A Deep Dive into AvosLocker Ransomware

SecurityScorecard.com

info@securityscorecard.com

LIFARS

a SecurityScorecard company

244 Fifth Avenue, Suite 2035,

New York, NY 10001

1.212.222.7061

Table of Contents

Executive Summary

AvosLocker is a ransomware-as-a-service (RaaS) group that appeared in 2021. The malware can run with one of the following parameters: “--help”, “--path”, “--disabledrives”, “--hide”, “--threads”, “--enablesmb”, “--brutesmb”, and “--nomutex.” The ransomware kills a list of targeted processes, deletes all Volume Shadow Copies using two commands, and clears all Windows event logs. The binary can target the logical drives as well as network shares by specifying proper arguments.

The encryption is done using multithreading with I/O completion ports. AvosLocker uses a combination of RSA and Salsa20 algorithms during the encryption process. Finally, the ransomware creates an image based on the ransom note text that is set as the Desktop Wallpaper.

Analysis and Findings

SHA256: EC955F589F25D0D28E55964A1AA79C27492026982994CD4CA1FAF7E8A78DB4BC

The malware performs a call to GetCurrentProcess and then opens the access token associated with the current process using the OpenProcessToken API (0xF01FF = **TOKEN_ALL_ACCESS**):

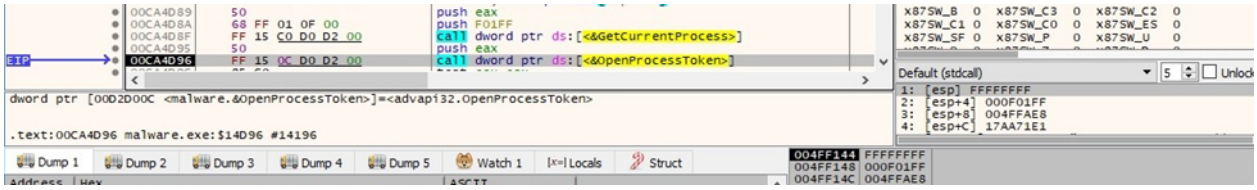


Figure 1

Most of the strings are encrypted using the XOR operator. An example of a decryption algorithm is displayed in figure 2:

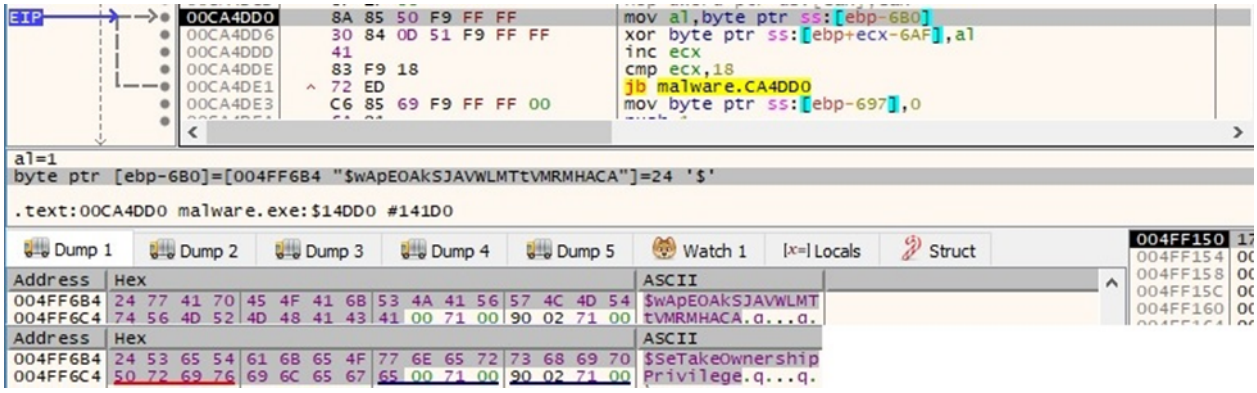


Figure 2

The LookupPrivilegeValueA function is utilized to retrieve the LUID (locally unique identifier) corresponding to the "SeTakeOwnershipPrivilege" privilege:

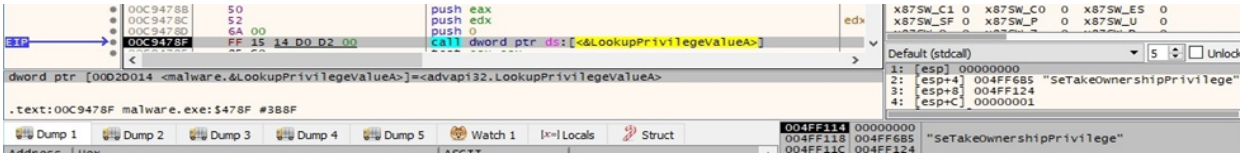


Figure 3

AvosLocker enables the above privilege in the access token via a function call to AdjustTokenPrivileges:

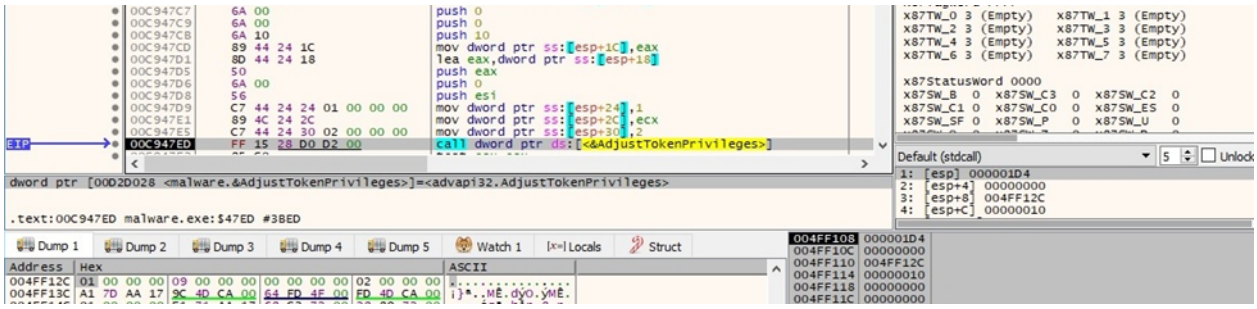
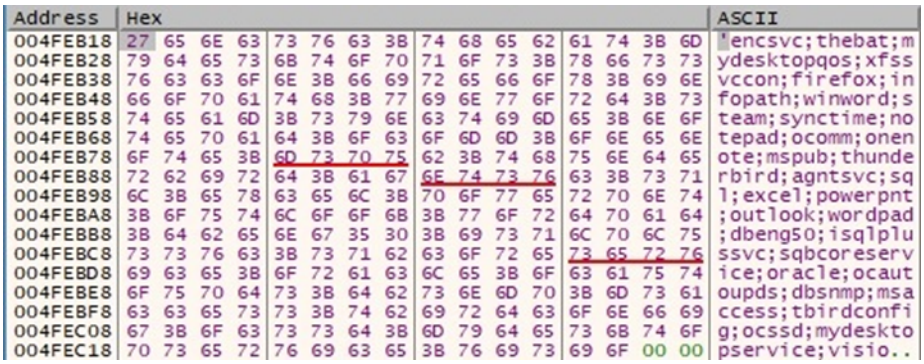


Figure 4

The binary decrypts a list of processes that will be killed (figure 5):

- "encsvc" "thebat" "mydesktopqos" "xfssvccon" "firefox" "infopath" "winword" "steam" "synctime" "notepad"
- "ocomm" "onenote" "mspub" "thunderbird" "agntsvc" "sql" "excel" "powerpnt" "outlook" "wordpad" "dbeng50"
- "isqlplussvc" "sqbcoreservice" "oracle" "ocautoupds" "dbsnmp" "msaccess" "tbirdconfig" "ocssd" "mydesktopservice" "visio"



Address	Hex	ASCII
004FEB18	27 65 6E 63 73 76 63 3B 74 68 65 62 61 74 3B 6D	'encsvc;thebat;m
004FEB28	79 64 65 73 6B 74 6F 70 71 6F 73 3B 78 66 73 73	ydesktopqos;xfss
004FEB38	76 63 63 6F 6E 3B 66 69 72 65 66 6F 78 3B 69 6E	vccon;firefox;in
004FEB48	66 6F 70 61 74 68 3B 77 69 6E 77 6F 72 64 3B 73	fopath;winword;s
004FEB58	74 65 61 6D 3B 73 79 6E 63 74 69 6D 65 3B 6E 6F	team;synctime;no
004FEB68	74 65 70 61 64 3B 6F 63 6F 6D 6D 3B 6F 6E 65 6E	tepad;ocomm;onen
004FEB78	6F 74 65 3B 6D 73 70 75 62 3B 74 68 75 6E 64 65	ote;mspub;thunde
004FEB88	72 62 69 72 64 3B 61 67 6E 74 73 76 63 3B 73 71	rbird;agntsvc;sq
004FEB98	6C 3B 65 78 63 65 6C 3B 70 6F 77 65 72 70 6E 74	l;excel;powerpnt
004FEBA8	3B 6F 75 74 6C 6F 6F 6B 3B 77 6F 72 64 70 61 64	;outlook;wordpad
004FEBB8	3B 64 62 65 6E 67 35 30 3B 69 73 71 6C 70 6C 75	;dbeng50;isqlplu
004FEBC8	73 73 76 63 3B 73 71 62 63 6F 72 65 73 65 72 76	ssvc;sqbcoreserv
004FEBD8	69 63 65 3B 6F 72 61 63 6C 65 3B 6F 63 61 75 74	ice;oracle;ocaut
004FEBE8	6F 75 70 64 73 3B 64 62 73 6E 6D 70 3B 6D 73 61	oupds;dbsnmp;msa
004FEBF8	63 63 65 73 73 3B 74 62 69 72 64 63 6F 6E 66 69	ccess;tbirdconfi
004FEC08	67 3B 6F 63 73 73 64 3B 6D 79 64 65 73 6B 74 6F	g;ocssd;mydeskto
004FEC18	70 73 65 72 76 69 63 65 3B 76 69 73 69 6F 00 00	pservice;visio..

Figure 5

CreateToolhelp32Snapshot is used to take a snapshot of all processes in the system (0x2 = **TH32CS_SNAPPROCESS**):

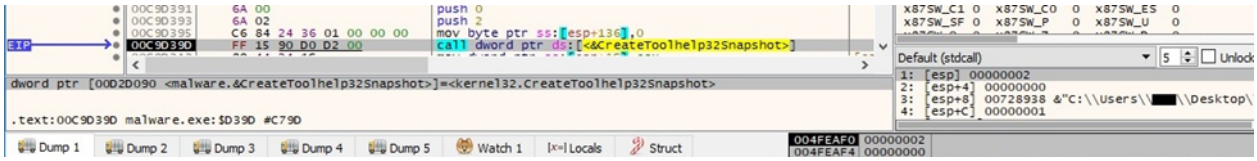


Figure 6

The ransomware extracts information about the first process from the snapshot using the Process32First routine:

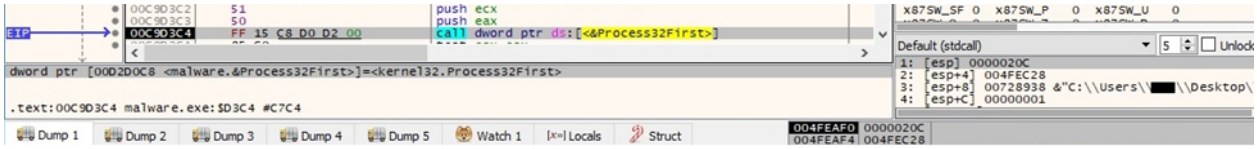


Figure 7

There is a comparison between the process name and the blacklisted processes:

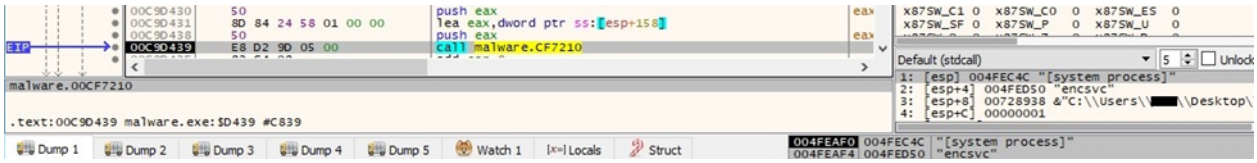


Figure 8

The malicious binary retrieves information about the next process from the snapshot via a call to Process32Next:

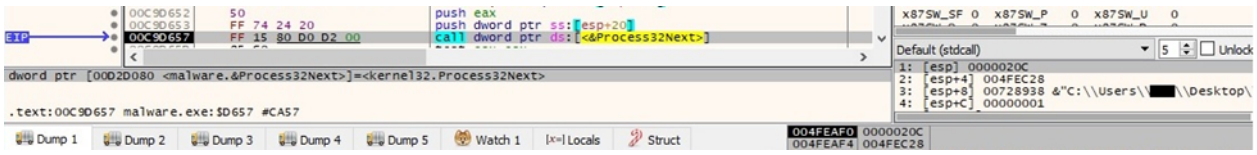


Figure 9

AvosLocker uses the FNV (Fowler-Noll-Vo) hashing algorithm to identify and call relevant APIs at runtime:

