A Detailed Analysis of The SunCrypt Ransomware

LIFARS
a SecurityScorecard company

SecurityScorecard.com

info@securityscorecard.com

©2022 SecurityScorecard Inc.

214 West 29th St, 5th Floor

New York, NY 10001

1.800.682.1707

Table of Contents

Executive Summary

SunCrypt ransomware is a less sophisticated malware that has impacted multiple companies since 2019. The malware can run with one of the following parameters: "-noshares", "-nomutex", "-noreport", "-noservices", "-vm", "-path", "-justcrypt", and "-keep_exe". The ransomware kills a list of targeted processes and deletes all Volume Shadow Copies using COM objects.

The encryption is done using multithreading with I/O completion ports, which is a common technique used by most current ransomware families. SunCrypt uses a combination of Curve25519 and ChaCha20 algorithms during the encryption routine. The binary deletes the Windows event logs via two different methods and performs self-deletion at the end of the execution.

Analysis and Findings

SHA256: 759f2b24be12e208903b00f9719db71a332ddf8252986c26afbcda9f32623bc4

The malware forces the system not to display the critical-error-handler message box using SetErrorMode (0x1 = **SEM_FAILCRITICALERRORS**):

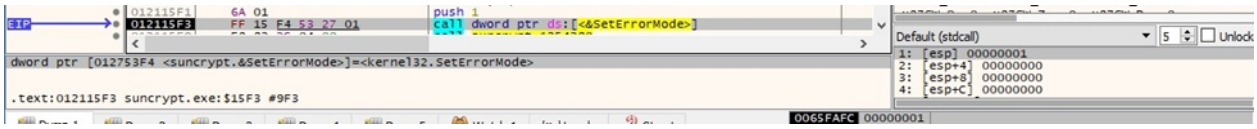


Figure 1

The binary loads multiple DLLs into the address space of the process by calling the LoadLibraryA API. The list of DLLs contains "ntdll.dll", "advapi32.dll", "kernel32.dll", and "rstrtmgr.dll". An example of such a function call is displayed below:

Figure 2

GetProcAddress is utilized to retrieve the address of multiple exported functions: "strncpy", "_atoi64", "atoi", "isxdigit", "isdigit", "memset", "memcpy", "NtSetInformationFile", "SystemFunction036", "SetVolumeMountPointW", "RmStartSession", "RmRegisterResources", "RmGetList", and "RmEndSession" (see figure 3).

Figure 3

SunCrypt can run with the following parameters "-noshares", "-nomutex", "-noreport", "-noservices", "-vm", "-path", "-justcrypt", "-keep_exe". We'll explain the purpose of each parameter in the upcoming paragraphs:

```
.text:012227C9 push    offset aNoshares ; "-noshares"
.text:012227CE call    sub_122CE90
.text:012227D3 add     esp, 4
.text:012227D6 xor     eax, 6Fh
.text:012227D9 imul    eax, 1000193h
.text:012227DF xor     eax, 6Eh
.text:012227E2 imul    eax, 1000193h
.text:012227E8 xor     eax, 2Dh
.text:012227EB imul    eax, 1000193h
.text:012227F1 mov     [esp+60h+var_24], eax
.text:012227F5 push    offset aNomutex ; "-nomutex"
.text:012227FA call    sub_122D1C0
```

Figure 4

The ransomware uses the FNV hash function in order to compute 4-byte hash values that are compared with the hard-coded ones corresponding to different parameters. The implementation of the hash function can be spotted through the identification of the FNV prime (0x01000193):

```
.text:0122CF8C mov     esi, ds:dword_1274BC8
.text:0122CF92 mov     ecx, ds:dword_1274A0C
.text:0122CF98 mov     ebx, [esp+24h+arg_0]
.text:0122CF9C mov     ebp, 208F3B67h
.text:0122CFA1 lea     edx, [eax-1]
.text:0122CFA4 imul    edx, eax
.text:0122CFA7 lea     eax, [esi-1]
.text:0122CFAA and     edx, 1
.text:0122CFAD setz    [esp+24h+var_22]
.text:0122CFB2 imul    eax, esi
.text:0122CFB5 test    edx, edx
.text:0122CFB7 mov     esi, 45F517F9h
.text:0122CFBC mov     edx, ds:dword_127482C
.text:0122CFC2 cmovz   ebp, esi
.text:0122CFC5 cmp     ecx, 0Ah
.text:0122CFC8 setl    [esp+24h+var_21]
.text:0122CFCD cmovl   ebp, esi
.text:0122CFD0 movzx   ecx, word ptr [ebx+12h]
.text:0122CFD4 movzx   edi, word ptr [ebx+10h]
.text:0122CFD8 mov     [esp+24h+var_1C], ebp
.text:0122CFDC movzx   esi, word ptr [ebx+0Eh]
.text:0122CFE0 xor     ecx, 811C9DC5h
.text:0122CFE6 imul    ecx, 1000193h
.text:0122CFEC xor     ecx, edi
.text:0122CFEE movzx   edi, word ptr [ebx+0Ch]
.text:0122CFF2 imul    ecx, 1000193h
.text:0122CFF8 xor     ecx, esi
.text:0122CFFA movzx   esi, word ptr [ebx+0Ah]
.text:0122CFFE imul    ecx, 1000193h
.text:0122D004 xor     ecx, edi
.text:0122D006 mov     edi, 0E97D6B6Fh
.text:0122D00B imul    ecx, 1000193h
.text:0122D011 xor     ecx, esi
.text:0122D013 imul    esi, ecx, 1000193h
.text:0122D019 test    al, 1
.text:0122D01B mov     ecx, 7F86D431h
.text:0122D020 cmovz   edi, ecx
.text:0122D023 cmp     edx, 0Ah
.text:0122D026 cmovl   edi, ecx
.text:0122D029 movzx   ecx, word ptr [ebx+8]
.text:0122D02D mov     [esp+24h+var_18], ecx
.text:0122D031 mov     ecx, 82C1E0F1h
.text:0122D036 jmp     short loc_122D05C
```

Figure 5

The GetCommandLineW routine is used to extract the command-line string for the current process:

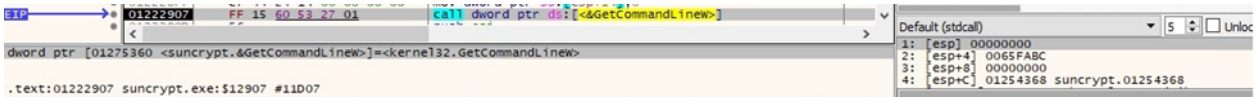


Figure 6

The malicious process retrieves an array of pointers to the command line arguments, along with a count of the arguments via a function call to CommandLineToArgvW:

Figure 7

OpenProcessToken is utilized to open the access token associated with the current process (0x20 = **TOKEN_ADJUST_PRIVILEGES**):

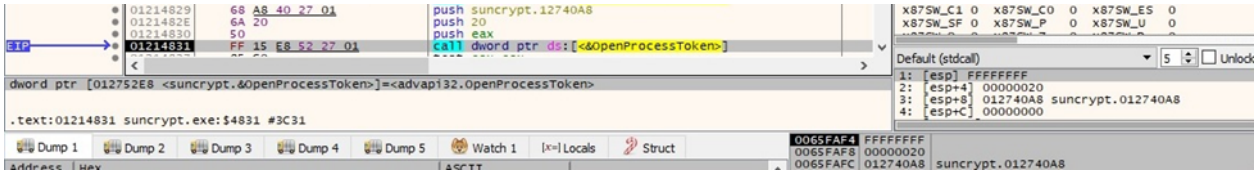


Figure 8

The malware performs multiple calls to LookupPrivilegeValueA in order to extract the locally unique identifier (LUID) for the following privileges: "SeTakeOwnershipPrivilege", "SeBackupPrivilege", "SeSecurityPrivilege", "SeRestorePrivilege", "SeDebugPrivilege", "SeImpersonatePrivilege", and "SeIncreaseBasePriorityPrivilege". Figure 9 displays an example of a function call:

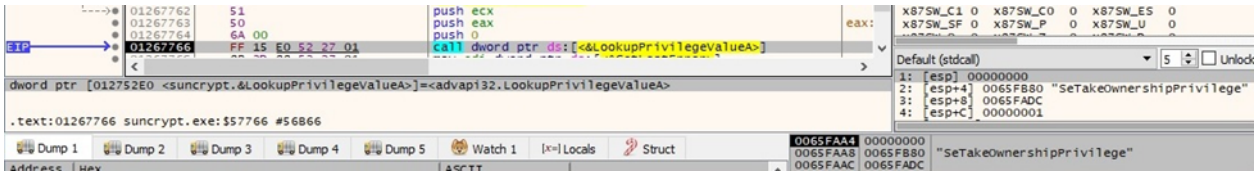


Figure 9

The malicious executable enables the above privileges using the AdjustTokenPrivileges function:

