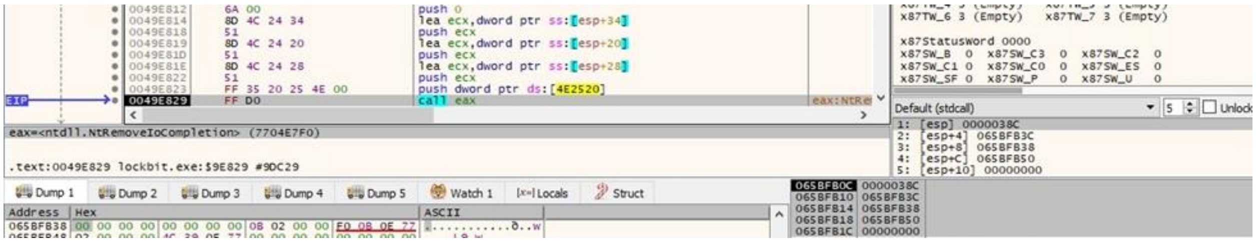


Figure 149

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 .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
065BFE40  BC 77 43 88 2F F4 A2 C0 63 3B F3 6B 85 17 37 FC  ¼wC./ô¢Àc;ók..7ü
065BFE50  4D ED F3 1F 62 19 51 DF 01 22 A2 B4 84 35 95 48  Míó.b.Qß."¢´.5.H
065BFE60  D9 C7 A1 40 BB DE F0 9F BA FC 52 2B 3E C9 C7 63  ÙÇ¡@»Þð.ºüR+>ÉÇc
065BFE70  00 01 5A F2 BB DF AA 6D 01 23 F8 46 3F EA 3F 25  ..Zò»ßªm.#øF?ê?%
065BFE80  8F 74 65 87 34 AB CF EA 35 88 37 AC 0A 62 08 89  .te.4«Ïê5.7¬.b..
065BFE90  35 44 C2 E0 01 EF 0D 0A 34 67 3A A6 3E 05 32 2F  5DÂà.ï..4g:¦>.2/
065BFEA0  7E 67 D7 52 7F 88 DA 58 4B EF E0 FE 75 EA D2 D1  ~g×R..ÚXKïàþuêÒÑ
065BFEB0  B9 D2 E9 CF C6 5A 33 97 8D B5 D3 69 F8 5F 01 B8  ¹ÒéÏÆZ3..µÓiø_.¸
065BFEC0  F6 AE 85 8E 30 F4 B6 19 BD 41 65 70 45 1E 64 C8  ö®..0ô¶.½AepE.dÈ
065BFED0  9F ED 6D E0 AF 19 DB F9 12 58 BE 89 57 46 DA 41  .ímà¯.Ûù.X¾.WFÚA
065BFEE0  F3 BA EE BB 5C A3 35 42 4E FB 8B CB 19 BD 51 8A  óºî»\£5BNû.Ë.½Q.
```

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

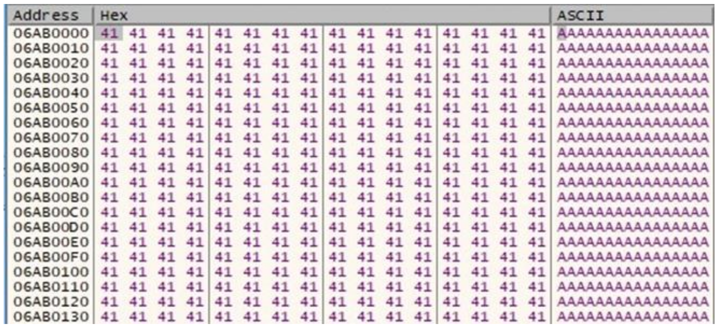


Figure 153

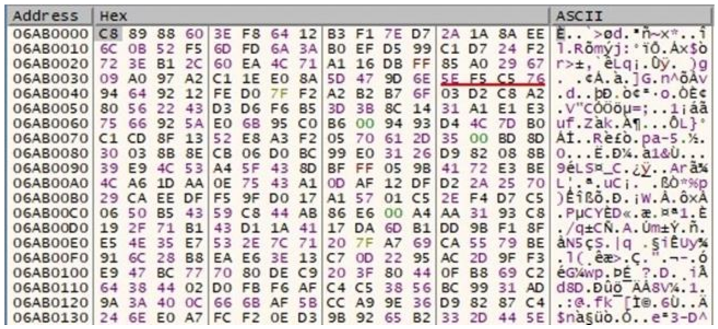


Figure 154

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

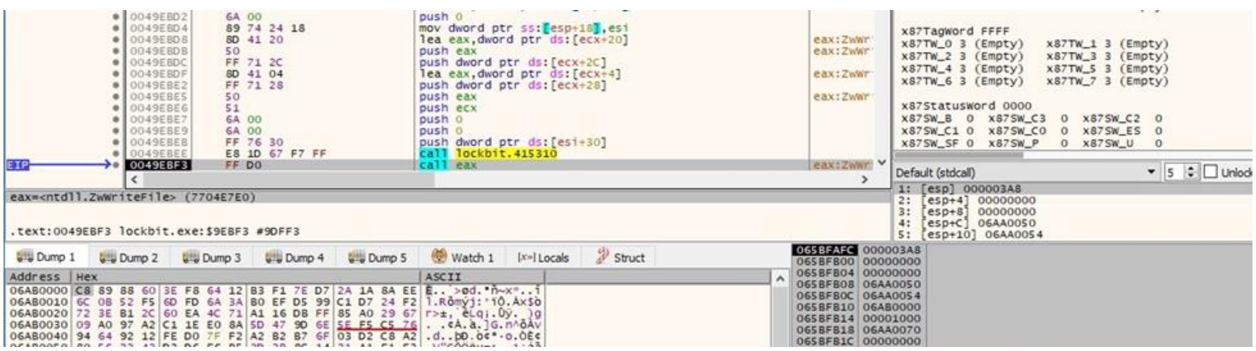


Figure 155

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).