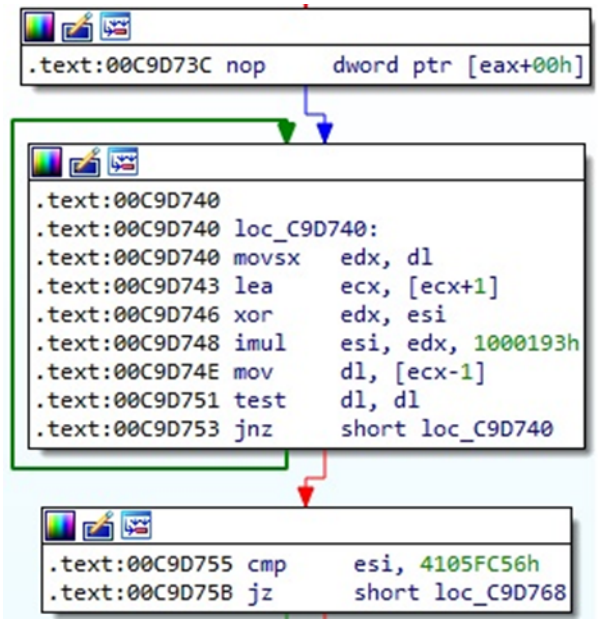


Figure 10

The executable opens a targeted process using OpenProcess (0x1 = **PROCESS_TERMINATE**):

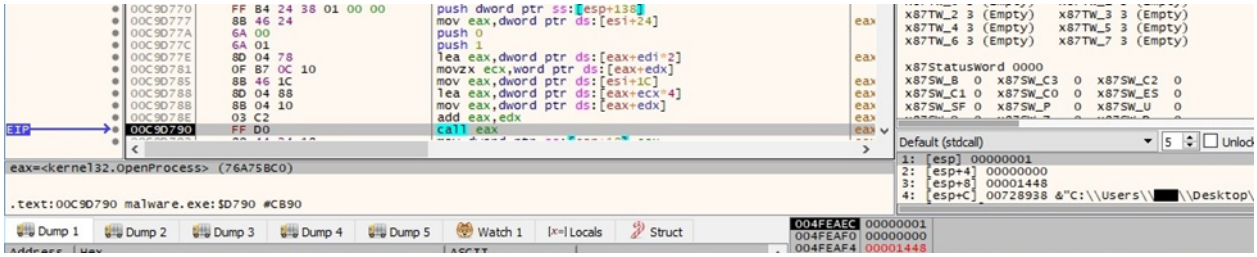


Figure 11

The TerminateProcess routine is utilized to kill the targeted process:

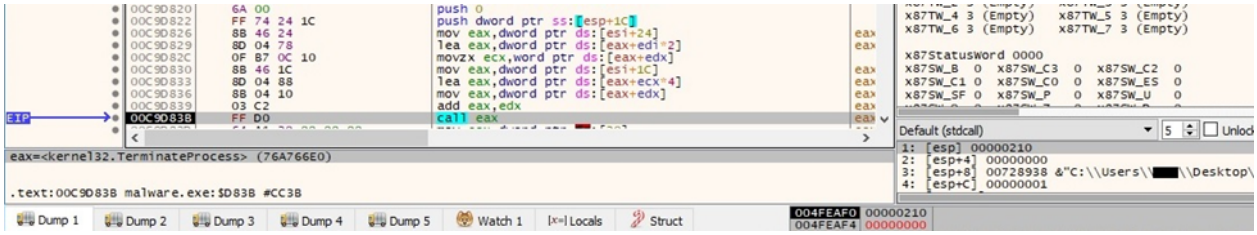


Figure 12

The ransomware writes the following data in the command line output:

Figure 13

We'll explain the purpose of each parameter in the following paragraphs.

The malware creates a mutex called "Zheic0WaWie6zeiy", which ensures that only one copy of the process is running at a single time:

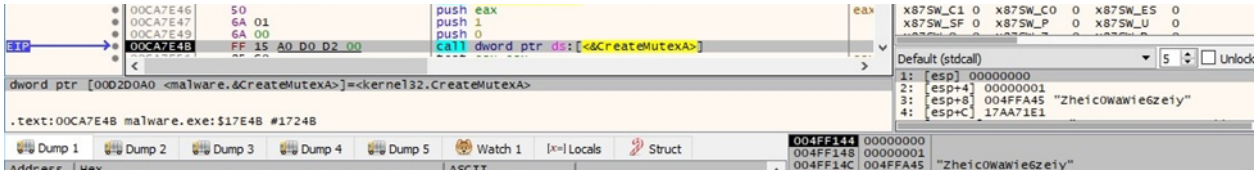


Figure 14

The process disables file system redirection by calling the Wow64DisableWow64FsRedirection API:

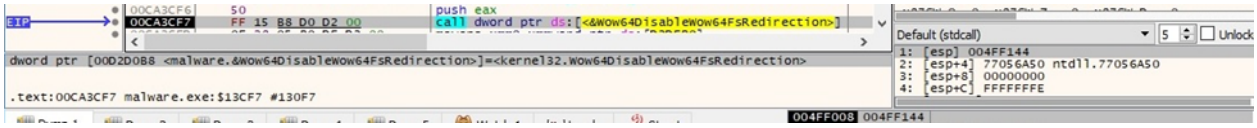


Figure 15

The binary calls the WinExec function in order to spawn multiple processes (figure 16):

- cmd /c wmic shadowcopy delete /nointeractive – delete volume shadow copies
- cmd /c vssadmin.exe Delete Shadows /All /Quiet – delete volume shadow copies
- cmd /c bcdedit /set {default} recoveryenabled No – disable automatic repair
- cmd /c bcdedit /set {default} bootstatuspolicy ignoreallfailures – ignore errors in the case of a failed boot / shutdown / checkpoint
- cmd /c powershell -command \"Get-EventLog -LogName * | ForEach { Clear-EventLog $_.Log }\" - clear all entries from the event logs

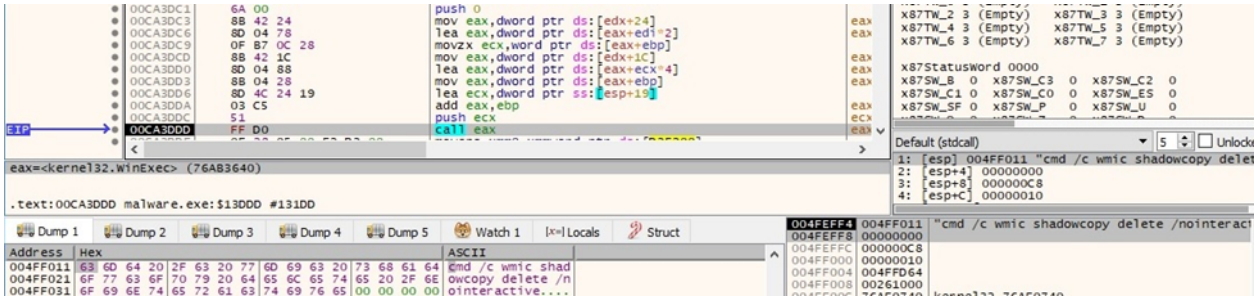
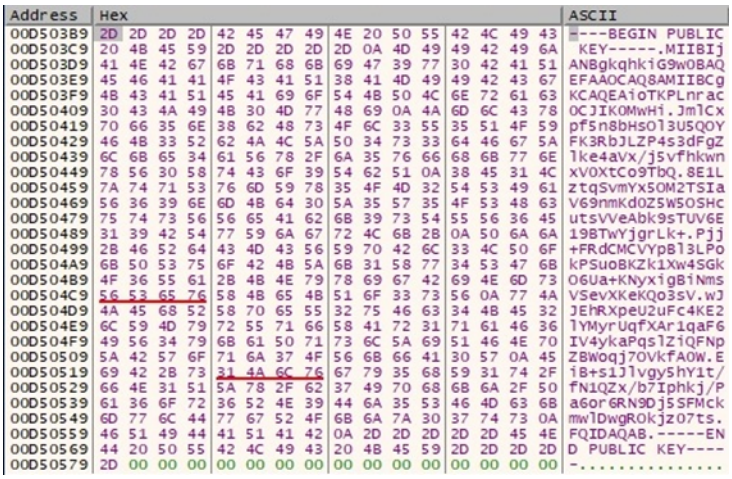


Figure 16

The file comes with a hard-coded RSA public key:



Address	Hex	ASCII
00D503B9	2D 2D 2D 2D 42 45 47 49 4E 20 50 55 42 4C 49 43	-----BEGIN PUBLIC
00D503C9	20 4B 45 59 2D 2D 2D 2D 2D 0A 4D 49 49 42 49 6A	KEY-----.MIIBIj
00D503D9	41 4E 42 67 6B 71 68 6B 69 47 39 77 30 42 41 51	ANBgkqhkiG9w0BAQ
00D503E9	45 46 41 41 4F 43 41 51 38 41 4D 49 49 42 43 67	EFAAOCAQ8AMIIBCg
00D503F9	4B 43 41 51 45 41 69 6F 54 4B 50 4C 6E 72 61 63	KCAQEAioTKPLnrac
00D50409	30 43 4A 49 4B 30 4D 77 48 69 0A 4A 6D 6C 43 78	0CJIK0MwHi.JmlCx
00D50419	70 66 35 6E 38 62 48 73 4F 6C 33 55 35 51 4F 59	pf5n8bHsOl3U5QOY
00D50429	46 4B 33 52 62 4A 4C 5A 50 34 73 33 64 46 67 5A	FK3RbJLZP4s3dFgZ
00D50439	6C 6B 65 34 61 56 78 2F 6A 35 76 66 68 6B 77 6E	lke4aVx/j5vfhkwn
00D50449	78 56 30 58 74 43 6F 39 54 62 51 0A 38 45 31 4C	xV0XtCo9TbQ.8E1L
00D50459	7A 74 71 53 76 6D 59 78 35 4F 4D 32 54 53 49 61	ztqSvmYx5OM2TSIa
00D50469	56 36 39 6E 6D 4B 64 30 5A 35 57 35 4F 53 48 63	V69nmKd0Z5W5OSHc
00D50479	75 74 73 56 56 65 41 62 6B 39 73 54 55 56 36 45	utsVVeAbk9sTUV6E
00D50489	31 39 42 54 77 59 6A 67 72 4C 6B 2B 0A 50 6A 6A	19BTwYjgrLk+.Pjj
00D50499	2B 46 52 64 43 4D 43 56 59 70 42 6C 33 4C 50 6F	+FRdCMCVYpBl3LPo
00D504A9	6B 50 53 75 6F 42 4B 5A 6B 31 58 77 34 53 47 6B	kPSuoBKZk1Xw4SGk
00D504B9	4F 36 55 61 2B 4B 4E 79 78 69 67 42 69 4E 6D 73	O6Ua+KNyxigBiNms
00D504C9	56 53 65 76 58 4B 65 4B 51 6F 33 73 56 0A 77 4A	VSevXKeKQo3sV.wJ
00D504D9	4A 45 68 52 58 70 65 55 32 75 46 63 34 4B 45 32	JEhRXpeU2uFc4KE2
00D504E9	6C 59 4D 79 72 55 71 66 58 41 72 31 71 61 46 36	lYMyrUqfXAr1qaF6
00D504F9	49 56 34 79 6B 61 50 71 73 6C 5A 69 51 46 4E 70	IV4ykaPqslZiQFNp
00D50509	5A 42 57 6F 71 6A 37 4F 56 6B 66 41 30 57 0A 45	ZBWoqj7OVkfA0W.E
00D50519	69 42 2B 73 31 4A 6C 76 67 79 35 68 59 31 74 2F	iB+s1Jlvgy5hY1t/
00D50529	66 4E 31 51 5A 78 2F 62 37 49 70 68 6B 6A 2F 50	fN1QZx/b7Iphkj/P
00D50539	61 36 6F 72 36 52 4E 39 44 6A 35 53 46 4D 63 6B	a6or6RN9Dj5SFMck
00D50549	6D 77 6C 44 77 67 52 4F 6B 6A 7A 30 37 74 73 0A	mwlDwgROkjz07ts.
00D50559	46 51 49 44 41 51 41 42 0A 2D 2D 2D 2D 2D 45 4E	FQIDAQAB.-----EN
00D50569	44 20 50 55 42 4C 49 43 20 4B 45 59 2D 2D 2D 2D	D PUBLIC KEY----
00D50579	2D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	-...............