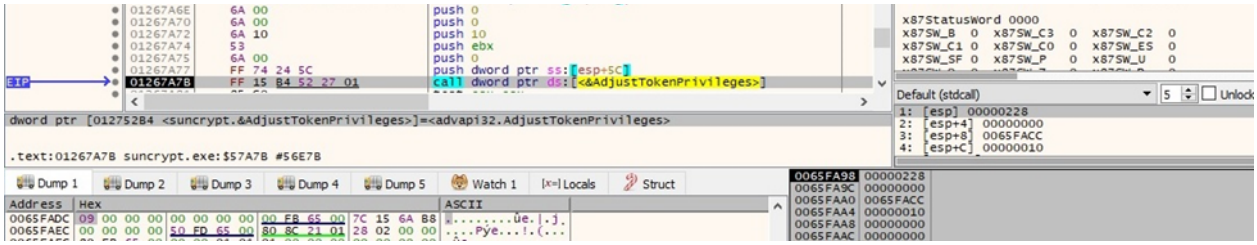


Figure 10

SunCrypt retrieves information about the current system via a function call to GetSystemInfo:

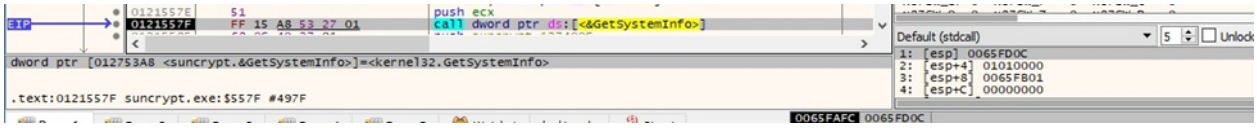


Figure 11

The GetModuleHandleA routine is utilized to extract a module handle for "ntdll.dll":

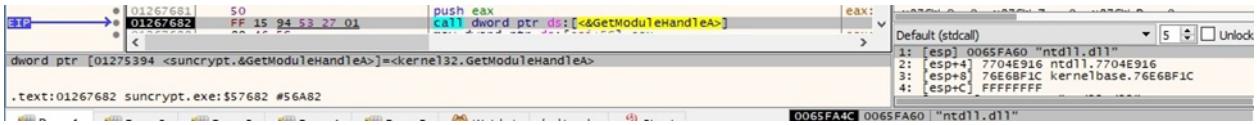


Figure 12

The malware retrieves version information about the operating system by calling the RtlGetVersion routine:

Figure 13

The binary creates a mutex called "0c91c96fd7124f21a0193cf842e3495f6daf84a394f44013e92a87ad9d2ef4a0ceec9dd2e2eca22e" in order to ensure that only one copy of the executable is running at a single time:

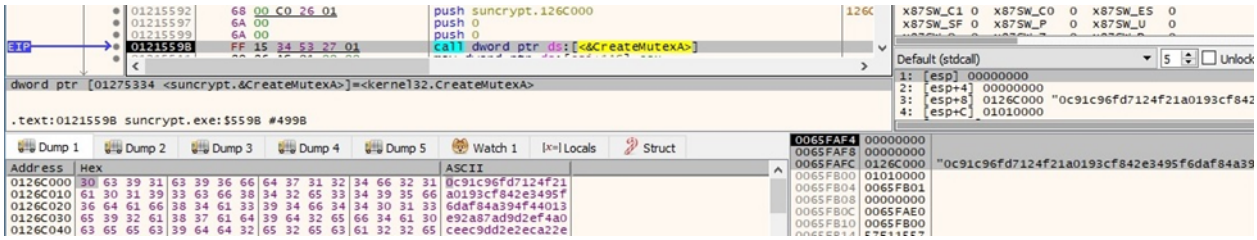


Figure 14

The executable takes a snapshot of all processes in the system using CreateToolhelp32Snapshot (0x2 = **TH32CS_SNAPPROCESS**):

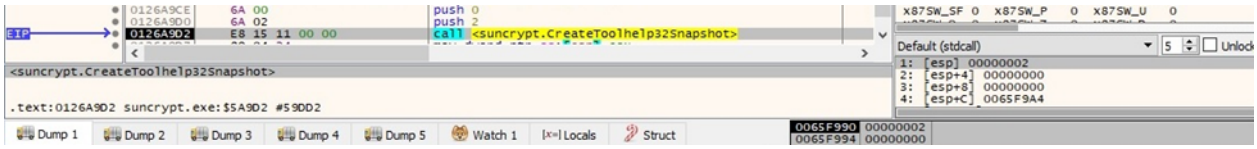


Figure 15

The processes are enumerated using the Process32First and Process32Next APIs:

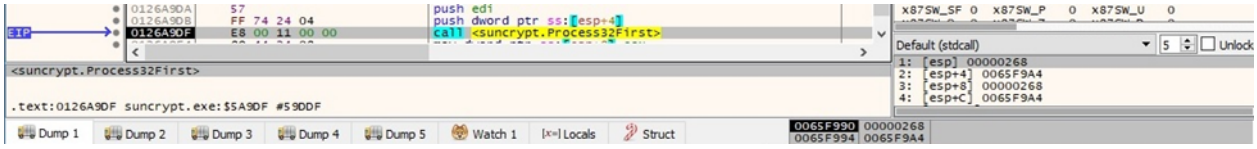


Figure 16

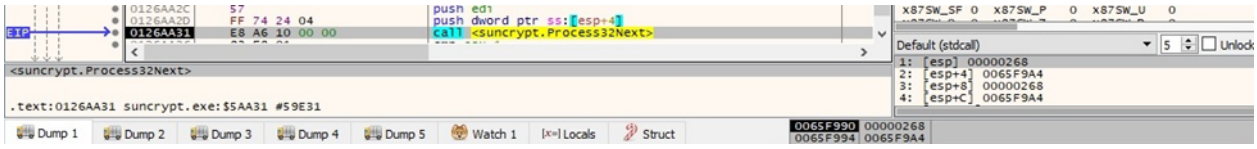


Figure 17

SunCrypt targets a list of processes that will be killed:

- "ocssd" "dbsnmp" "synctime" "agntsvc" "isqlplussvc" "xfssvccon" "mydesktopservice" "ocautoupds" "encsvc" "firefox" "tbirdconfig"
- "mydesktopqos" "ocomm" "dbeng50" "sqbcoreservice" "excel" "infopath" "msaccess" "mspub" "onenote" "outlook" "powerpnt" "steam"
- "thebat" "thunderbird" "visio" "winword" "wordpad" "ssms" "notepad" "fdhost" "fdlauncher" "launchpad" "sqlceip" "sqlwriter"

The comparison between a process name and one of the above processes is employed using StrStrIA:

Figure 18

The ransomware opens a targeted process via a function call to OpenProcess (0x1FFFFF = **PROCESS_ALL_ACCESS**):

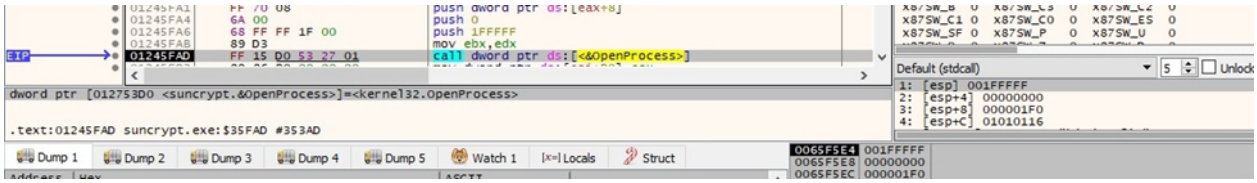


Figure 19

The TerminateProcess routine is used to kill a targeted process:

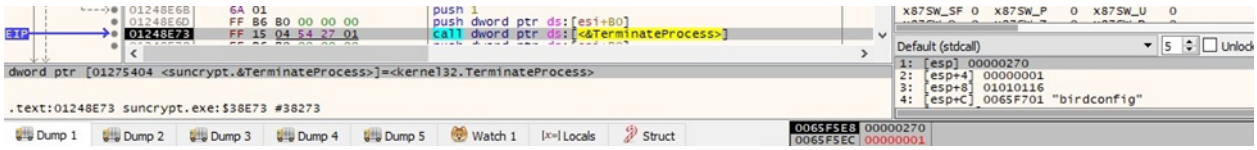


Figure 20

The malicious file tries to locate the "winlogon.exe" process:

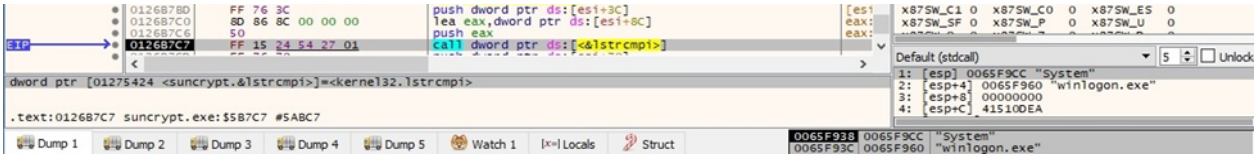


Figure 21

OpenProcess is used to retrieve a handle to the above process (0x400 = **PROCESS_QUERY_INFORMATION**):

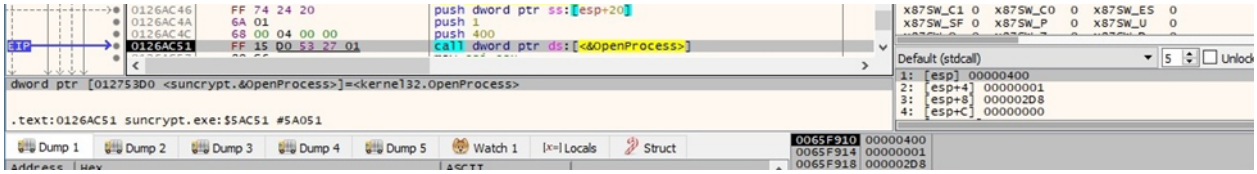


Figure 22

The executable opens the access token associated with “winlogon.exe” (0xF = **TOKEN_ASSIGN_PRIMARY** | **TOKEN_DUPLICATE** | **TOKEN_IMPERSONATE** | **TOKEN_QUERY**):

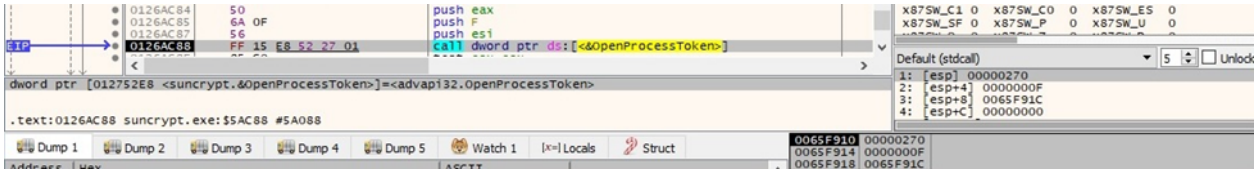


Figure 23

The DuplicateTokenEx API is used to create a new access token that duplicates the token extracted above (0x2000000 = **MAXIMUM_ALLOWED**, 0x2 = **SecurityIdentification**, 0x2 = **TokenImpersonation**):

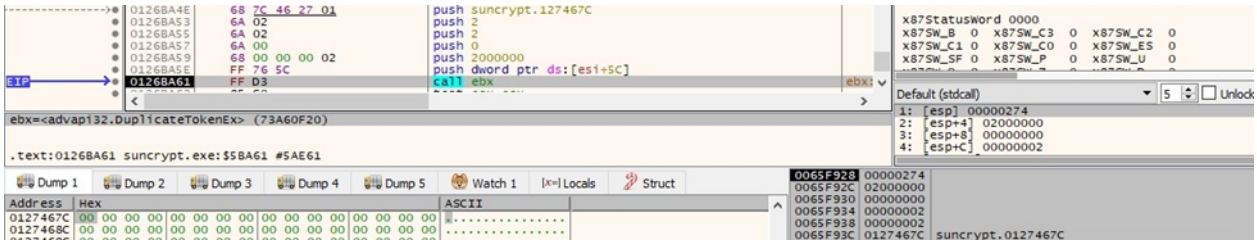


Figure 24

The process assigns the impersonation token to the calling thread using SetThreadToken:

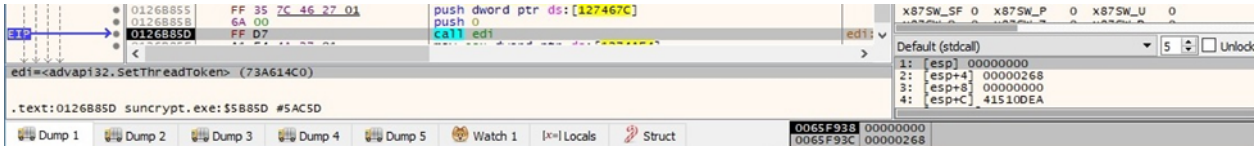


Figure 25

SunCrypt sets the highest possible priority for the current process (0x100 = **REALTIME_PRIORITY_CLASS**):

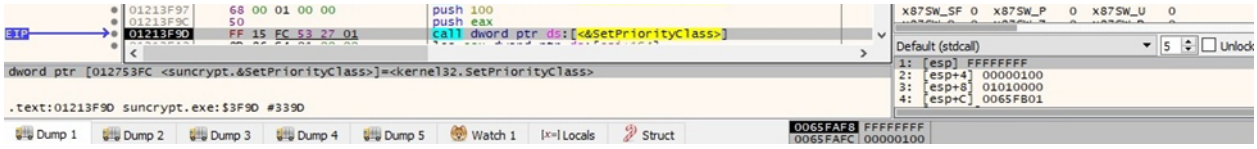


Figure 26

The process I/O priority is set to 3 (High) via a function call to ZwSetInformationProcess (0x21 = **ProcessIoPriority**):

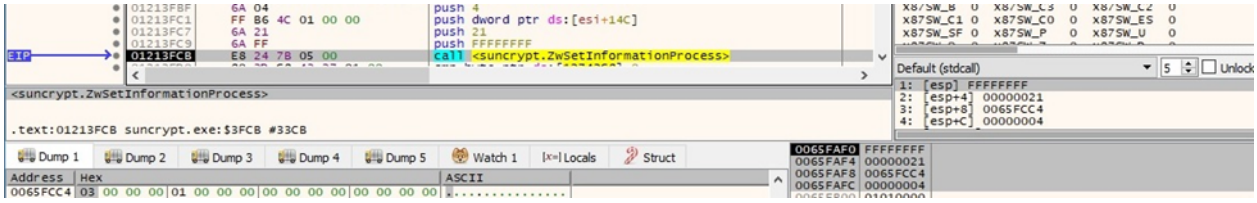


Figure 27

A new thread is created by calling the CreateThread API:

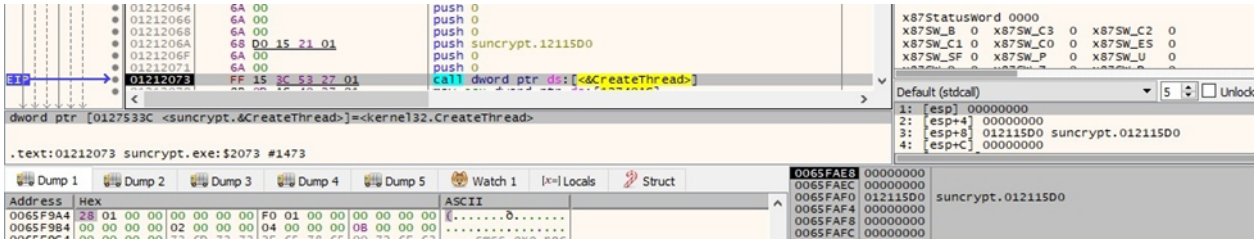


Figure 28

There is a function call to RevertToSelf that terminates the impersonation.