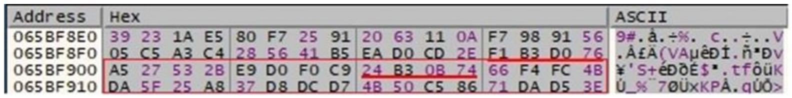


Figure 157

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:

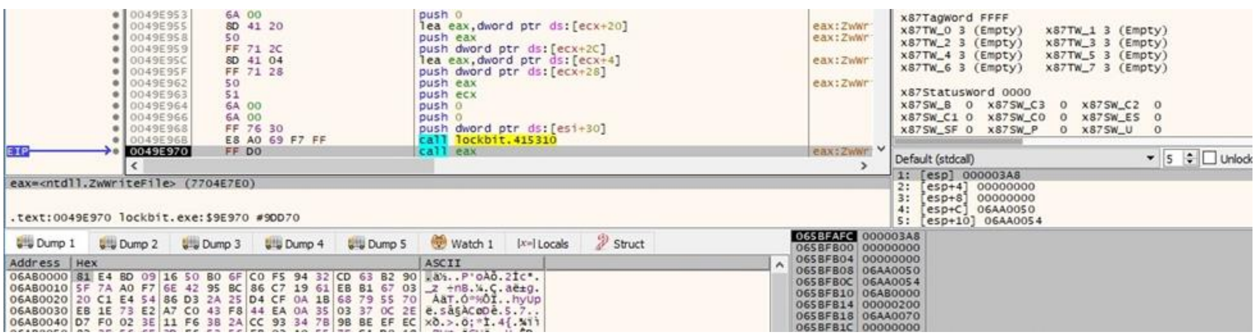


Figure 159

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

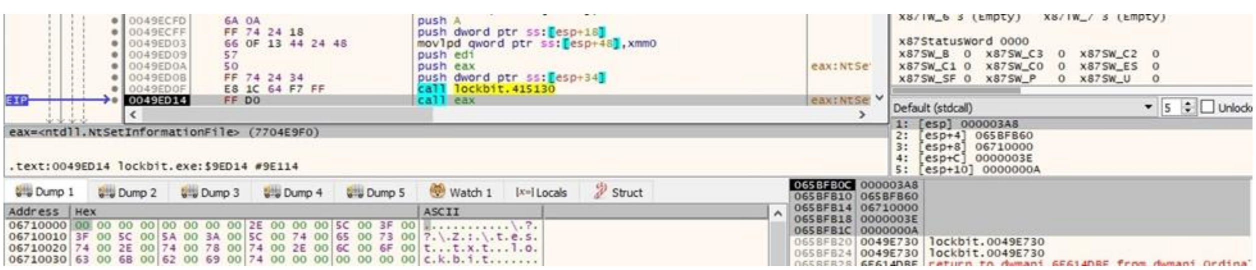


Figure 160

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

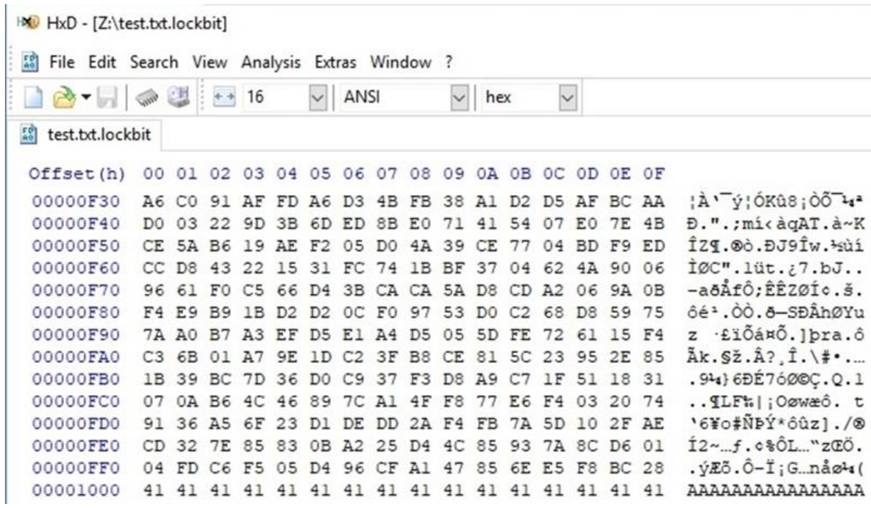


Figure 161

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

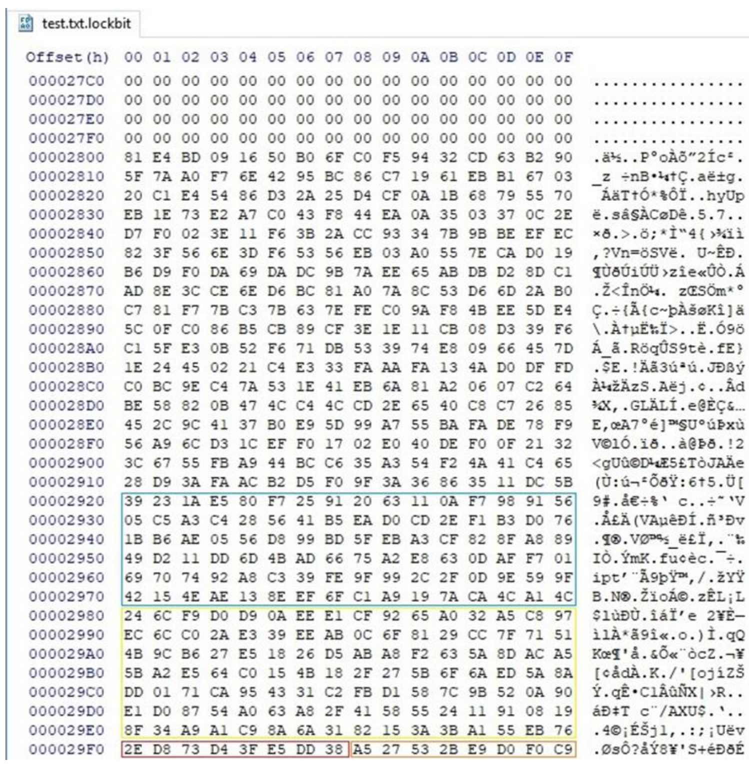


Figure 162

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

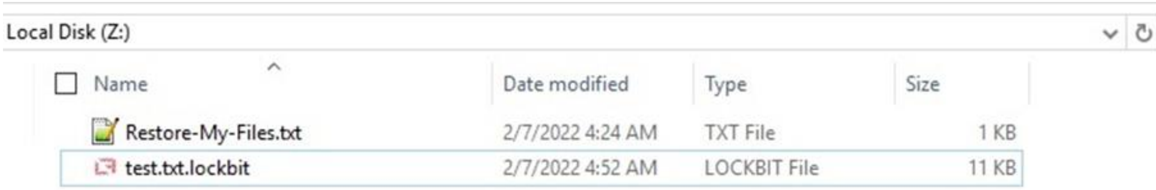


Figure 163

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**):

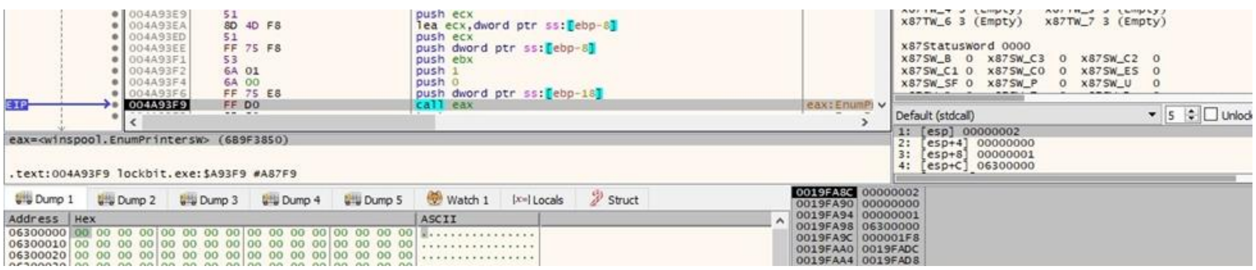


Figure 164

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:

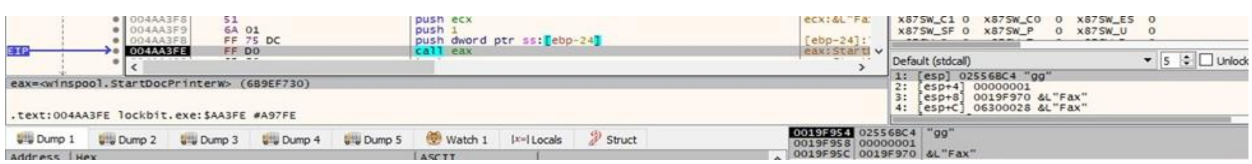


Figure 166

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:

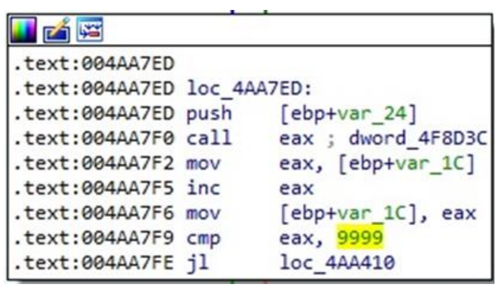


Figure 170

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**):

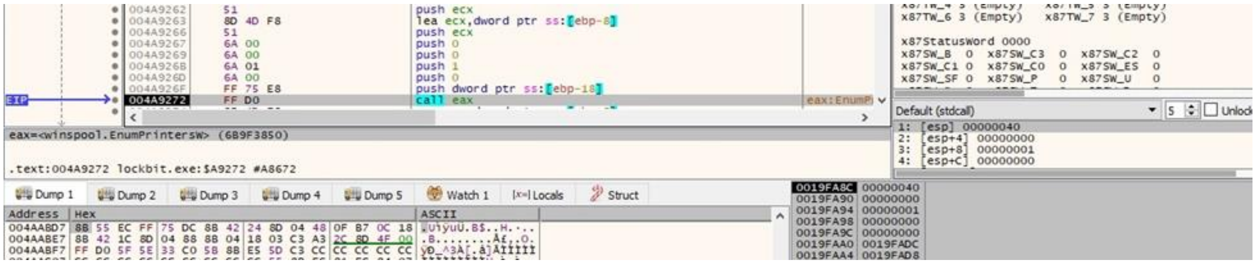


Figure 171

Figure 172

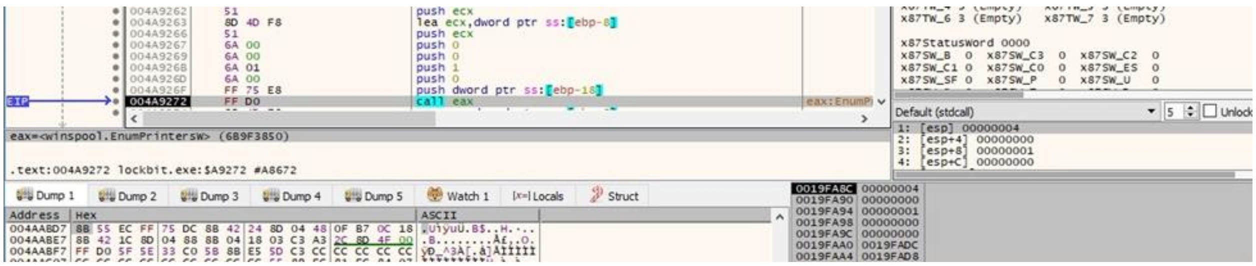


Figure 173

LockBit Wallpaper Setup

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

The GdiplusStartup API is utilized to initialize Windows GDI+:

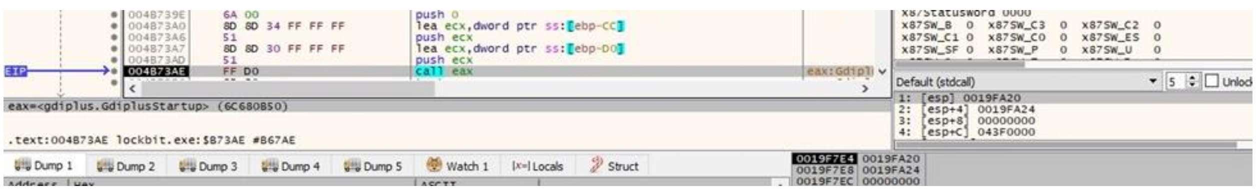


Figure 174

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

Figure 175

The malware allocates memory for Windows GDI+ objects using GdipAlloc:

Figure 176

A Bitmap object is created based on an array of bytes by calling the GdipCreateBitmapFromScan0 function (0x26200a = **PixelFormat32bppARGB**):