Figure 17

AvosLocker creates an input/output (I/O) completion port that is not yet associated with a file handle (0xFFFFFFFF = **INVALID_HANDLE_VALUE**):

Figure 18

The malware creates multiple threads that will handle the files encryption. As we can see in figures 19 and 20, even if the starting address of the thread is StartAddress (sub_D0155F), the actual relevant function that will be called is sub_CBBAD0:

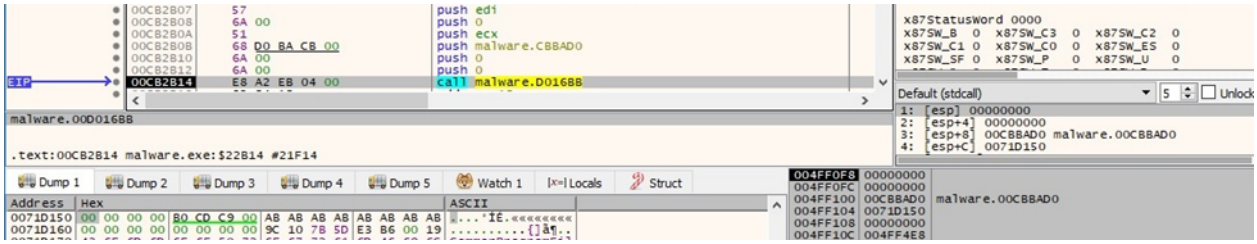


Figure 19

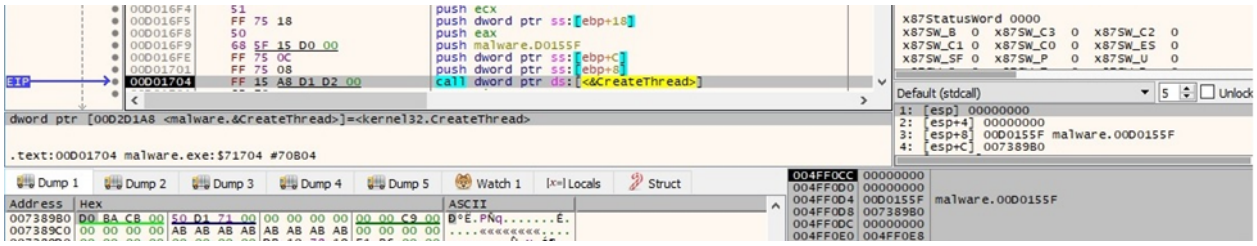


Figure 20

The thread's priority is set to 0x2 (**THREAD_PRIORITY_HIGHEST**) via a function call to SetThreadPriority:

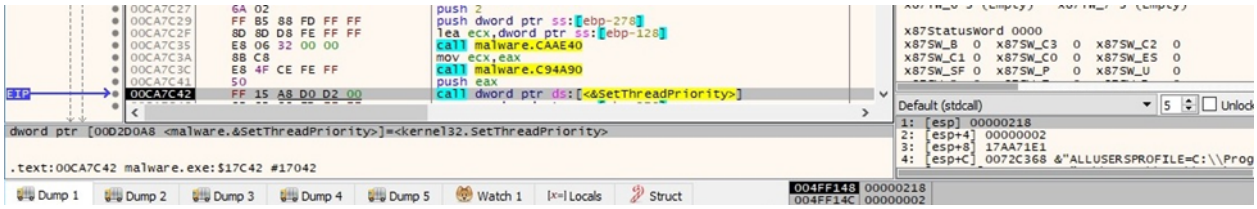


Figure 21

The number of created threads is 200 (default value); however, it can be modified using the -t (or --threads) parameter.

FindFirstVolumeW is utilized to retrieve the first volume of the local machine:

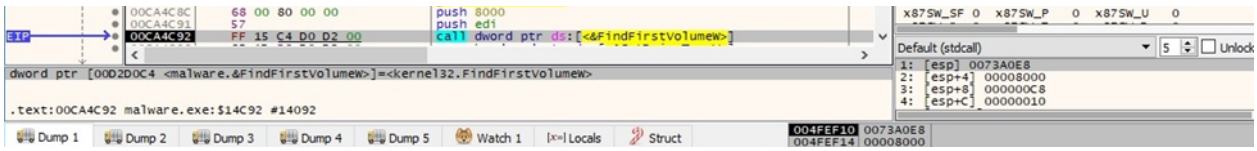


Figure 22

The ransomware extracts a list of drive letters and mounted folder paths for a volume using the GetVolumePathNamesForVolumeNameW function:

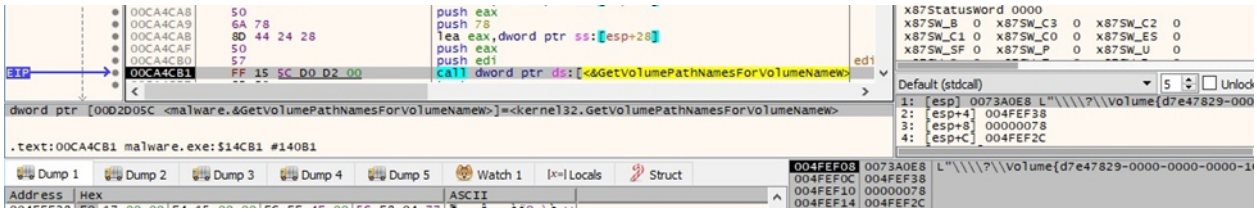


Figure 23

The volume enumeration continues by calling FindNextVolumeW:

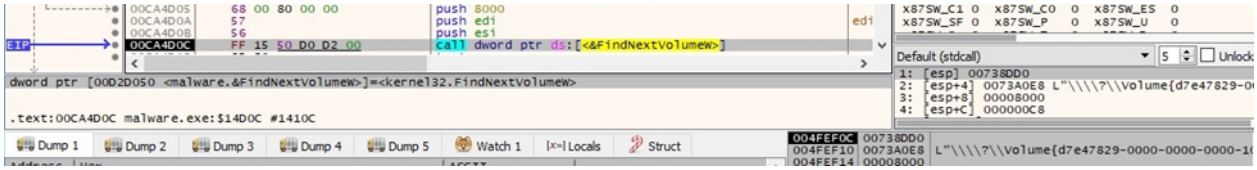


Figure 24

The malware is looking for volumes that aren't mounted using the GetDriveTypeW routine (0x1 = **DRIVE_NO_ROOT_DIR**):

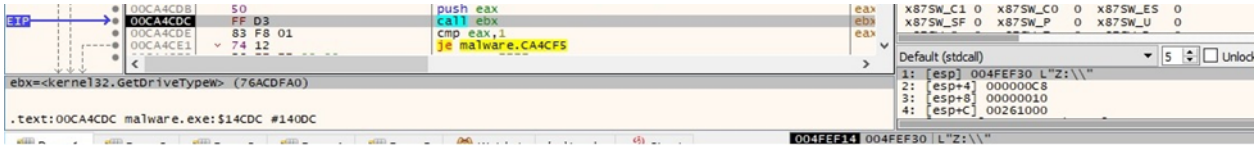


Figure 25

The binary associates an unmounted volume with a drive letter using SetVolumeMountPointW:

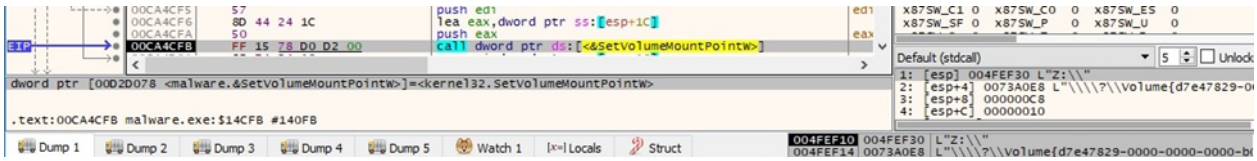


Figure 26

AvosLocker obtains a bitmask representing the available disk drives:

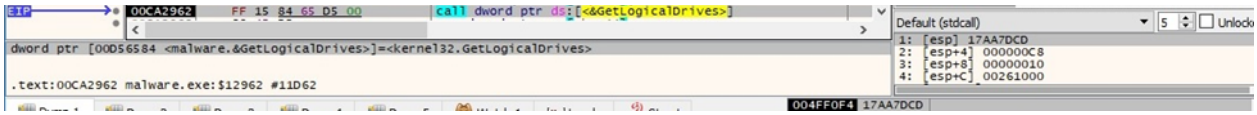


Figure 27

The process creates a thread for each drive that is found. The same method as above is utilized here, i.e., the starting address of the thread is StartAddress (sub_D0155F); however, the important function is sub_CBB930:

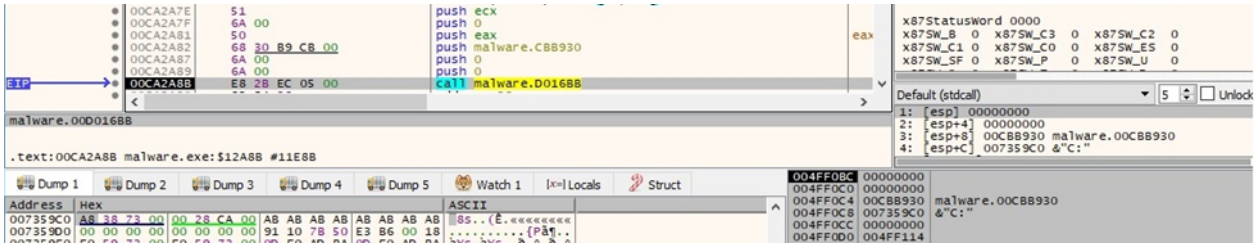


Figure 28

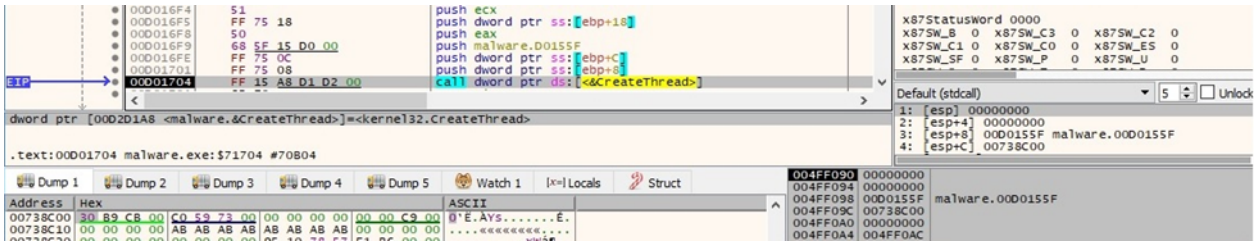


Figure 29

All identified drives are written to the command line, as highlighted in figure 30.

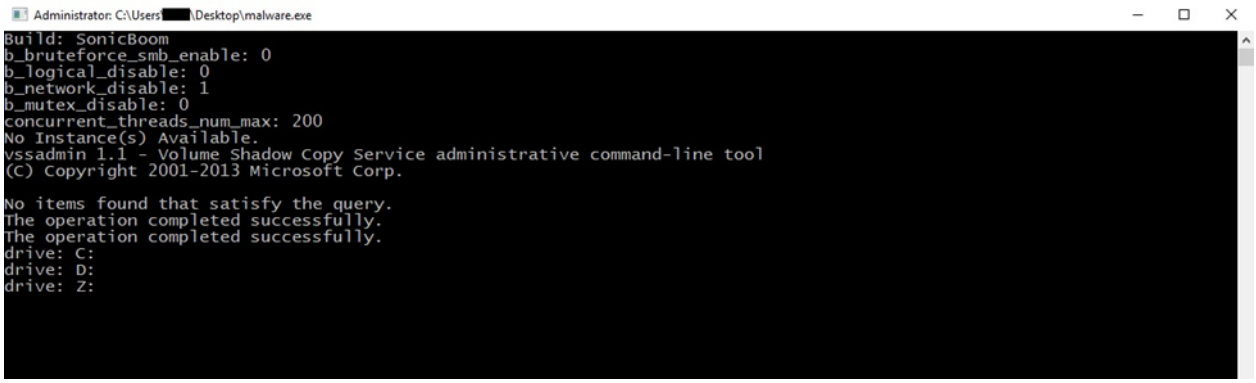


Figure 30

The new threads' priority is also set to **THREAD_PRIORITY_HIGHEST** by the malicious binary.

GetConsoleWindow is used to retrieve the window handle used by the console associated with the process:

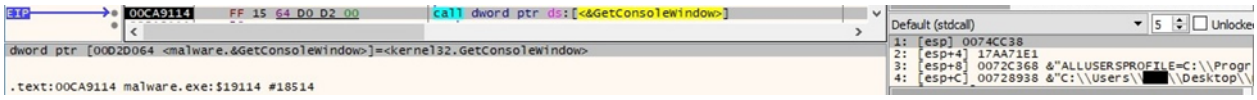


Figure 31

The malware calls the ShutdownBlockReasonCreate API and indicates that the machine should not be shut down during the encryption process:

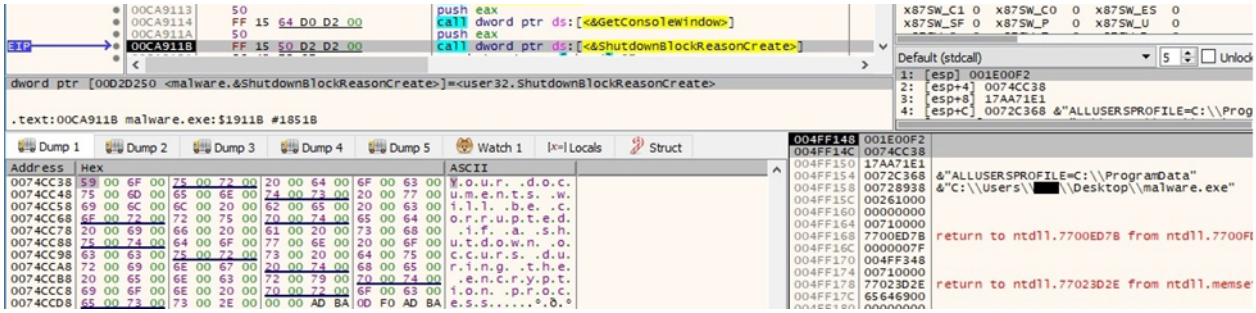


Figure 32

The ransomware extracts the identifier of the calling thread:

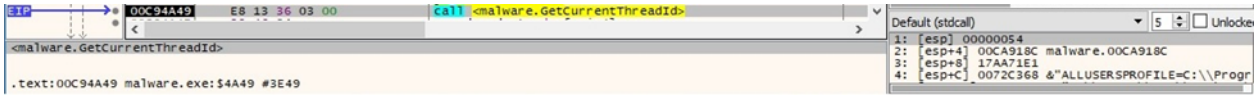


Figure 33

AvosLocker blocks the calling thread until all created threads will terminate their work using the Join method:

```
.text:00C94A53 push    0               ; int
.text:00C94A55 push    dword ptr [esi+4] ; int
.text:00C94A58 push    dword ptr [esi] ; hHandle
.text:00C94A5A call    __Thrd_join
.text:00C94A5F add     esp, 0Ch
.text:00C94A62 test    eax, eax
.text:00C94A64 jnz     short loc_C94A7B
```

Figure 34

Thread activity – sub_CBB930 function

The ransomware decrypts a list of extensions that will be skipped (figure 35):

- "avos" "avoslinux" "avos2" "avos2j" "themepack" "nls" "diagpkg" "msi" "lnk" "exe" "cab" "scr" "bat" "drv" "rtp" "msp"
- "prf" "msc" "ico" "key" "ocx" "diagcab" "diagcfg" "pdb" "wpx" "hlp" "icns" "rom" "dll" "msstyles" "mod" "ps1" "ics" "hta"
- "bin" "cmd" "ani" "386" "lock" "cur" "idx" "sys" "com" "deskthemepack" "shs" "ldf" "theme" "mpa" "nomedia" "spl" "cpl" "adv" "icl" "msu"

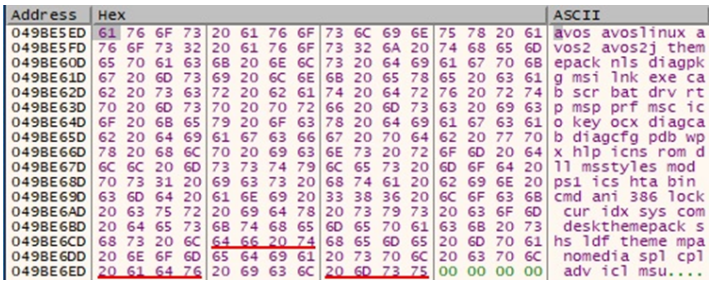


Figure 35

The malicious executable starts enumerating the drive by calling the FindFirstFileW function:

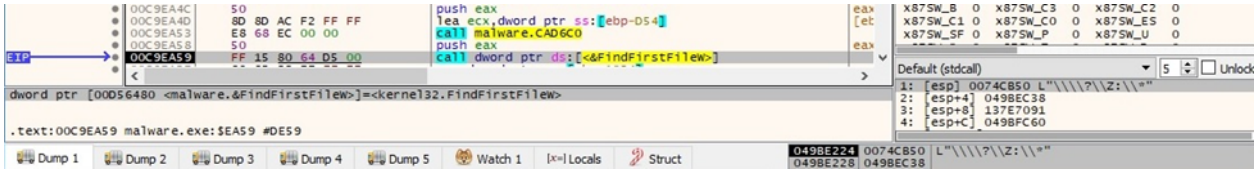


Figure 36

The ransomware creates a ransom note called “GET_YOUR_FILES_BACK.txt” in every directory that will be encrypted (0x40000000 = **GENERIC_WRITE**, 0x2 = **CREATE_ALWAYS**, 0x80 = **FILE_ATTRIBUTE_NORMAL**):

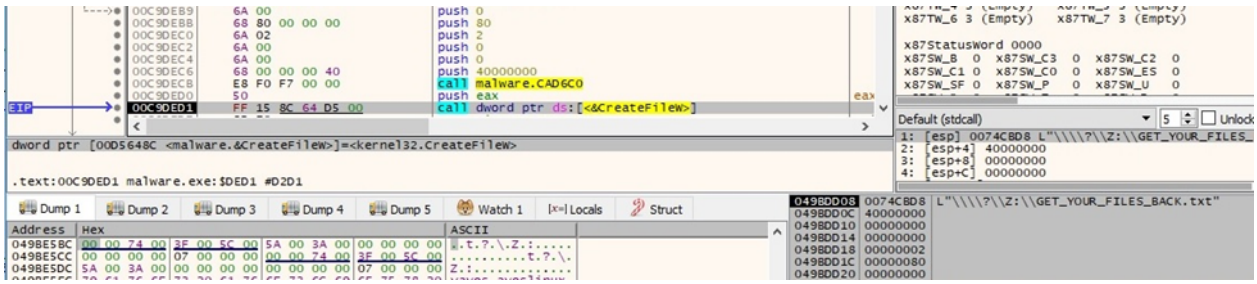


Figure 37

The WriteFile routine is used to populate the ransom note:

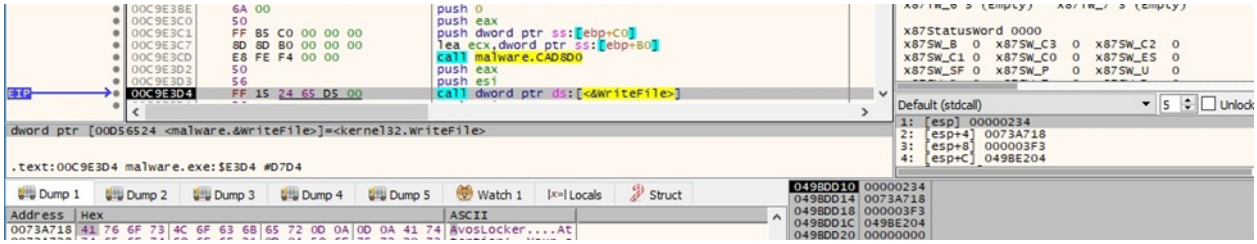


Figure 38

```
AvosLocker

Attention!
Your systems have been encrypted, and your confidential documents were downloaded.
In order to restore your data, you must pay for the decryption key & application.
You may do so by visiting us at http://avosjon4pfh3y7ew3jdwz6ofw7ll1jcxlbk7hcxxmnxlh5kvf2akoqjad.onion.
This is an onion address that you may access using Tor Browser which you may download at https://www.torproject.org/download/
Details such as pricing, how long before the price increases and such will be available to you once you enter your ID presented to you below in this note in our website.
Contact us soon, because those who don't have their data leaked in our press release blog and the price they'll have to pay will go up significantly.
The corporations whom don't pay or fail to respond in a swift manner have their data leaked in our blog, accessible at http://avosqxh72b5ia23dl5fgwcpndkctuzqvh2iefk5imp3pi5gfhel5klad.onion


Your ID: 6fe8a214a13dc49ce542b47e1d63135f48e40ccf6e0cd20fdbeafdc720a717cb
```

Figure 39

The process decrypts a list of folders that will not be encrypted (figure 40):

- "Program Files" "Windows" "Windows.old" "bootmgr" "ProgramData" "System Volume Information"
- "AppData" "Public" "All Users" "boot" "Intel" "WinNT" "Sophos" "Microsoft." "Games" "config.msi"

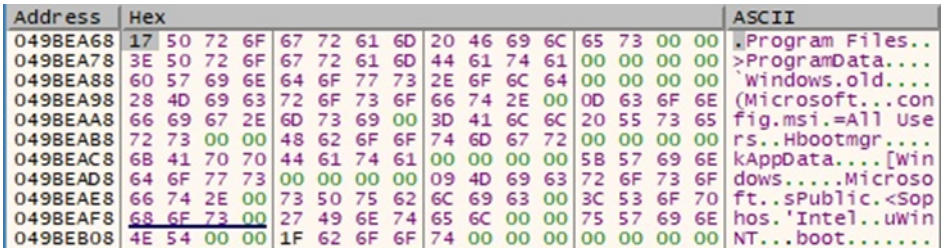


Figure 40

AvosLocker continues the file enumeration using FindNextFileW:

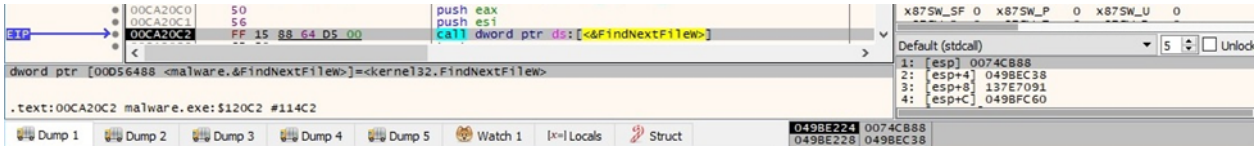


Figure 41

A list of files that will be skipped is decrypted using the XOR operator (figure 42):

- "GET_YOUR_FILES_BACK.txt" "desktop.ini" "autorun.inf" "ntldr" "bootsect.bak" "thumbs.db"
- "boot.ini" "ntuser.dat" "iconcache.db" "bootfont.bin" "ntuser.ini" "ntuser.dat.log" "Thumbs.db"

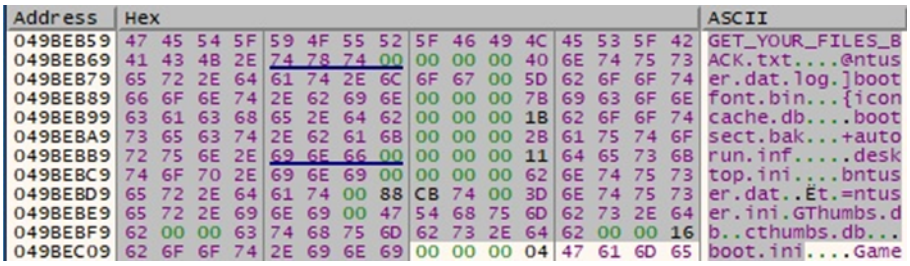


Figure 42

An example of a comparison between the file extension and one that is whitelisted is shown below:

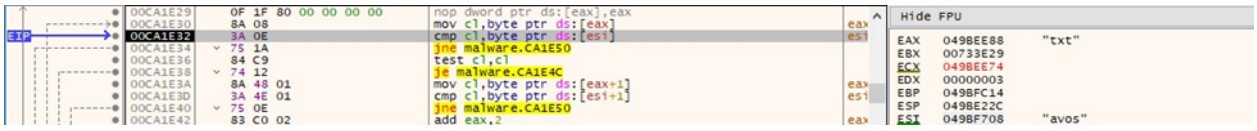


Figure 43

The ransomware sends an I/O completion packet that contains the targeted file path to the IOCP created earlier:

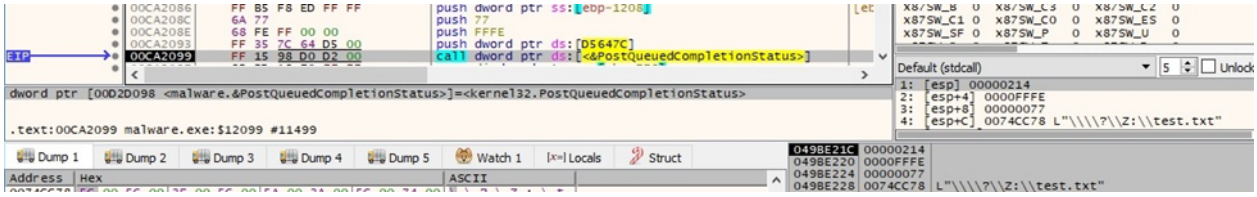


Figure 44

Thread activity – sub_CBBAD0 function

GetQueuedCompletionStatus is utilized to dequeue an I/O completion packet from the IOCP:

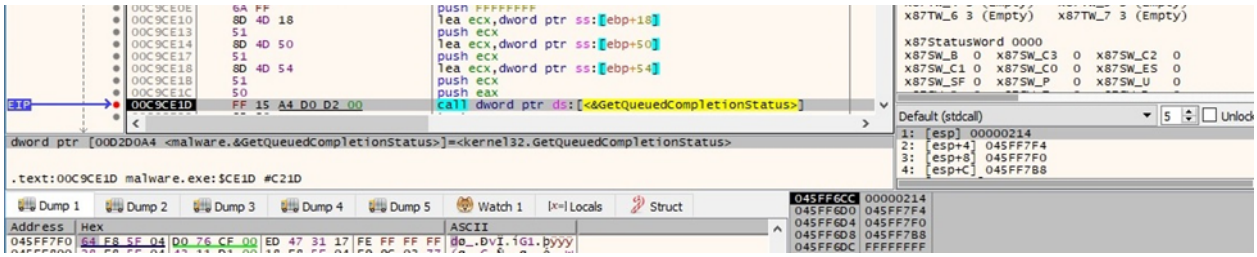


Figure 45

AvosLocker retrieves file system attributes for a file or directory and avoids the **FILE_ATTRIBUTE_SYSTEM** (0x4) attribute:

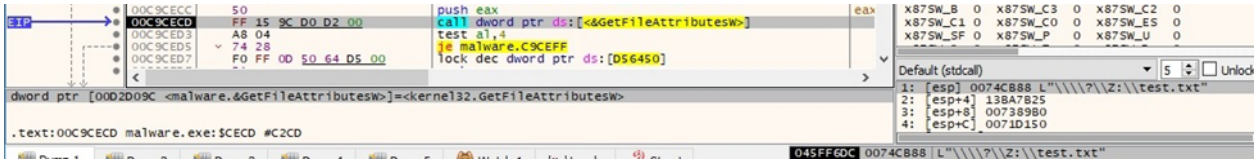


Figure 46

Based on the assembly code we analyzed, the ransomware uses a free C++ library of cryptographic schemes called Cryptopp (https://github.com/weidai11/cryptopp):

```
.text:00CDB234
.text:00CDB234 loc_CDB234:
.text:00CDB234 push    offset aCryptographicA ; "Cryptographic algorithms are disabled b"...
.text:00CDB239 lea     ecx, [ebp+var_28]
.text:00CDB23C call    sub_CADA40
.text:00CDB241 lea     eax, [ebp+var_28]
.text:00CDB244 mov     [ebp+var_4], 0
.text:00CDB24B push    eax             ; void *
.text:00CDB24C lea     ecx, [ebp+var_50]
.text:00CDB24F call    sub_CDB7D0
.text:00CDB254 push    offset __TI3?AVSelfTestFailure@CryptoPP@@ ; throw info for 'class CryptoPP::SelfTestFailure'
.text:00CDB254                         ; throw info for 'class CryptoPP::SelfTestFailure'
.text:00CDB259 lea     eax, [ebp+var_50]
.text:00CDB25C push    eax
.text:00CDB25D call    sub_CF782E
```

Figure 47

The malicious process acquires a handle to a key container within a particular cryptographic service provider (0x1 = **PROV_RSA_FULL**, 0xF0000000 = **CRYPT_VERIFYCONTEXT**):

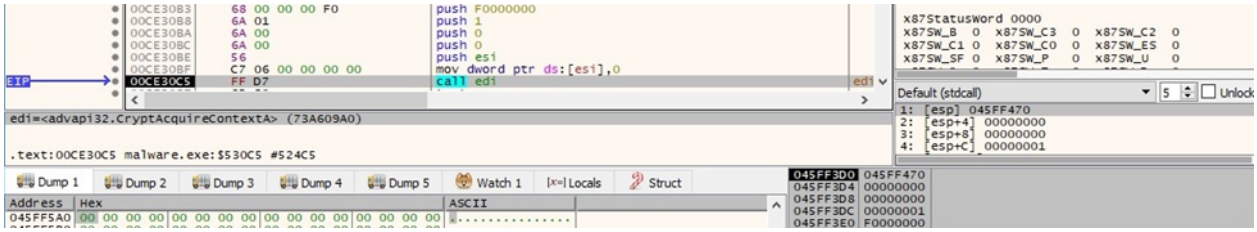


Figure 48

The ransomware generates 32 random bytes via a function call to CryptGenRandom. These bytes will be used to derive a Salsa20 key and a nonce:

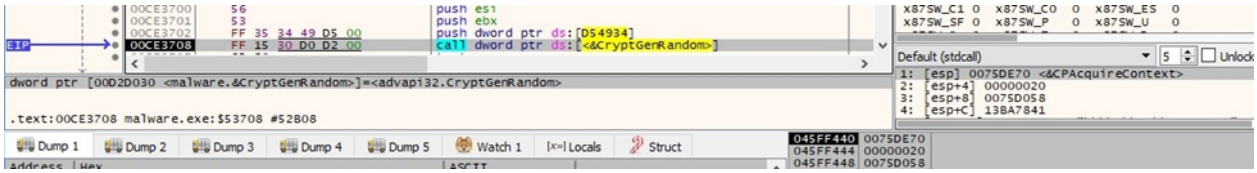


Figure 49

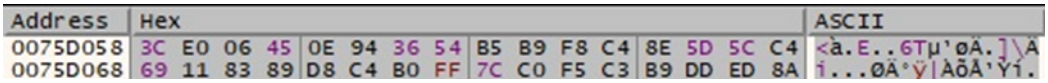


Figure 50

The RSA public key is converted into an array of bytes using CryptStringToBinaryA:

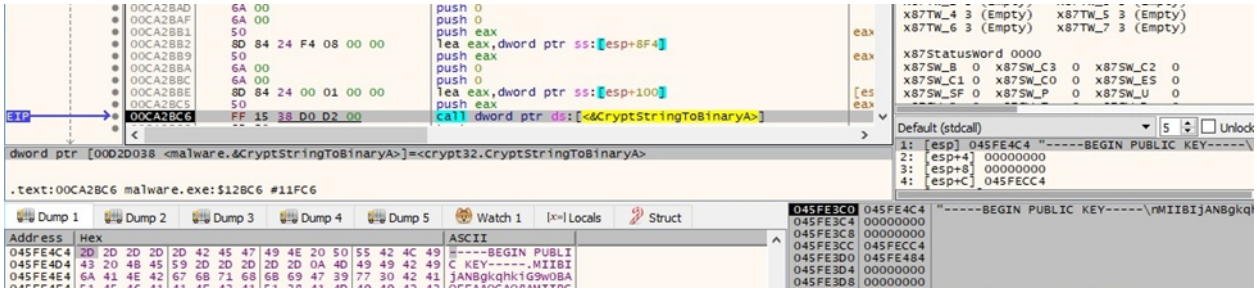


Figure 51

The CryptDecodeObjectEx API is utilized to decode a structure of a particular type (0x1 = **X509_ASN_ENCODING**, 0x8 = **X509_PUBLIC_KEY_INFO**, 0x8000 = **CRYPT_DECODE_ALLOC_FLAG**):

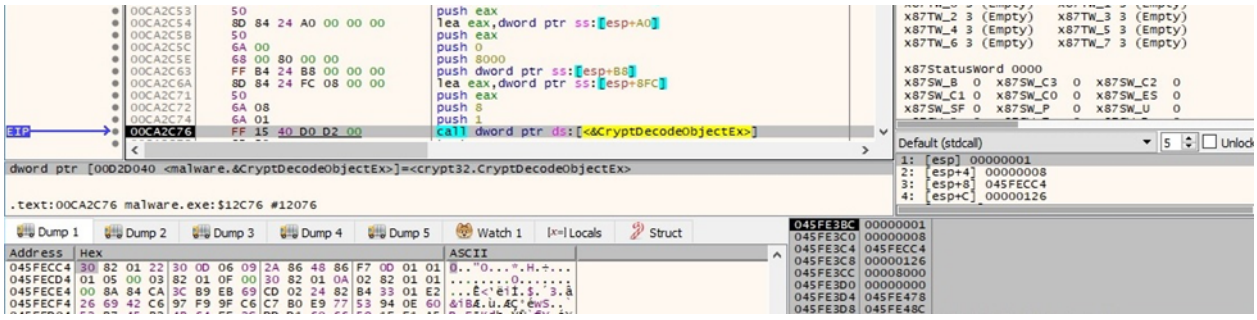


Figure 52

The process converts and imports the RSA public key information into the provider and returns a handle (0x1 = **X509_ASN_ENCODING**):

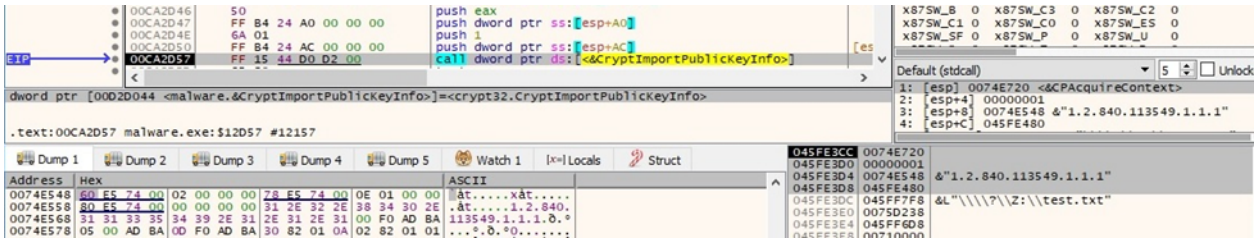


Figure 53

The Salsa20 key (32 bytes) and a nonce (8 bytes) that were derived from the randomly generated buffer are encrypted using the RSA public key:

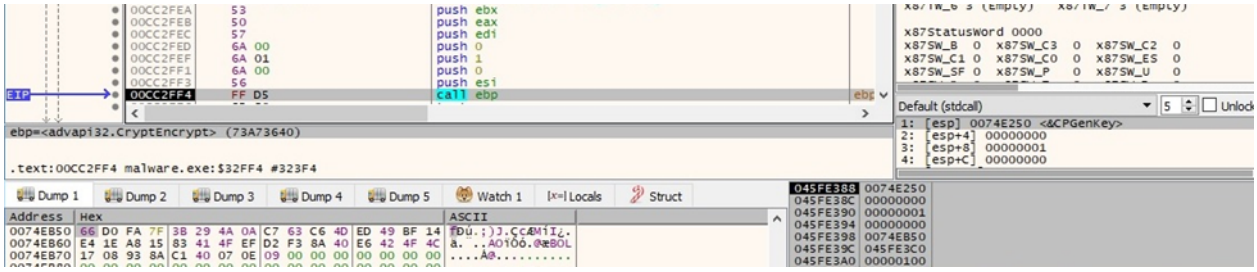


Figure 54

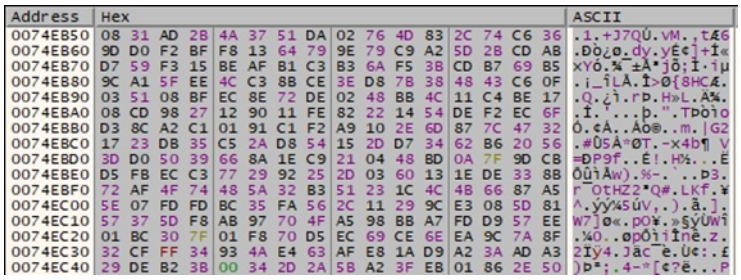


Figure 55

The above buffer is reversed and converted to Base64 format (0x40000001 = **CRYPT_STRING_NOCRLF** | **CRYPT_STRING_BASE64**):

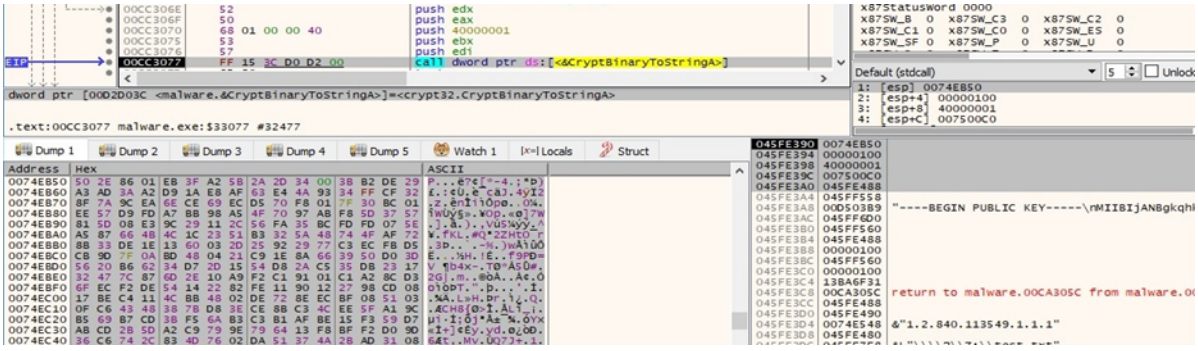
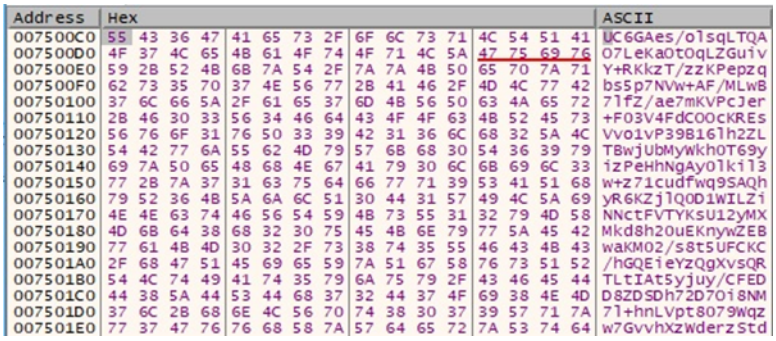