Delete Volume Shadow Copies

CoInitialize is used to initialize the COM library on the current thread:

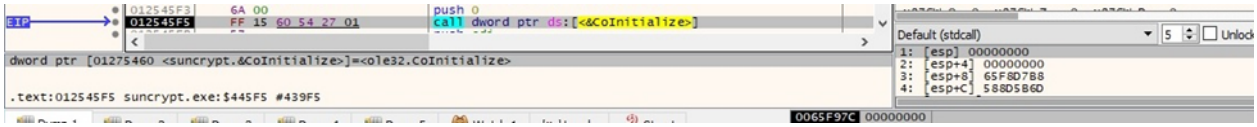


Figure 29

The ransomware creates an IWbemContext Interface by calling the CoCreateInstance API with the {674B6698-EE92-11D0-AD71-00C04FD8FDFF} parameter:

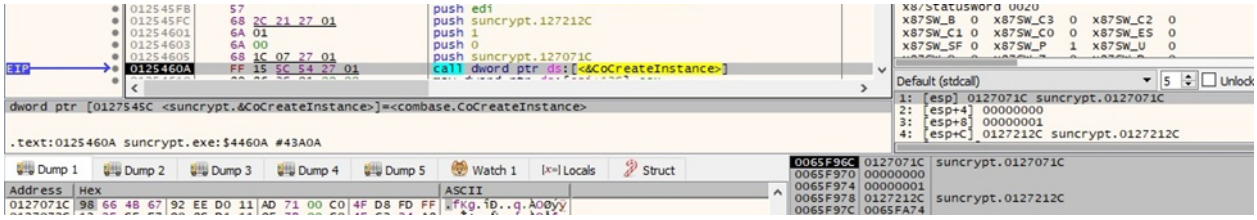


Figure 30

The IsWow64Process routine is utilized to determine whether the current process is running on a 64-bit environment:

Figure 31

The malware creates an IWbemLocator object with the CLSID {4590f811-1d3a-11d0-891f-00aa004b2e24}":

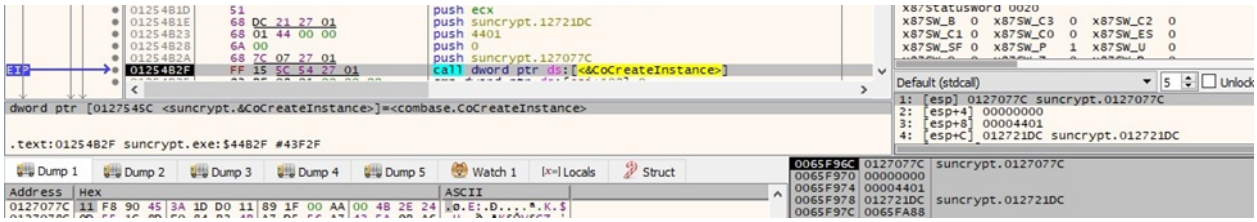


Figure 32

The malicious executable calls the ConnectServer function for connecting to the local "ROOT\CIMV2" namespace:

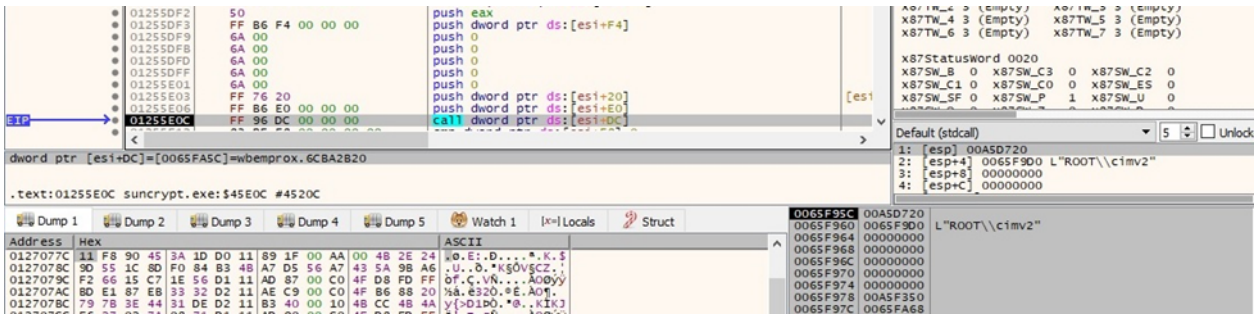


Figure 33

There is a function call to CoSetProxyBlanket that sets the authentication information used to make calls on a proxy (0xA = **RPC_C_AUTHN_WINNT**, 0x3 = **RPC_C_AUTHN_LEVEL_CALL**, 0x3 = **RPC_C_IMP_LEVEL_IMPERSONATE**):

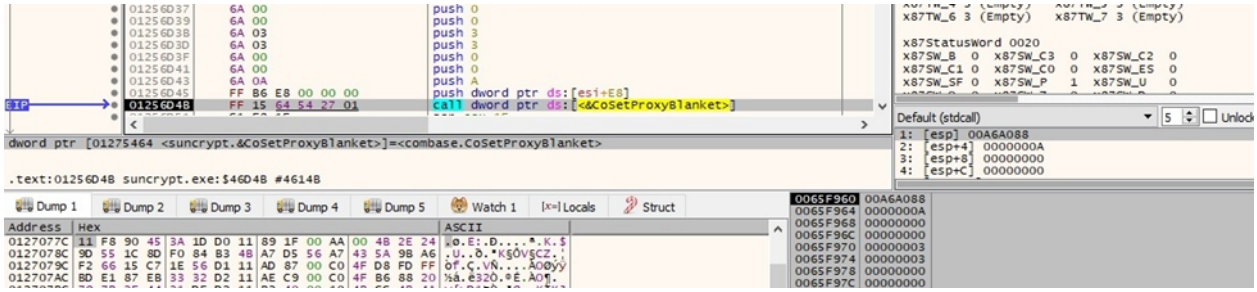


Figure 34

The malware retrieves an enumerator of all shadow copies using the following WQL query "SELECT * FROM Win32_ShadowCopy":

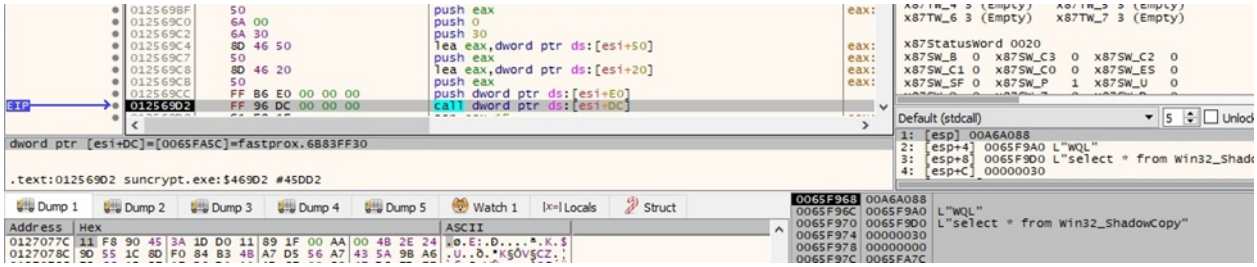


Figure 35

The id property value of a specific shadow copy is extracted using the IwbemClassObject::Get method:

Figure 36

The Volume Shadow Copies are deleted using the DeleteInstance function:

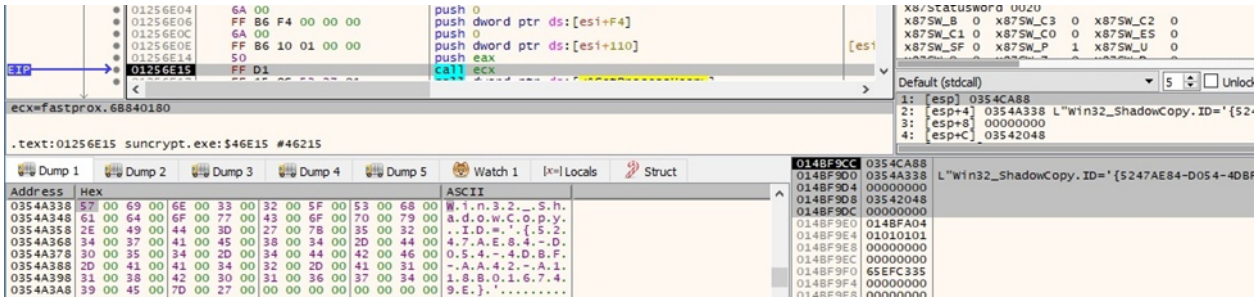


Figure 37

SunCrypt utilizes multithreading with I/O completion ports when encrypting files. The main purpose is to establish a communication between the main thread and the worker threads that are responsible for files encryption.