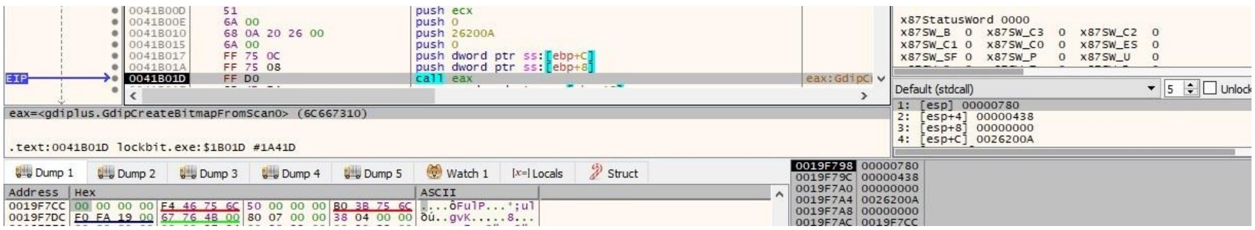


Figure 177

CreateStreamOnHGlobal is utilized to create a stream object:

Figure 178

The binary creates a Bitmap object based on the above stream using GdipCreateBitmapFromStream:

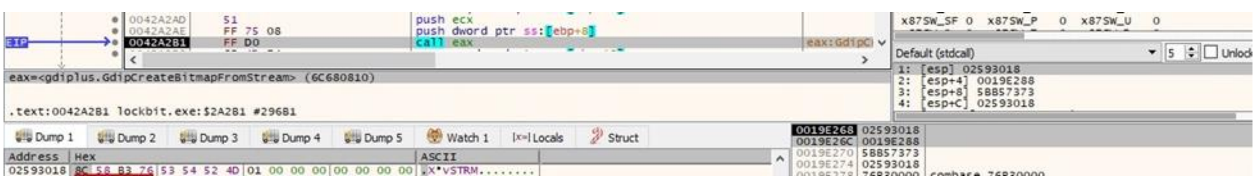


Figure 179

A new private font collection is created via a call to GdipNewPrivateFontCollection:

Figure 180

The malicious process adds a memory font to the private font collection:

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 GdipAddPathArcI 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:

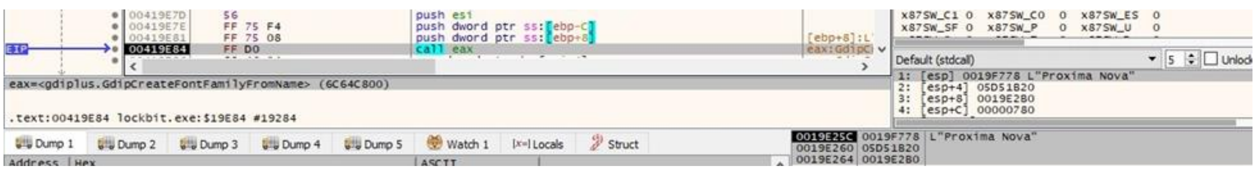


Figure 188

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:

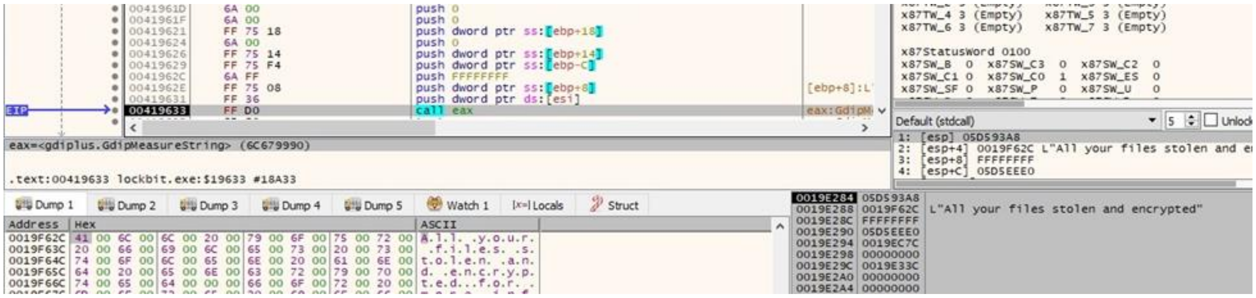


Figure 191

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

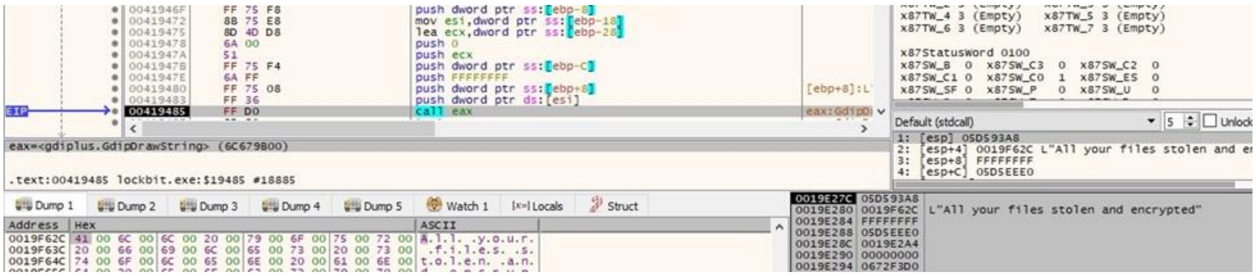


Figure 192

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

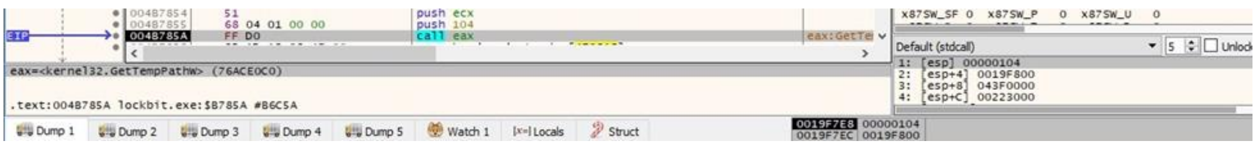


Figure 193

GetTempFileNameW is utilized to create a temporary file:

Figure 194

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

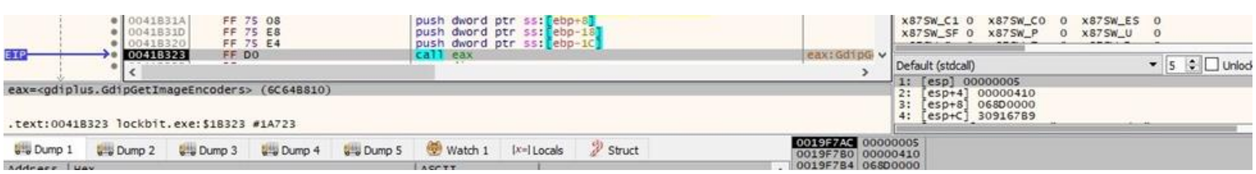


Figure 195

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**):

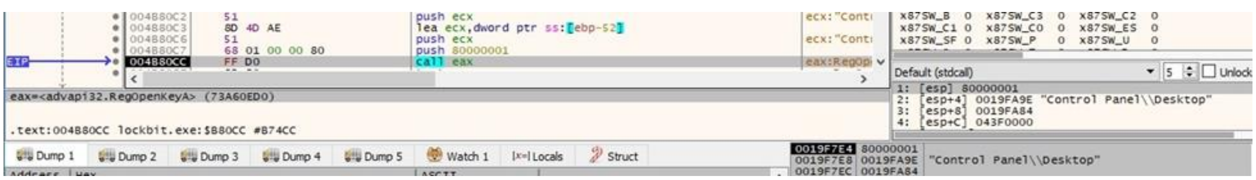


Figure 198

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:

TileWallpaper	REG_SZ	0
TranscodedImageCache	REG_BINARY	7a c3 01 00 36 90 7e 00 80 07 00 00 38 04 00 00 33 b1 88 97 26 1c d8 01 43 00 3a 00 5c 00 55 00 73 00 65 00 72...