Figure 56



Address	Hex	ASCII
007500C0	55 43 36 47 41 65 73 2F 6F 6C 73 71 4C 54 51 41	UC6GAes/olsqLTQA
007500D0	4F 37 4C 65 4B 61 4F 74 4F 71 4C 5A 47 75 69 76	O7LeKaOtOqLZGuiv
007500E0	59 2B 52 4B 6B 7A 54 2F 7A 7A 4B 50 65 70 7A 71	Y+RKkzT/zzKPepzq
007500F0	62 73 35 70 37 4E 56 77 2B 41 46 2F 4D 4C 77 42	bs5p7NVw+AF/MLwB
00750100	37 6C 66 5A 2F 61 65 37 6D 4B 56 50 63 4A 65 72	7lfZ/ae7mKVPcJer
00750110	2B 46 30 33 56 34 46 64 43 4F 4F 63 4B 52 45 73	+F03V4FdCOOcKREs
00750120	56 76 6F 31 76 50 33 39 42 31 36 6C 68 32 5A 4C	Vvo1vP39B16lh2ZL
00750130	54 42 77 6A 55 62 4D 79 57 6B 68 30 54 36 39 79	TBwjUbMyWkh0T69y
00750140	69 7A 50 65 48 68 4E 67 41 79 30 6C 6B 69 6C 33	izPeHhNgAy0lkil3
00750150	77 2B 7A 37 31 63 75 64 66 77 71 39 53 41 51 68	w+z71cudfwq9SAQh
00750160	79 52 36 4B 5A 6A 6C 51 30 44 31 57 49 4C 5A 69	yR6KZjlQ0D1WILZi
00750170	4E 4E 63 74 46 56 54 59 4B 73 55 31 32 79 4D 58	NNctFVTYKsU12yMX
00750180	4D 6B 64 38 68 32 30 75 45 4B 6E 79 77 5A 45 42	Mkd8h20uEKnywZEB
00750190	77 61 4B 4D 30 32 2F 73 38 74 35 55 46 43 4B 43	waKM02/s8t5UFCKC
007501A0	2F 68 47 51 45 69 65 59 7A 51 67 58 76 73 51 52	/hGQEieYzQgXvsQR
007501B0	54 4C 74 49 41 74 35 79 6A 75 79 2F 43 46 45 44	TLtIAt5yjuy/CFED
007501C0	44 38 5A 44 53 44 68 37 32 44 37 4F 69 38 4E 4D	D8ZDSDh72D7Oi8NM
007501D0	37 6C 2B 68 6E 4C 56 70 74 38 30 37 39 57 71 7A	7l+hnLVpt8079Wqz
007501E0	77 37 47 76 76 68 58 7A 57 64 65 72 7A 53 74 64	w7GvvhXzWderzStd

Figure 57

The AllocateAndInitializeSid function is utilized to allocate and initialize a security identifier (SID) with 2 subauthorities:

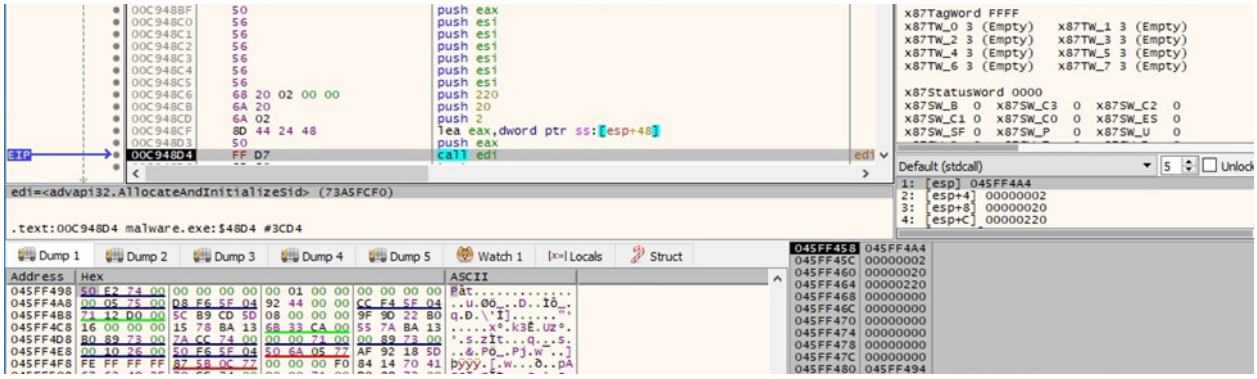


Figure 58

The malware creates a new ACL by calling the SetEntriesInAclA routine:

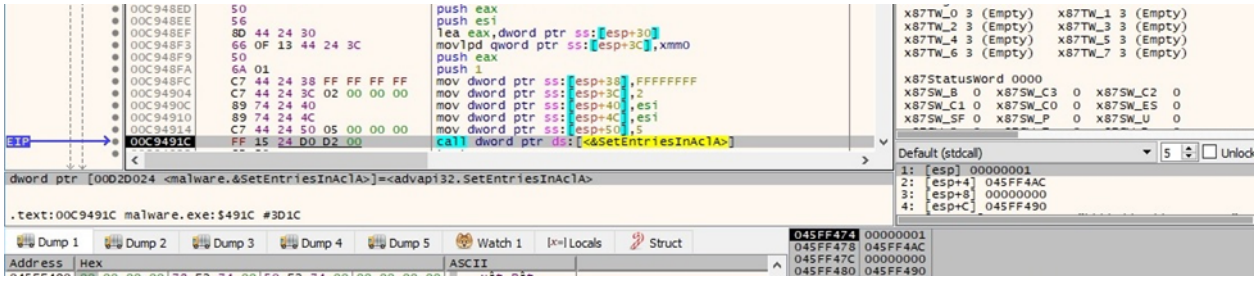


Figure 59

SetNamedSecurityInfoW is utilized to modify the DACL of the targeted file (0x1 = **SE_FILE_OBJECT**, 0x4 = **DACL_SECURITY_INFORMATION**):

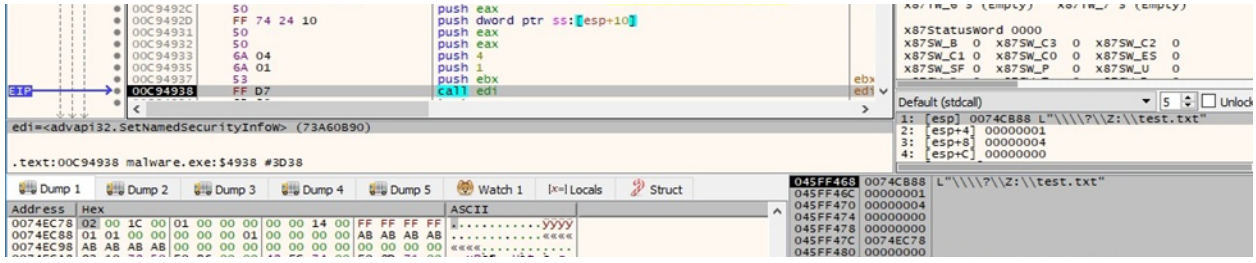


Figure 60

The ransomware opens the file via a call to CreateFileW (0xc0000000 = **GENERIC_READ** | **GENERIC_WRITE**, 0x3 = **OPEN_EXISTING**, 0x80 = **FILE_ATTRIBUTE_NORMAL**):

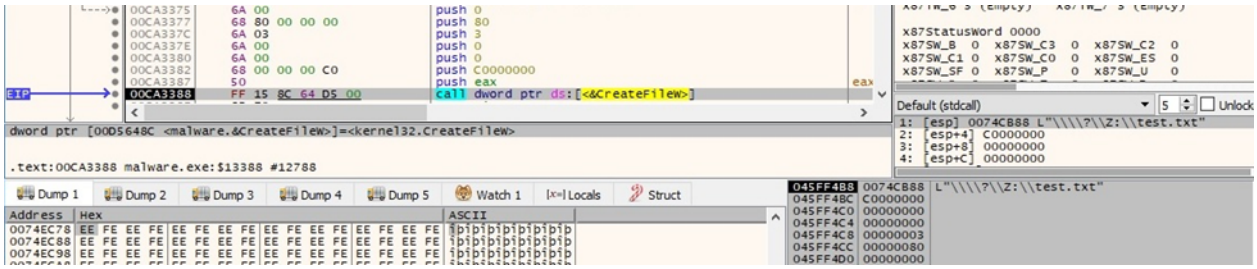


Figure 61

The file size is retrieved using the GetFileSizeEx API:

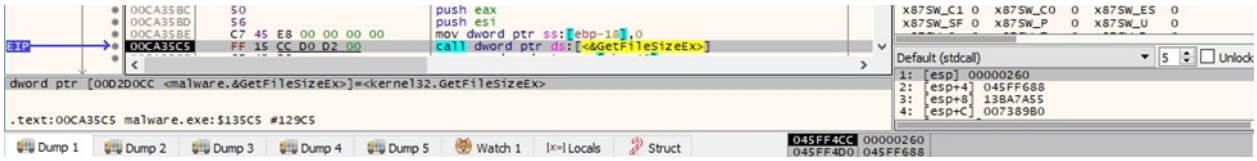


Figure 62

AvosLocker reads 1MB at a time:

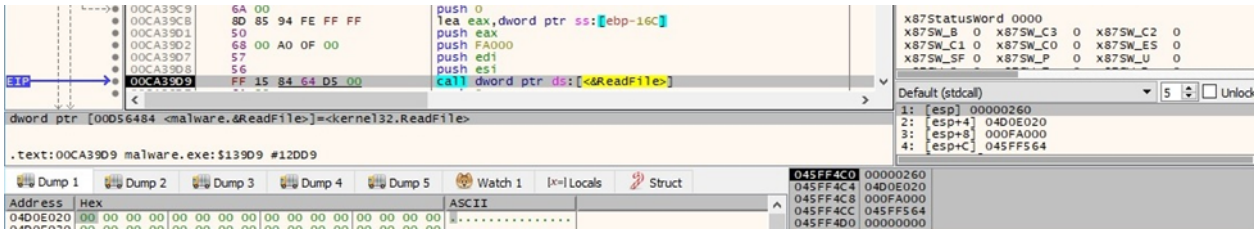


Figure 63

The process moves the file pointer to the beginning of the file by calling SetFilePointer (0x0 = **FILE_BEGIN**):

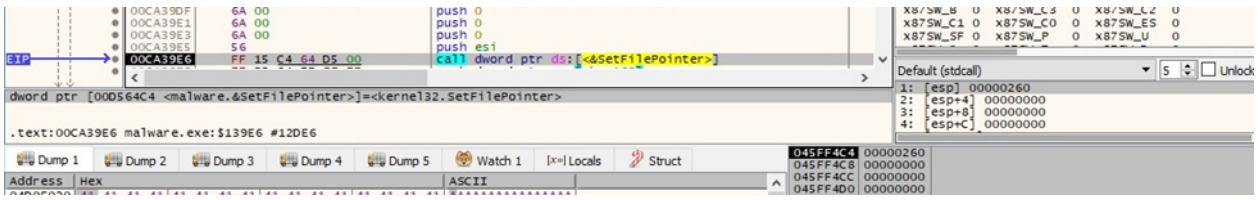


Figure 64

The binary passes a pointer to the file content to the encryption function:

Figure 65

The file content is encrypted using the Salsa20 algorithm. The implementation below is very similar to the one presented at https://github.com/weidai11/cryptopp/blob/master/salsa.cpp:

```
.text:00CDF672 loc_CDF672:
.text:00CDF672 movdqa  xmm0, [esp+3F0h+var_30C]
.text:00CDF67B movdqa  xmm4, [esp+3F0h+var_2FC]
.text:00CDF684 movdqa  xmm2, [esp+3F0h+var_3CC]
.text:00CDF68A movdqa  xmm6, [esp+3F0h+var_3BC]
.text:00CDF690 paddd   xmm0, xmm2
.text:00CDF694 paddd   xmm4, xmm6
.text:00CDF698 movdqa  xmm1, xmm0
.text:00CDF69C movdqa  xmm5, xmm4
.text:00CDF6A0 pslld   xmm0, 7
.text:00CDF6A5 pslld   xmm4, 7
.text:00CDF6AA psrld   xmm1, 19h
.text:00CDF6AF psrld   xmm5, 19h
.text:00CDF6B4 pxor    xmm0, [esp+3F0h+var_38C]
.text:00CDF6BA pxor    xmm4, [esp+3F0h+var_37C]
.text:00CDF6C0 pxor    xmm0, xmm1
.text:00CDF6C4 pxor    xmm4, xmm5
.text:00CDF6C8 movdqa  [esp+3F0h+var_38C], xmm0
.text:00CDF6CE movdqa  [esp+3F0h+var_37C], xmm4
.text:00CDF6D4 movdqa  xmm1, xmm0
.text:00CDF6D8 movdqa  xmm5, xmm4
.text:00CDF6DC paddd   xmm0, xmm2
.text:00CDF6E0 paddd   xmm4, xmm6
.text:00CDF6E4 movdqa  xmm3, xmm0
.text:00CDF6E8 movdqa  xmm7, xmm4
.text:00CDF6EC pslld   xmm0, 9
.text:00CDF6F1 pslld   xmm4, 9
.text:00CDF6F6 psrld   xmm3, 17h
.text:00CDF6FB psrld   xmm7, 17h
.text:00CDF700 pxor    xmm0, [esp+3F0h+var_34C]
.text:00CDF709 pxor    xmm4, [esp+3F0h+var_33C]
.text:00CDF712 pxor    xmm0, xmm3
.text:00CDF716 pxor    xmm4, xmm7
.text:00CDF71A movdqa  [esp+3F0h+var_34C], xmm0
.text:00CDF723 movdqa  [esp+3F0h+var_33C], xmm4
.text:00CDF72C movdqa  xmm3, xmm0
.text:00CDF730 movdqa  xmm7, xmm4
.text:00CDF734 paddd   xmm0, xmm1
.text:00CDF738 paddd   xmm4, xmm5
.text:00CDF73C movdqa  xmm1, xmm0
.text:00CDF740 movdqa  xmm5, xmm4
.text:00CDF744 pslld   xmm0, 0Dh
.text:00CDF749 pslld   xmm4, 0Dh
.text:00CDF74E psrld   xmm1, 13h
```