The ransomware creates an I/O completion port that is not yet associated with a file handle using CreateIoCompletionPort:

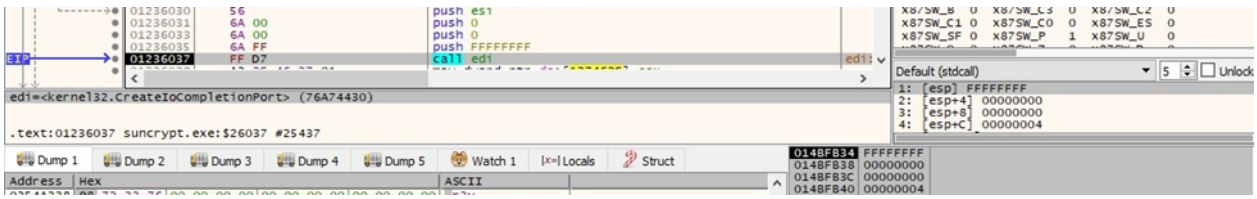


Figure 38

The binary creates 4 (2 * number of cores) threads that will handle the files encryption (the IOCP handle is passed as a parameter):

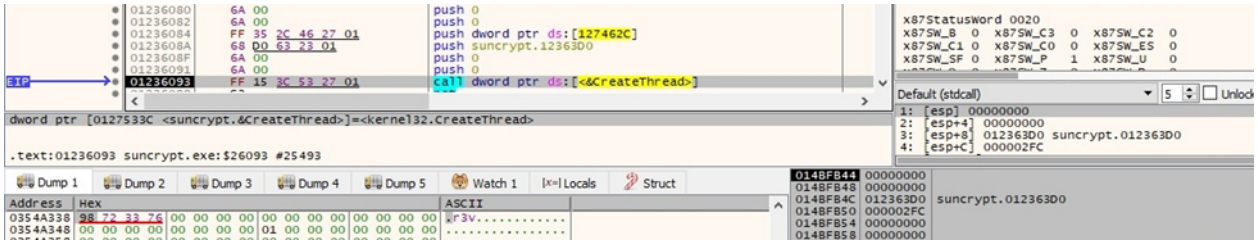


Figure 39

The GetLogicalDrives API is used to extract a bitmask representing the available disk drives:

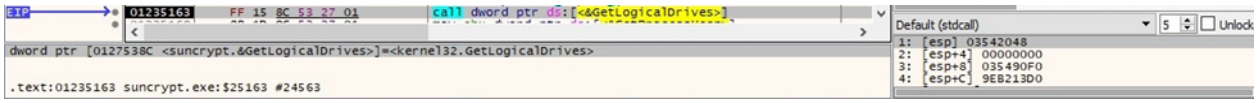


Figure 40

The drive type is retrieved via a call to GetDriveTypeW. It expects a return value that is less or equal to 5:

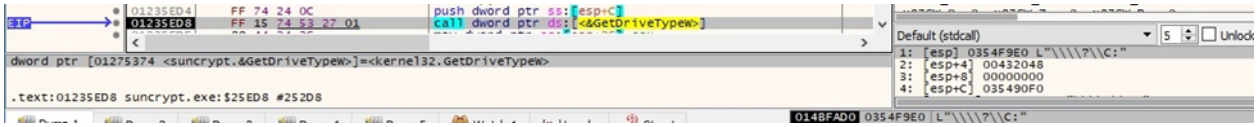


Figure 41

SunCrypt verifies whether the Windows Boot Manager (bootmgr) file is present in any of the extracted drives using the GetFileAttributesW routine:

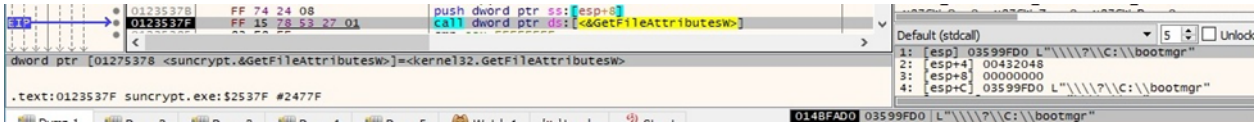


Figure 42

For example, the above file exists in the C drive, and this one will not be encrypted by the malware. This operation is unusual for most of the ransomware families, because other families choose to whitelist specific directories (“Program Files”) rather than avoiding to encrypt the drive completely.

The malware verifies the presence of a boot file called bootmgr.efi:

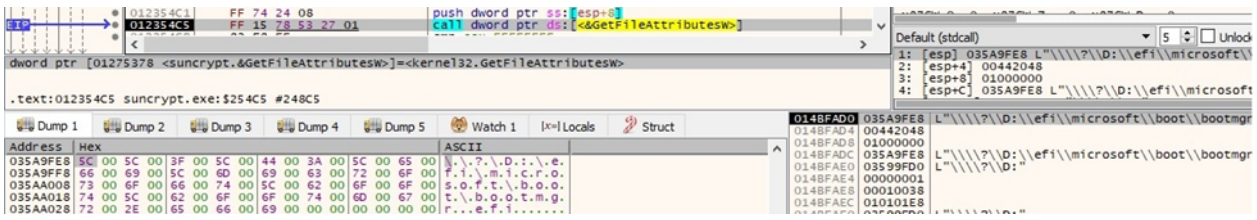


Figure 43

The CreateThread function is used to create a new thread that will traverse the targeted drive (see figure 44). It’s important to mention that there is no checking to determine if the drive is

empty or not (for example, the D drive might correspond to CD/DVD drive).

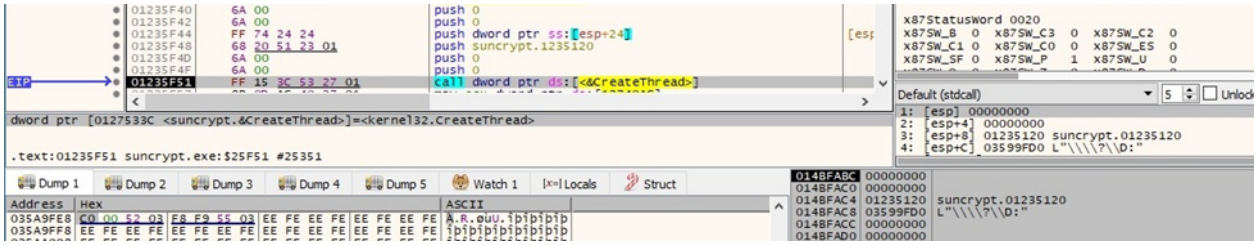


Figure 44

Thread activity – sub_1235120 function

There is a comparison between the drive name and the "\\AppData" or "\\Application Data" strings:

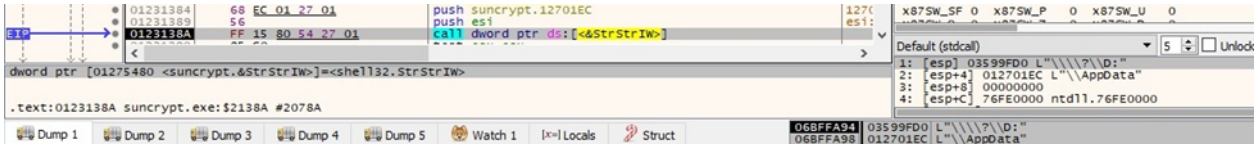


Figure 45

SetFileAttributesW is utilized to set an attribute for the drive (0x80 = **FILE_ATTRIBUTE_NORMAL**):

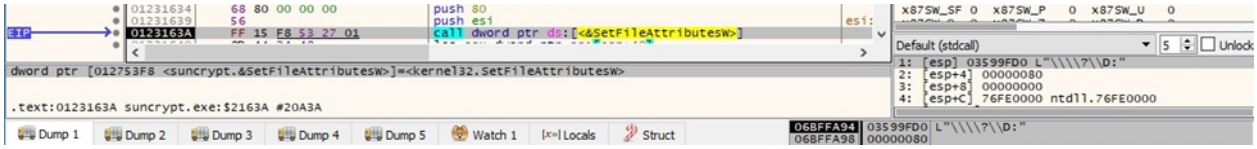


Figure 46

The ransomware enumerates the above drive using FindFirstFileW (figure 47); however, it corresponds to the DVD drive and it's empty. This execution flow will be explained in detail when encrypting network shares.

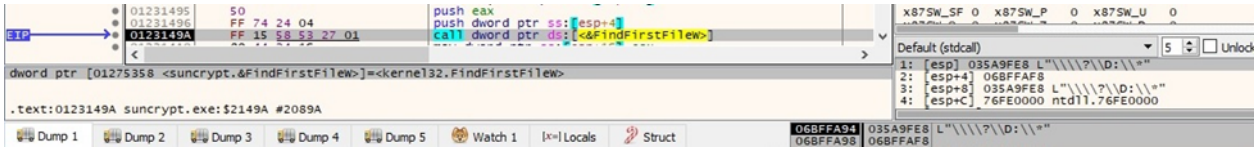


Figure 47

We continue with the analysis of the main thread.