TranscodedImageCount	REG_DWORD	0x00000001 (1)
UserPreferencesMask	REG_BINARY	9e 1e 07 80 12 00 00 00
WallPaper	REG_SZ	C:\Users\[redacted]\AppData\Local\Temp\B397.tmp.bmp
WallpaperOriginX	REG_DWORD	0x00000000 (0)
WallpaperOriginY	REG_DWORD	0x00000000 (0)
WallpaperStyle	REG_SZ	2

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:

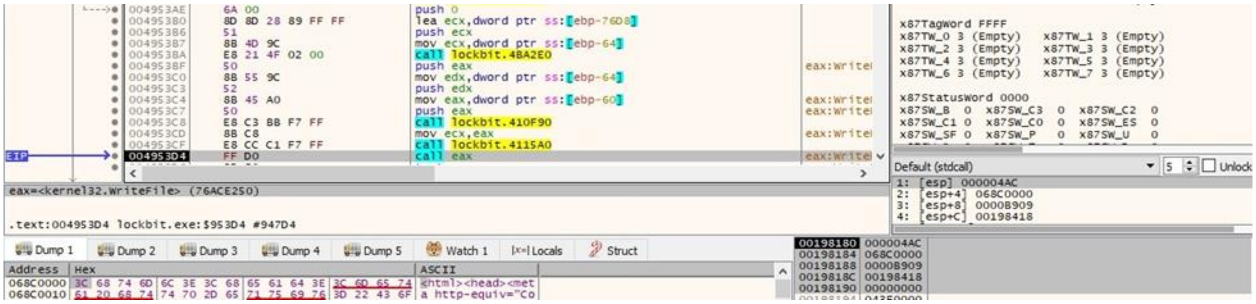


Figure 204

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**):

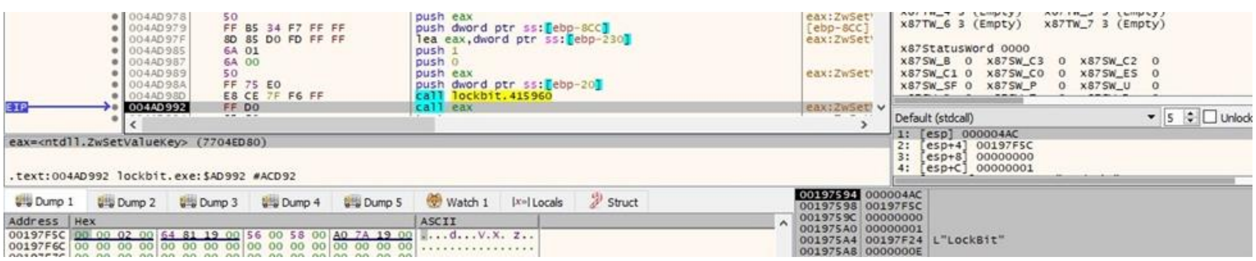


Figure 206

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

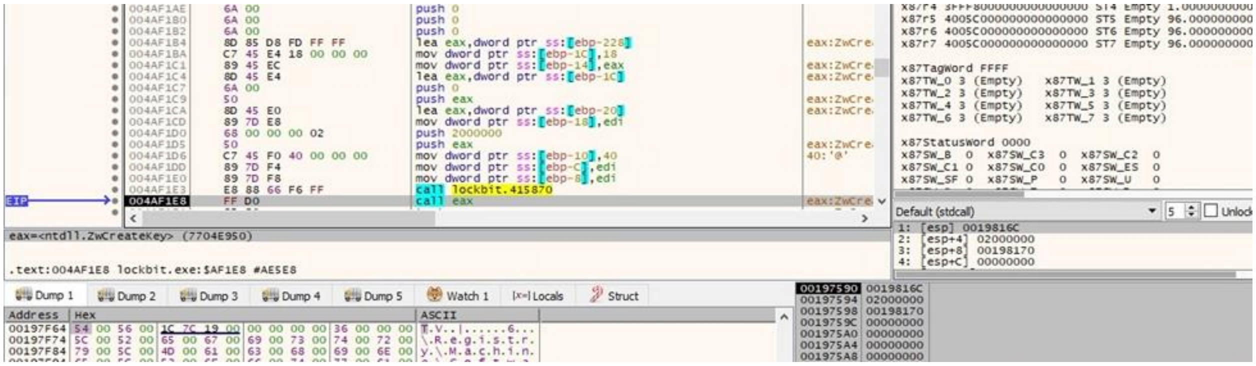


Figure 207

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