Figure 66

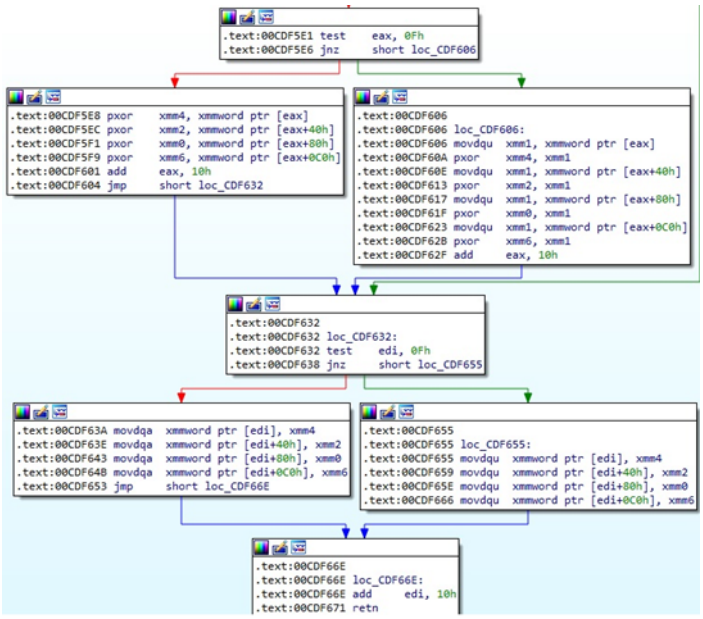


Figure 67

The encrypted file content and the encrypted Salsa20 key and nonce are written to the file using WriteFile:

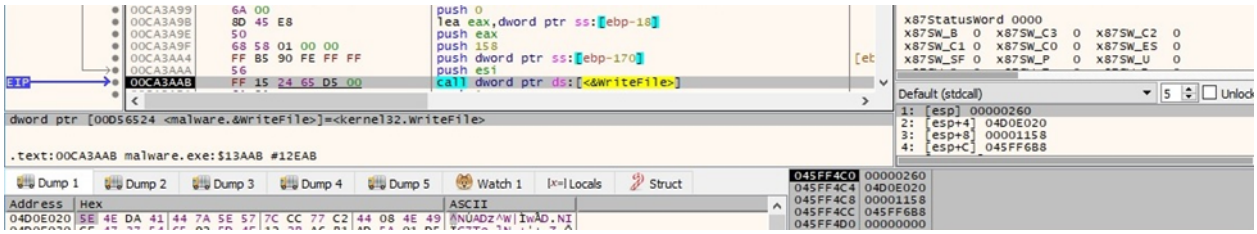


Figure 68

The ".avos2" extension is appended to the encrypted files:

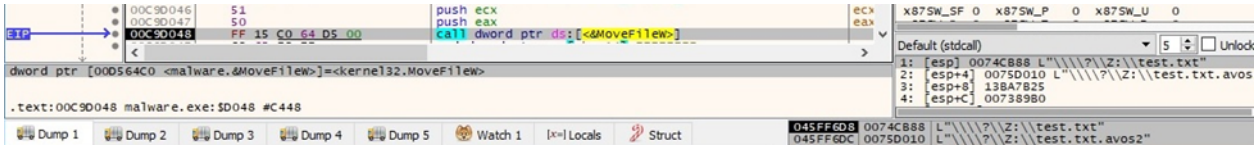


Figure 69

An example of an encrypted file is displayed in figure 70.