The malware starts to enumerate the network resources via a function call to WNetOpenEnumW (0x2 = **RESOURCE_GLOBALNET**, 0x0 = **RESOURCETYPE_ANY**, 0x13 = **RESOURCEUSAGE_ALL**):

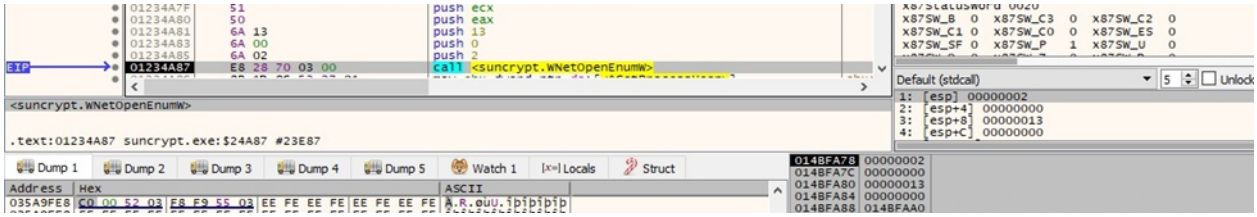


Figure 48

The enumeration of network resources continues by calling the WNetEnumResourceW API:

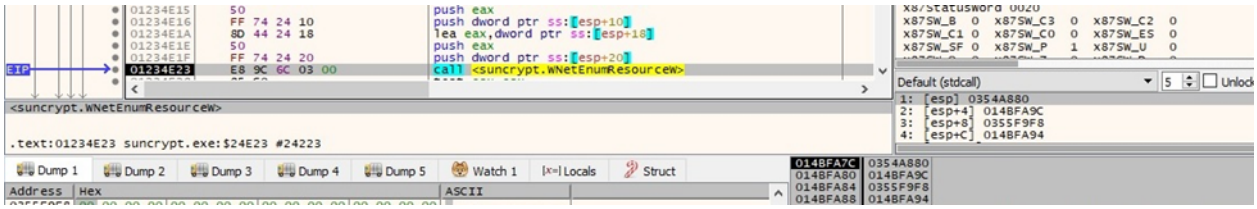


Figure 49

The binary makes a connection to a network share using the WNetAddConnection2W routine:

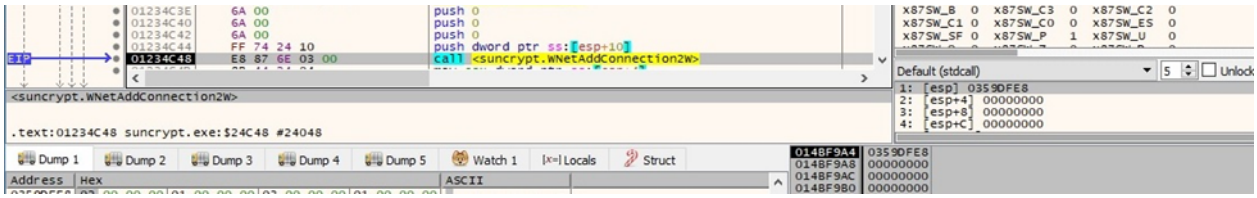


Figure 50

SunCrypt starts enumerating a network share using FindFirstFileW:

Figure 51

A file extension is extracted by calling the PathFindExtensionW function:

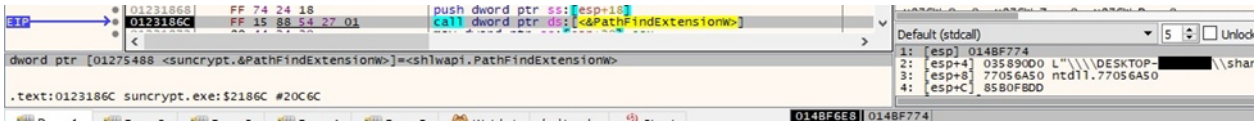


Figure 52

The files that have the following extensions will be skipped: ".exe", ".dll", ".ocx", and ".sys". An example of such comparison is displayed in figure 53:

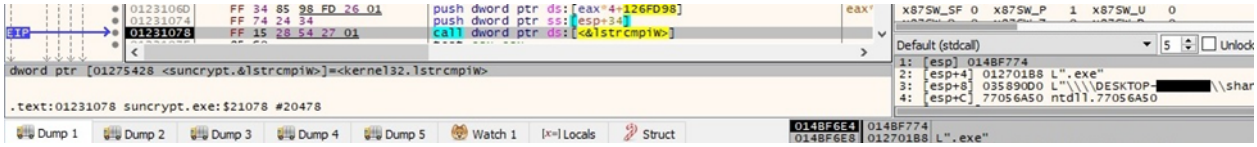


Figure 53

The following directories/files will not be encrypted: "windows", "$Recycle.bin", "System Volume Information", "ntldr", "ntdetect.com", "bootfont.bin", "boot.ini", and "YOUR_FILES_ARE_ENCRYPTED.HTML". An example of such comparison is displayed below:

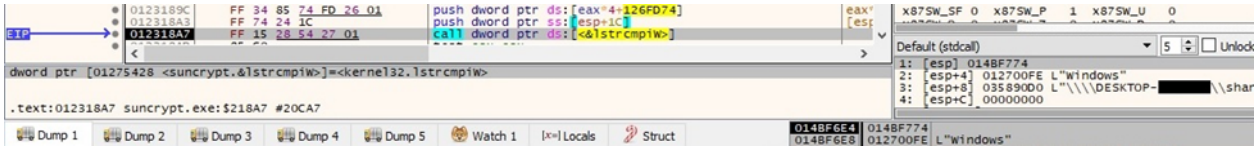


Figure 54

The file enumeration continues by calling the FindNextFileW routine:

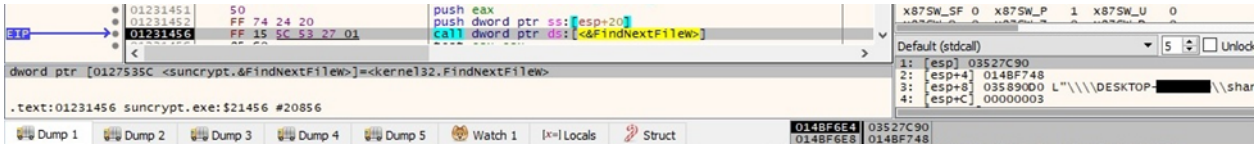


Figure 55

SunCrypt generates 32 random bytes by calling the SystemFunction036 function:

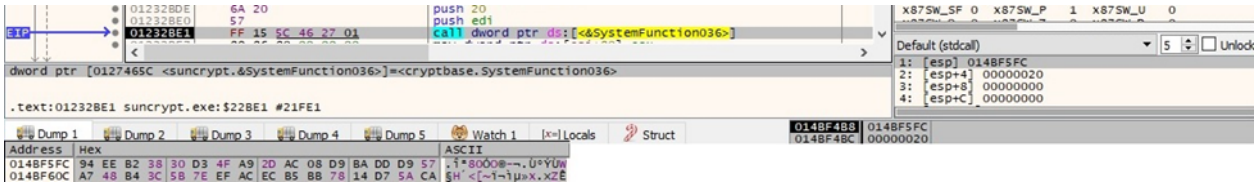


Figure 56

This buffer represents a 32-byte secret key for Curve25519 (ECC algorithm). The ransomware jumps to the curve function that is used to compute the session public key (observe a base point of 09 followed by all zeros):

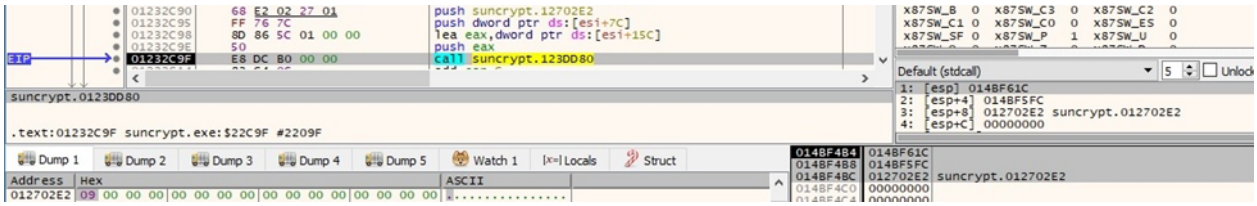


Figure 57

The capa tool identifies the implementation of the Curve25519 algorithm (see figure 58). The session public key computed above is shown in figure 59.

```
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 @ 0x422C6E in function 0x421AA0
  and:
    and:
      number: 0xF8 @ 0x422C7C
      mnemonic: and @ 0x422C7C, 0x422C86
    and:
      number: 0x3F @ 0x422C86
      mnemonic: and @ 0x422C7C, 0x422C86
    and:
      number: 0x40 @ 0x422C88
      mnemonic: or @ 0x422C88
basic block @ 0x42DD80 in function 0x42DD80
  and:
    and:
      number: 0xF8 @ 0x42DDD4
      mnemonic: and @ 0x42DD86, 0x42DDD4, 0x42DDF3, 0x42DE2F, and 1 more...
    and:
      number: 0x3F @ 0x42DDF3
      mnemonic: and @ 0x42DD86, 0x42DDD4, 0x42DDF3, 0x42DE2F, and 1 more...
    and:
      number: 0x40 @ 0x42DDF6
      mnemonic: or @ 0x42DDF6, 0x42DE11, 0x42DE1A, 0x42DE20, and 15 more...
```