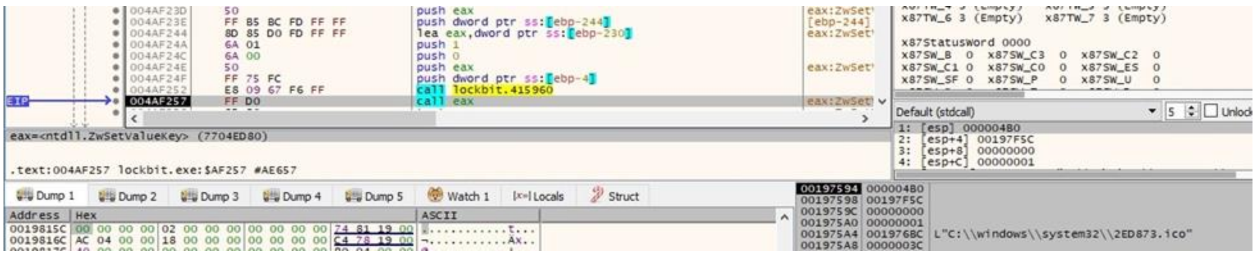


Figure 208

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**):

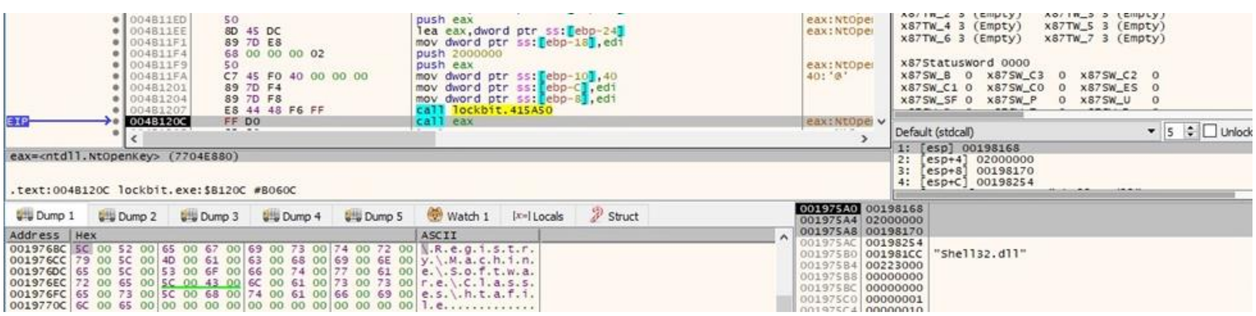


Figure 214

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

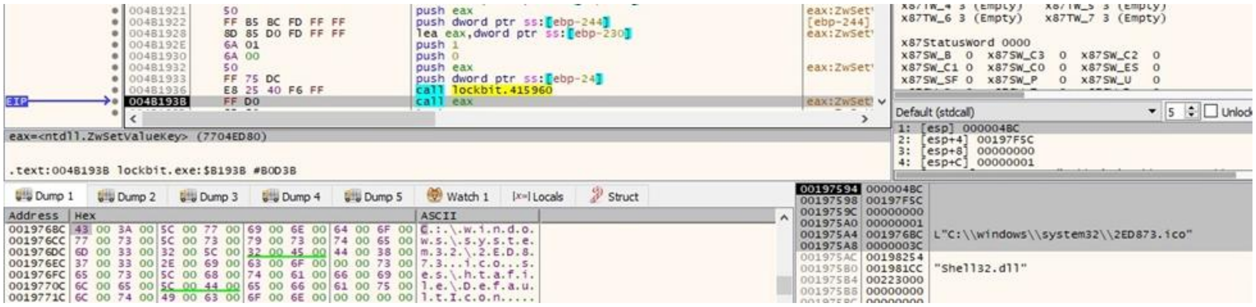


Figure 215

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

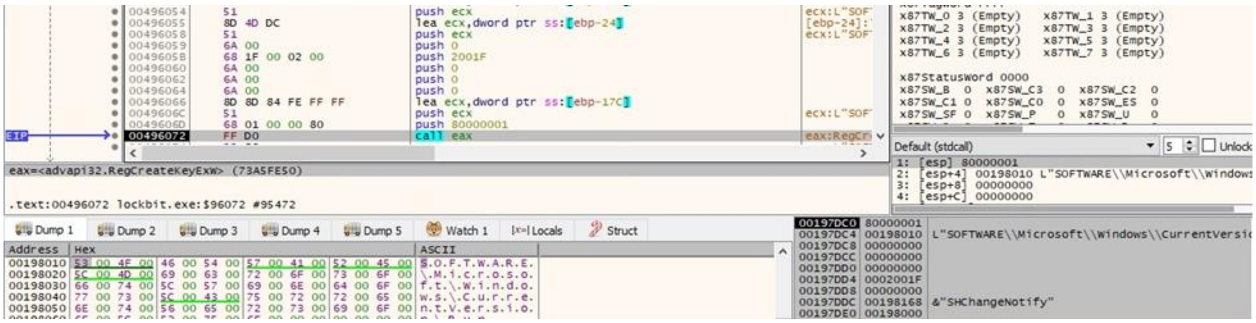


Figure 216

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

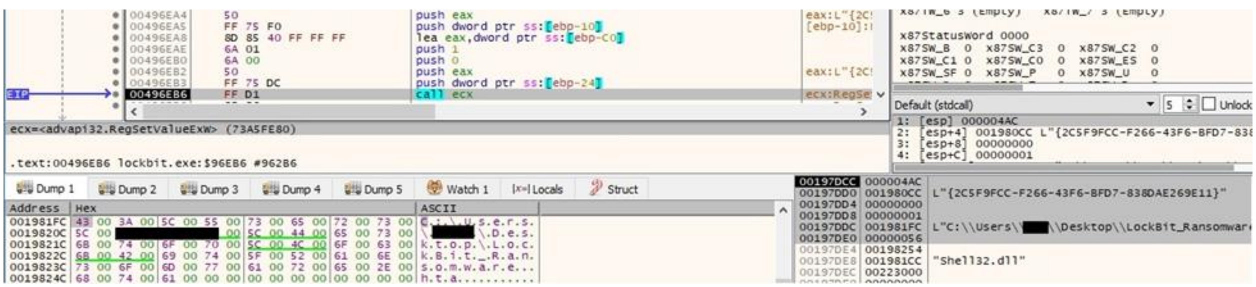


Figure 217

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

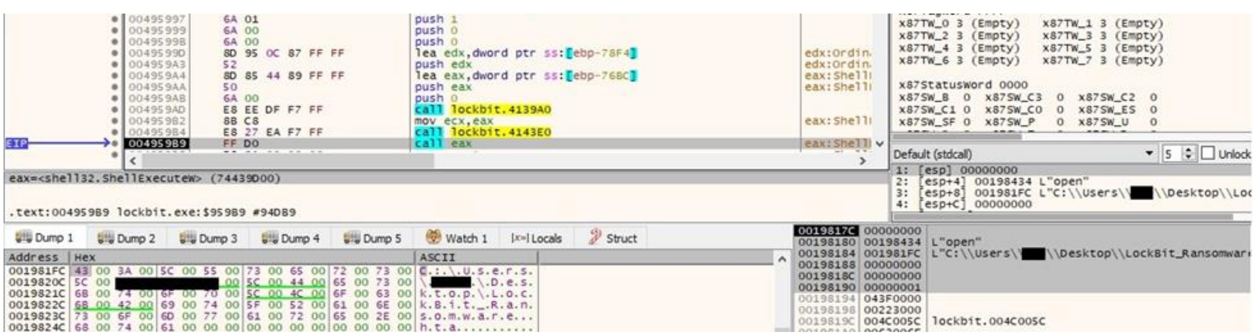


Figure 218

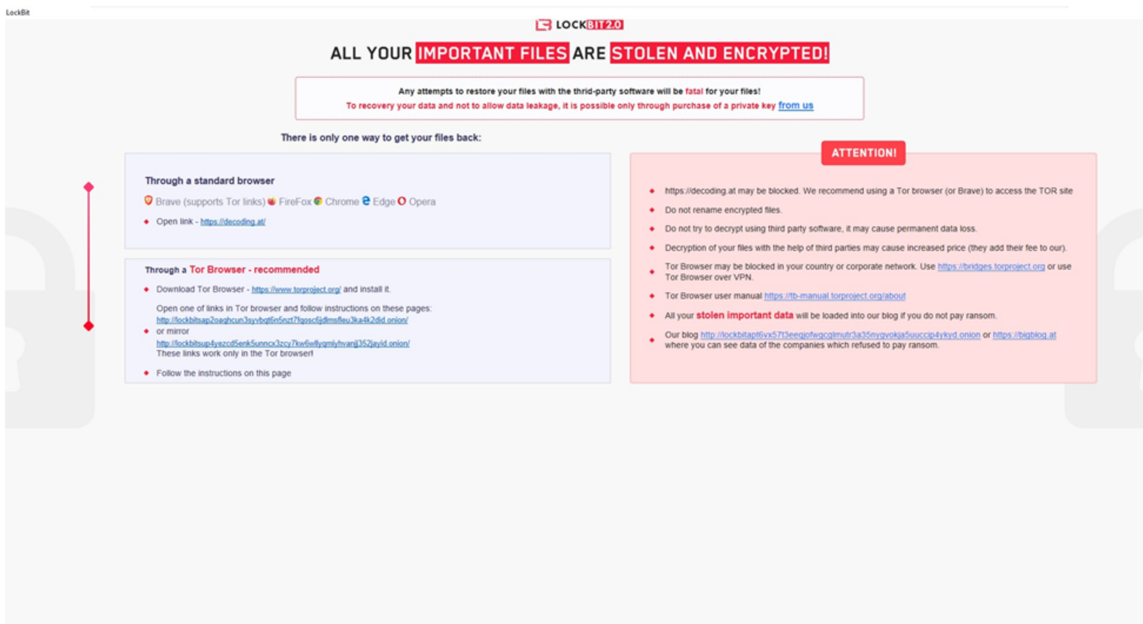


Figure 219