test.txt.avos2

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000F70   EF 71 48 31 9F 43 3A 1A B7 37 49 DC 55 BB 53 35  ïqH1ŸC:.·7IÜU»S5
00000F80   B1 B9 3D 35 B8 73 46 A1 96 F9 0B F1 B9 E6 20 2A  ±¹=5¸sF¡-ù.ñ¹æ *
00000F90   91 C2 93 94 A1 E4 4F 0C 53 FC 1E 2B 12 2F EF 2C  'Â"”¡äO.Sü.+./ï,
00000FA0   7A F4 FB 8D B7 A0 2A BA 69 E0 A2 3F EB 95 15 28  zôû.· *ºià¢?ë•.(
00000FB0   4C 36 4D EB D2 64 38 E9 12 ED FF F0 A6 31 20 05  L6MëÒd8é.íÿð¦1 .
00000FC0   99 F1 2C 5B 1C 4E 29 99 82 15 42 B3 E6 F8 11 65  ™ñ,[.N)™‚.B³æø.e
00000FD0   35 27 64 C2 2F CC 0F 27 36 19 35 C4 DC 28 9A 2F  5'dÂ/Ì.'6.5ÄÜ(š/
00000FE0   FF 3D 3D 62 E0 D5 E2 38 E8 36 24 26 A5 85 FD B0  ÿ==bàÕâ8è6$&¥…ý°
00000FF0   0B 15 24 BD C8 DF DE A0 8E AD 7C D4 E9 1E B5 18  ..$½ÈßÞ Ž.|Ôé.µ.
00001000   55 43 36 47 41 65 73 2F 6F 6C 73 71 4C 54 51 41  UC6GAes/olsqLTQA
00001010   4F 37 4C 65 4B 61 4F 74 4F 71 4C 5A 47 75 69 76  O7LeKaOtOqLZGuiv
00001020   59 2B 52 4B 6B 7A 54 2F 7A 7A 4B 50 65 70 7A 71  Y+RKkzT/zzKPepzq
00001030   62 73 35 70 37 4E 56 77 2B 41 46 2F 4D 4C 77 42  bs5p7NVw+AF/MLwB
00001040   37 6C 66 5A 2F 61 65 37 6D 4B 56 50 63 4A 65 72  71fZ/ae7mKVPcJer
00001050   2B 46 30 33 56 34 46 64 43 4F 4F 63 4B 52 45 73  +F03V4FdCOOcKREs
00001060   56 76 6F 31 76 50 33 39 42 31 36 6C 68 32 5A 4C  VvolvP39B16lh2ZL
00001070   54 42 77 6A 55 62 4D 79 57 6B 68 30 54 36 39 79  TBwjUbMyWkh0T69y
00001080   69 7A 50 65 48 68 4E 67 41 79 30 6C 6B 69 6C 33  izPeHhNgAy0lkil3
00001090   77 2B 7A 37 31 63 75 64 66 77 71 39 53 41 51 68  w+z71cudfwq9SAQh
000010A0   79 52 36 4B 5A 6A 6C 51 30 44 31 57 49 4C 5A 69  yR6KZjlQ0D1WILZi
000010B0   4E 4E 63 74 46 56 54 59 4B 73 55 31 32 79 4D 58  NNctFVTYKsU12yMX
000010C0   4D 6B 64 38 68 32 30 75 45 4B 6E 79 77 5A 45 42  Mkd8h20uEKnywZEB
000010D0   77 61 4B 4D 30 32 2F 73 38 74 35 55 46 43 4B 43  waKM02/s8t5UFCKC
000010E0   2F 68 47 51 45 69 65 59 7A 51 67 58 76 73 51 52  /hGQEieYzQgXvsQR
000010F0   54 4C 74 49 41 74 35 79 6A 75 79 2F 43 46 45 44  TLtIAt5yjuy/CFED
00001100   44 38 5A 44 53 44 68 37 32 44 37 4F 69 38 4E 4D  D8ZDSDh72D7Oi8NM
00001110   37 6C 2B 68 6E 4C 56 70 74 38 30 37 39 57 71 7A  7l+hnLVpt8079Wqz
00001120   77 37 47 76 76 68 58 7A 57 64 65 72 7A 53 74 64  w7GvvhXzWderzStd
00001130   6F 73 6C 35 6E 6E 6C 6B 45 2F 69 2F 38 74 43 64  osl5nnlkE/i/8tCd
00001140   4E 73 5A 30 4C 49 4E 4E 64 67 4C 61 55 54 64 4B  NsZ0LINNdgLaUTdK
00001150   4B 36 30 78 43 41 3D 3D                          K60xCA==
```

Figure 70

We continue with the analysis of the main thread.

AvosLocker displays some statistics regarding encryption in the command line window:

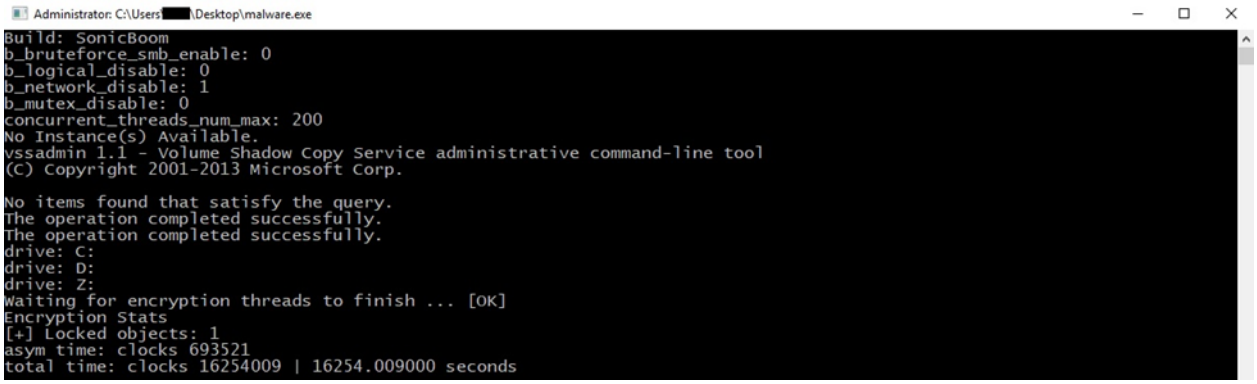


Figure 71

The ransomware decrypts and runs a PowerShell script (see figure 72):

- powershell -Command "$a = [System.IO.File]::ReadAllText(\"Z:\GET_YOUR_FILES_BACK.txt\");Add-Type -AssemblyName System.Drawing;$filename = \"$env:temp\$(Get-Random).png\";$bmp = new-object System.Drawing.Bitmap 1920,1080;$font = new-object System.Drawing.Font Consolas,10;$brushBg = [System.Drawing.Brushes]::Black;$brushFg = [System.Drawing.Brushes]::White;$format = [System.Drawing.StringFormat]::GenericDefault;$format.Alignment = [System.Drawing.StringAlignment]::Center;$format.LineAlignment = [System.Drawing.StringAlignment]::Center;$graphics = [System.Drawing.Graphics]::FromImage($bmp);$graphics.FillRectangle($brushBg,0,0,$bmp.Width,$bmp.Height);$graphics.DrawString($a,$font,$brushFg,[System.Drawing.RectangleF]::FromLTRB(0, 0, 1920, 1080),$format);$graphics.Dispose();$bmp.Save($filename);reg add \"HKEY_CURRENT_USER\Control Panel\Desktop\" /v Wallpaper /t REG_SZ /d $filename /f;Start-Sleep 1;rundll32.exe user32.dll, UpdatePerUserSystemParameters, 0, $false;"

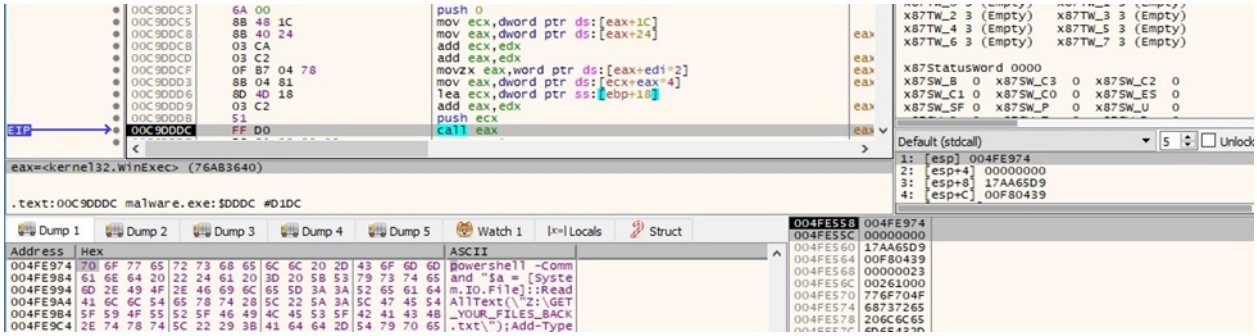


Figure 72

The script's purpose is to create an image that contains the ransom note and set that as the Desktop Wallpaper.

The final image that will be set is highlighted below:

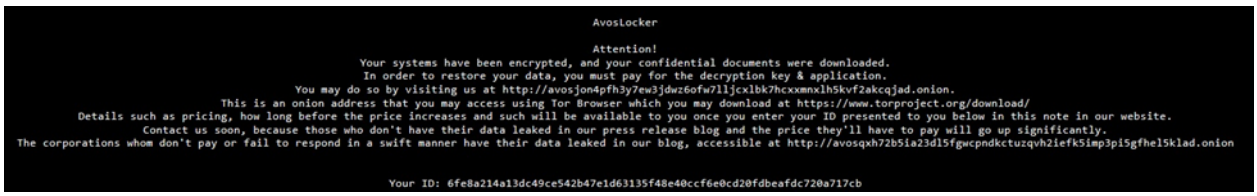


Figure 73

Running with the -h (--help) parameter

AvosLocker displays the help menu:

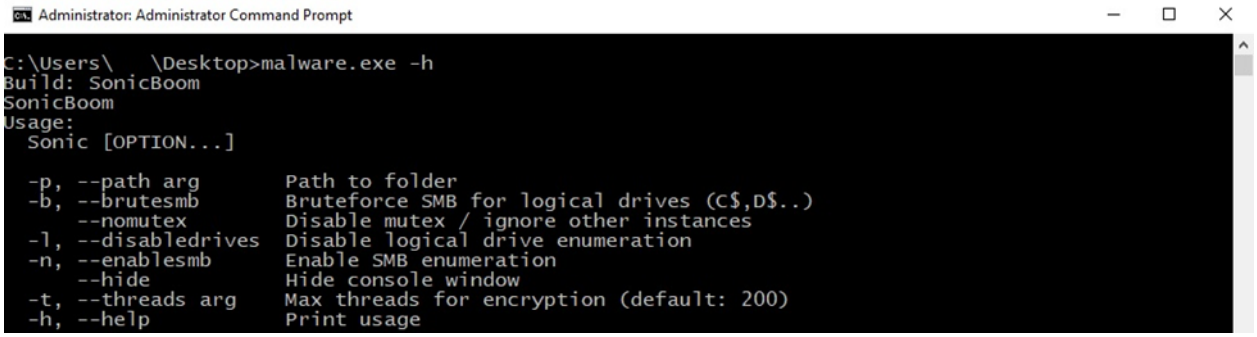


Figure 74

Running with the -p (--path) parameter

The ransomware only encrypts this specific directory:

Figure 75

Running with the -l (--disabledrives) parameter

AvosLocker doesn't encrypt the logical drives:

Figure 76

Running with the --hide parameter

The malicious executable retrieves the window handle used by the console:

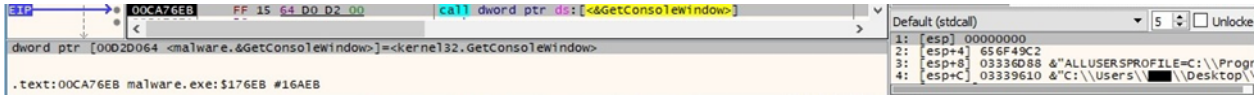


Figure 77

The console window is hidden by calling the ShowWindow function (0x0 = **SW_HIDE**):

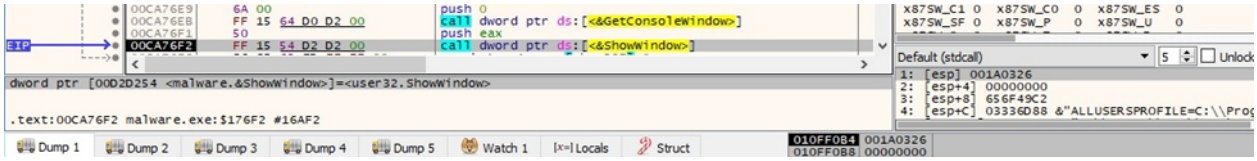


Figure 78

Running with the -t (--threads) parameter

This parameter represents the number of threads that will concurrently encrypt the files:

Figure 79

Running with the -n (--enablesmb) parameter

The ransomware starts enumerating all resources on the network via a function call to WNetOpenEnumA (0x2 = **RESOURCE_GLOBALNET**, 0x0 = **RESOURCETYPE_ANY**):

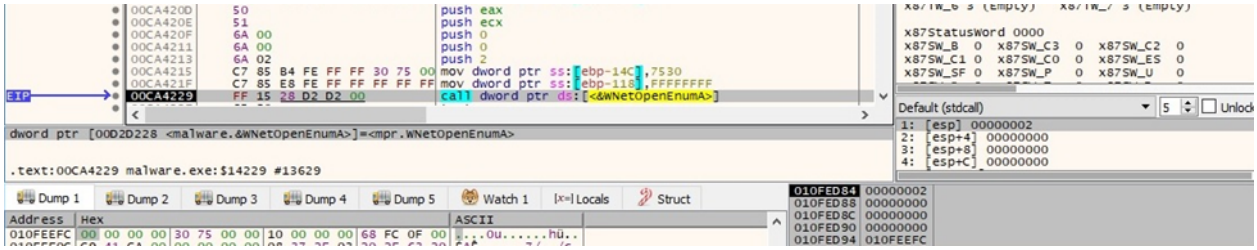


Figure 80

WNetEnumResourceA is utilized to continue the enumeration of network resources:

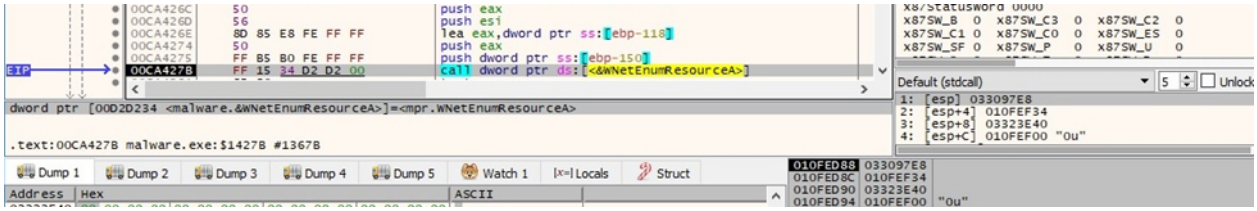


Figure 81

AvosLocker connects to a network share by calling the WNetAddConnection2A routine (0x4 = **CONNECT_TEMPORARY**):

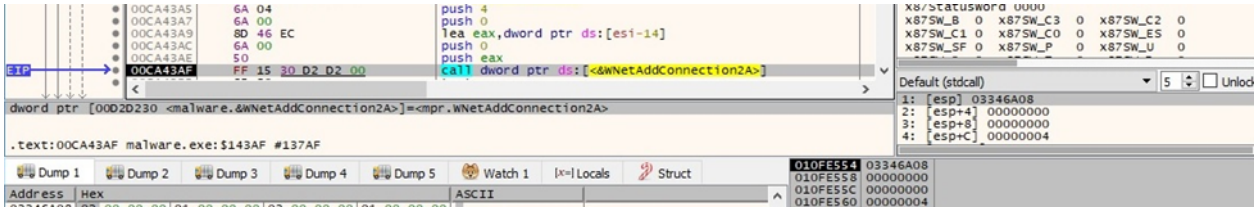


Figure 82

The network share name that will be encrypted is written to the command line:

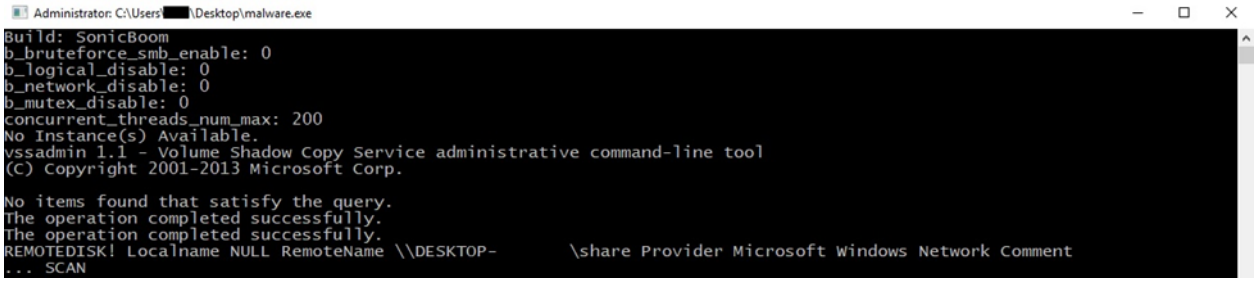


Figure 83

A similar thread that has enumerated logical drives is also created in order to traverse the network shares:

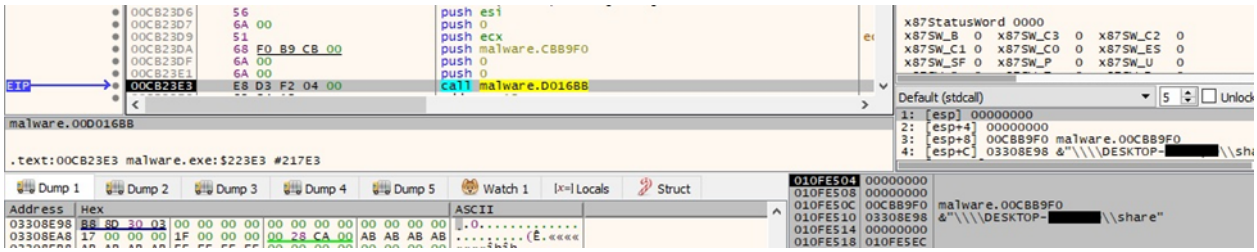


Figure 84

Running with the -b (--brutesmb) -n (--enablesmb) parameters

The process of discovering all network shares is identical to the above. The main idea, in this case, is that the malware is looking to extract the hostname/IP address from an available network share and trying to find logical drives based on it.

The binary makes a connection to a potential logical drive using the WNetAddConnection2A function (0x4 = **CONNECT_TEMPORARY**):

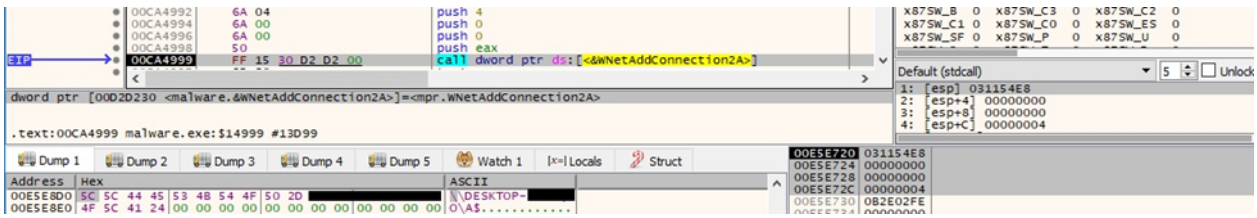


Figure 85

For all logical drives that can be found using the above method, the process creates a new thread that will enumerate them:

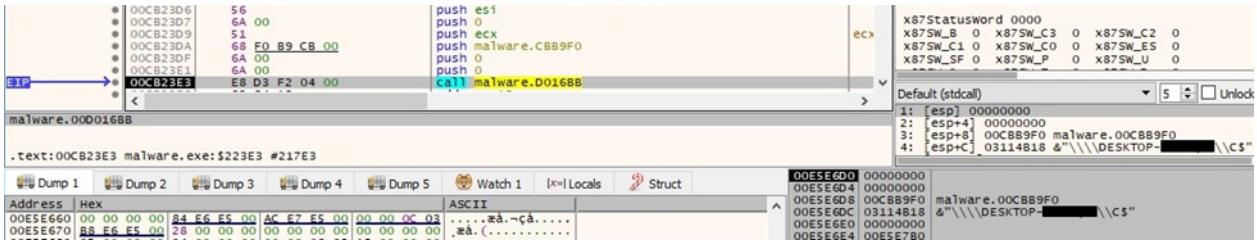


Figure 86

Finally, AvosLocker writes the logical drives that were found to the command line:

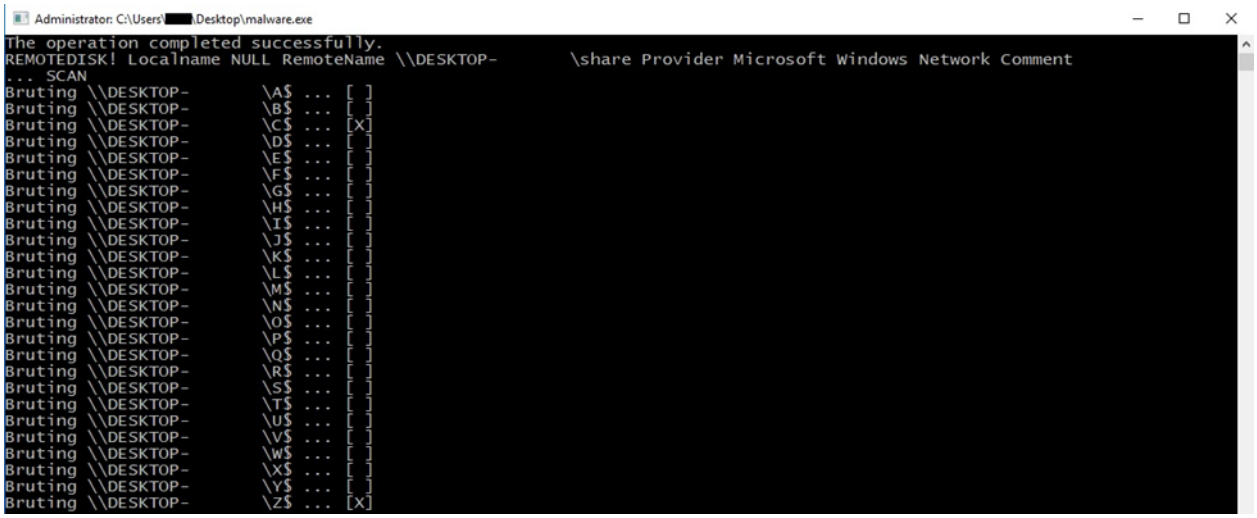


Figure 87

Running with the --nomutex parameter

The ransomware doesn't create the mutex in this case.

Indicators of Compromise

Mutex

Zheic0WaWie6zeiy

AvosLocker Ransom Note

GET_YOUR_FILES_BACK.txt

Processes spawned

cmd /c wmic shadowcopy delete /nointeractive

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

cmd /c bcdedit /set {default} recoveryenabled No

cmd /c bcdedit /set {default} bootstatuspolicy ignoreallfailures

cmd /c powershell -command \"Get-EventLog -LogName * | ForEach { Clear-EventLog $_.Log }\"