Figure 58

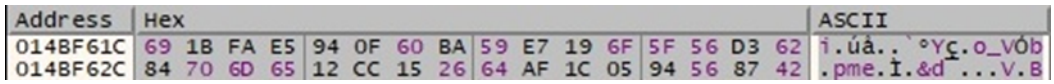


Figure 59

The session public key is appended to the file chosen for encryption:

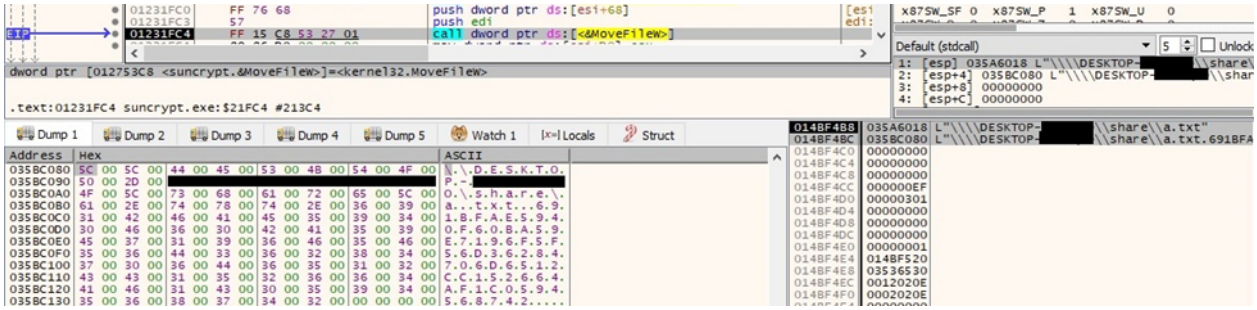


Figure 60

The ransomware opens the newly modified file using CreateFileW (0xc0010000 = **GENERIC_READ** | **GENERIC_WRITE** | **DELETE**, 0x1 = **FILE_SHARE_READ**, 0x3 = **OPEN_EXISTING**, 0x50000000 = **FILE_FLAG_OVERLAPPED** | **FILE_FLAG_RANDOM_ACCESS**):

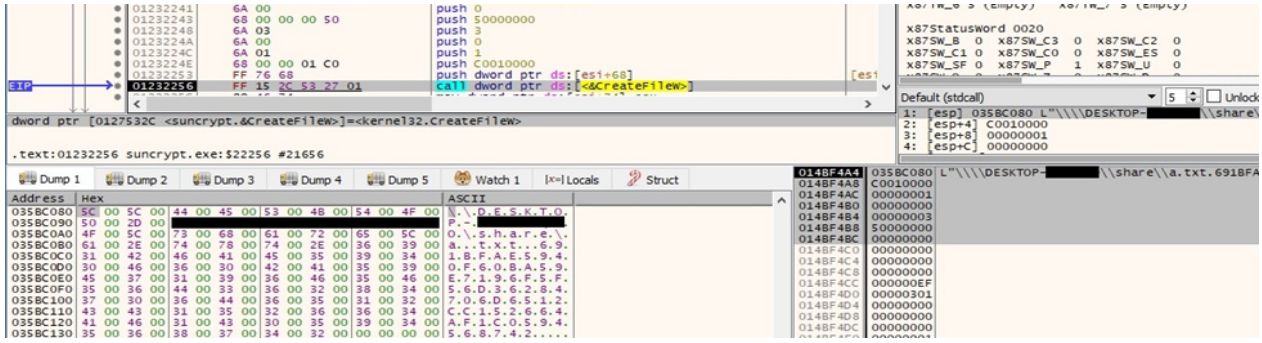


Figure 61

SunCrypt comes with a hard-coded Curve25519 public key:

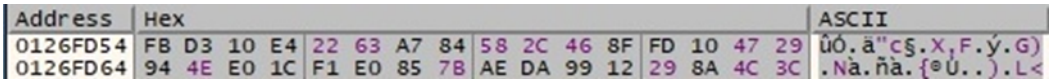


Figure 62

The ransomware computes a shared secret between the above key and the generated session public key using the Curve25519 algorithm:

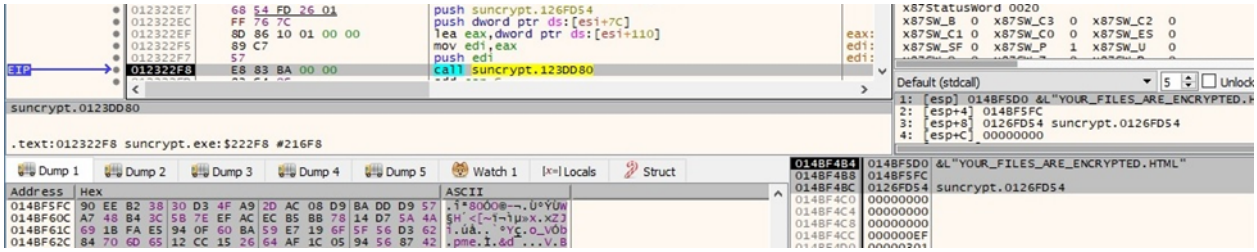


Figure 63

The shared secret is a 32-byte buffer that will be used to encrypt the targeted file using the ChaCha algorithm, as we will describe later on:

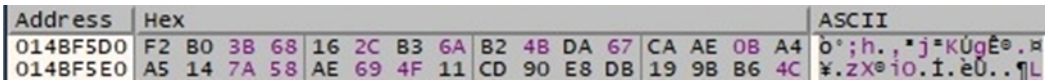


Figure 64

The binary adds the “expand 32-byte k” string to the above buffer, which suggests that the encryption algorithm will be Salsa20 or ChaCha:

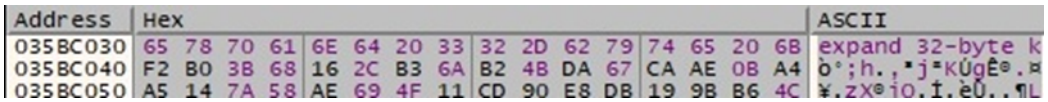


Figure 65

SunCrypt associates the IOCP created earlier with the targeted file handle using the CreateIoCompletionPort API:

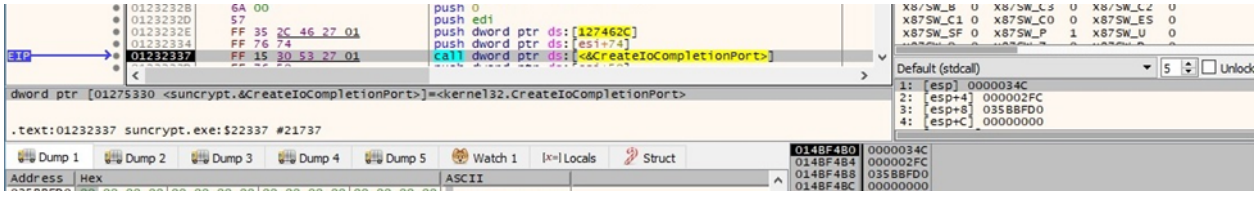


Figure 66

PostQueuedCompletionStatus is utilized to send an I/O completion packet to the IOCP:

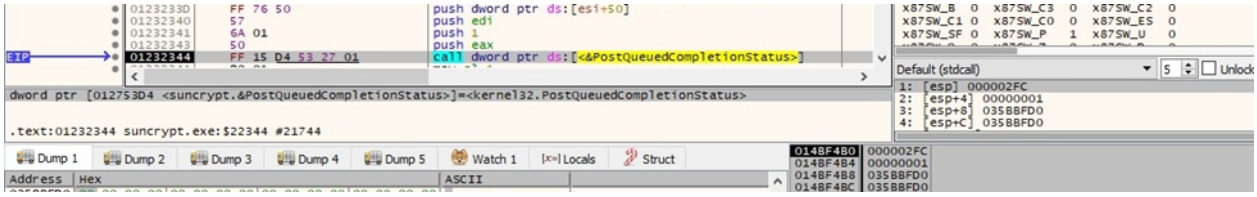


Figure 67

The ransomware creates a ransom note called "YOUR_FILES_ARE_ENCRYPTED.HTML" in every directory (0x40000000 = **GENERIC_WRITE**, 0x1 = **FILE_SHARE_READ**, 0x1 = **CREATE_NEW**):