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:

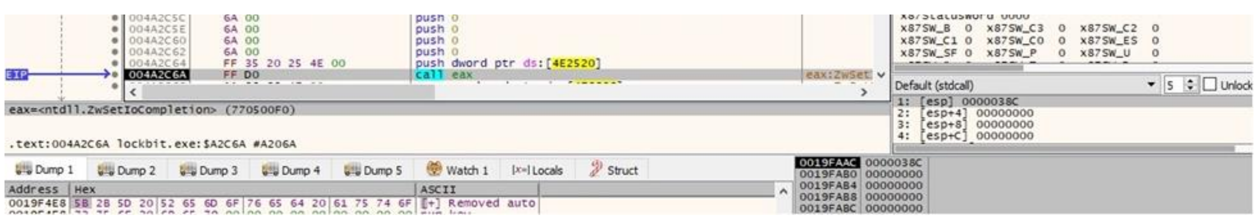


Figure 221

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

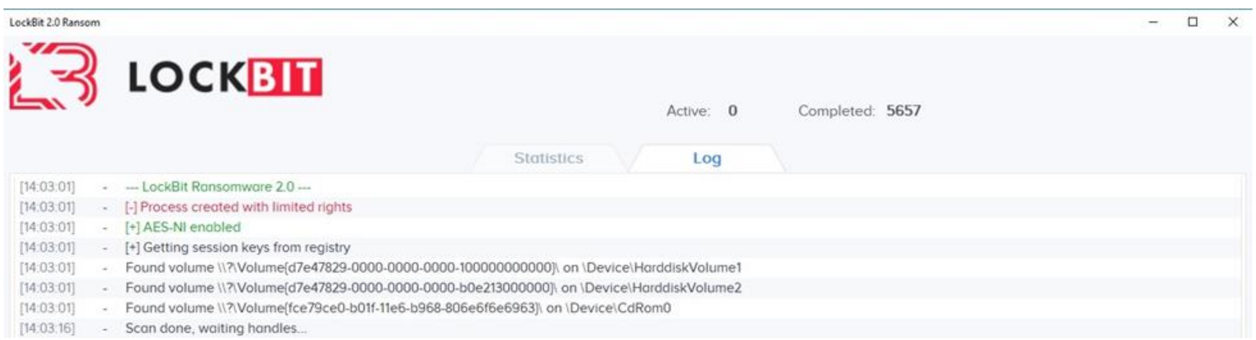


Figure 224

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 wdswfsafe fdhost GDscan ZhuDongFangYu QBDBMgrN mysqld AutodeskDesktopApp acwebbrowser Creative Cloud Adobe Desktop Service CoreSync Adobe CEF Helper node AdobeIPCBroker 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 xfssvccon 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 ADExplorer ADExplorer64 ADExplorer64a 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 CAARCUpdateSvc WSBExchange MSExchange MSExchange$