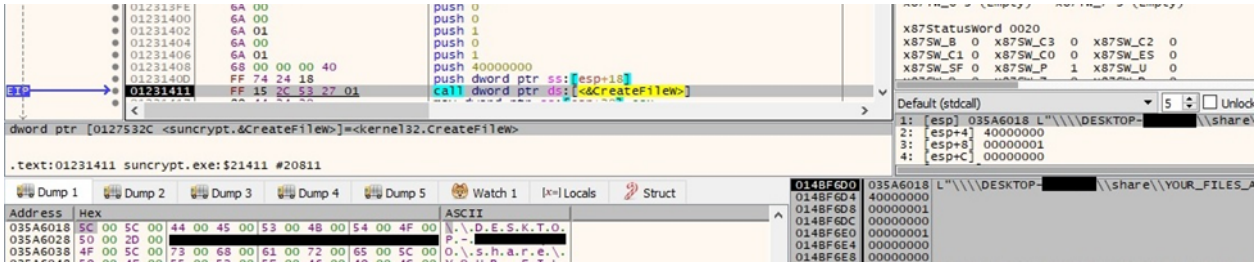
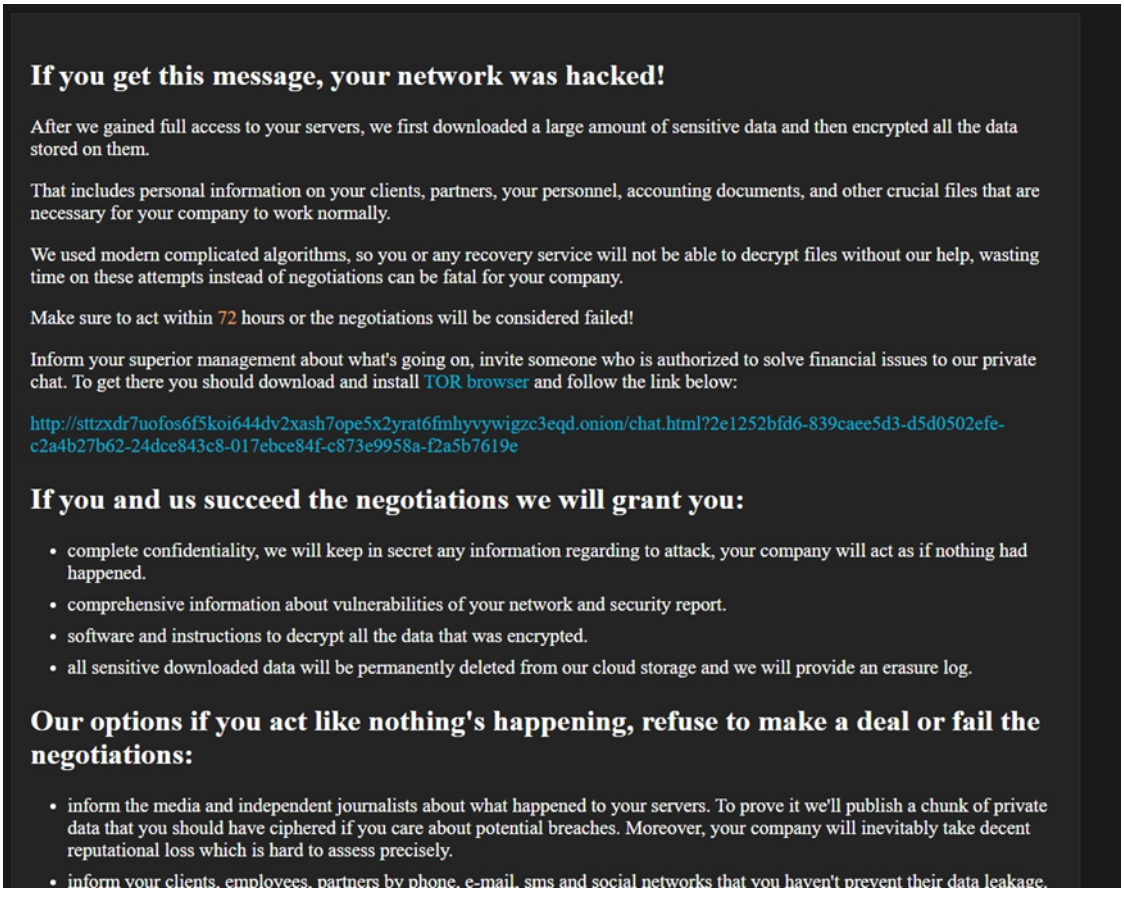


Figure 68

The ransom note is displayed below:



If you get this message, your network was hacked!

After we gained full access to your servers, we first downloaded a large amount of sensitive data and then encrypted all the data stored on them.

That includes personal information on your clients, partners, your personnel, accounting documents, and other crucial files that are necessary for your company to work normally.

We used modern complicated algorithms, so you or any recovery service will not be able to decrypt files without our help, wasting time on these attempts instead of negotiations can be fatal for your company.

Make sure to act within 72 hours or the negotiations will be considered failed!

Inform your superior management about what's going on, invite someone who is authorized to solve financial issues to our private chat. To get there you should download and install TOR browser and follow the link below:

http://sttzxdr7uofos6f5koi644dv2xash7ope5x2yrat6fmhyvywigzc3eqd.onion/chat.html?2e1252bfd6-839caee5d3-d5d0502efe-c2a4b27b62-24dce843c8-017ebce84f-c873e9958a-f2a5b7619e

If you and us succeed the negotiations we will grant you:

- complete confidentiality, we will keep in secret any information regarding to attack, your company will act as if nothing had happened.
- comprehensive information about vulnerabilities of your network and security report.
- software and instructions to decrypt all the data that was encrypted.
- all sensitive downloaded data will be permanently deleted from our cloud storage and we will provide an erasure log.

Our options if you act like nothing's happening, refuse to make a deal or fail the negotiations:

- inform the media and independent journalists about what happened to your servers. To prove it we'll publish a chunk of private data that you should have ciphered if you care about potential breaches. Moreover, your company will inevitably take decent reputational loss which is hard to assess precisely.
- inform your clients, employees, partners by phone, e-mail, sms and social networks that you haven't prevent their data leakage.

Figure 69

Thread activity – sub_12115D0 function

The malicious process retrieves a handle that can be used to enumerate the list of channels that are registered on the local computer via a call to EvtOpenChannelEnum:

Figure 70

The enumeration starts by extracting a channel name from the enumerator using EvtNextChannelPath:

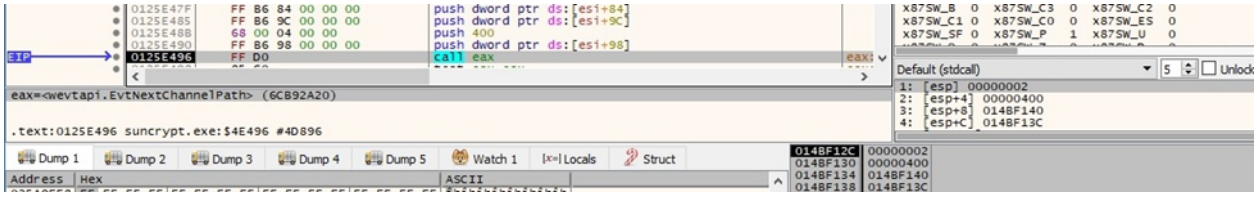


Figure 71

The purpose of the malware is to clear the event logs using the EvtClearLog routine:

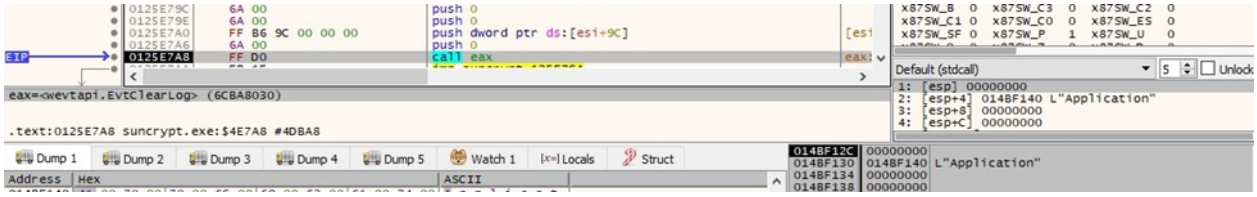


Figure 72

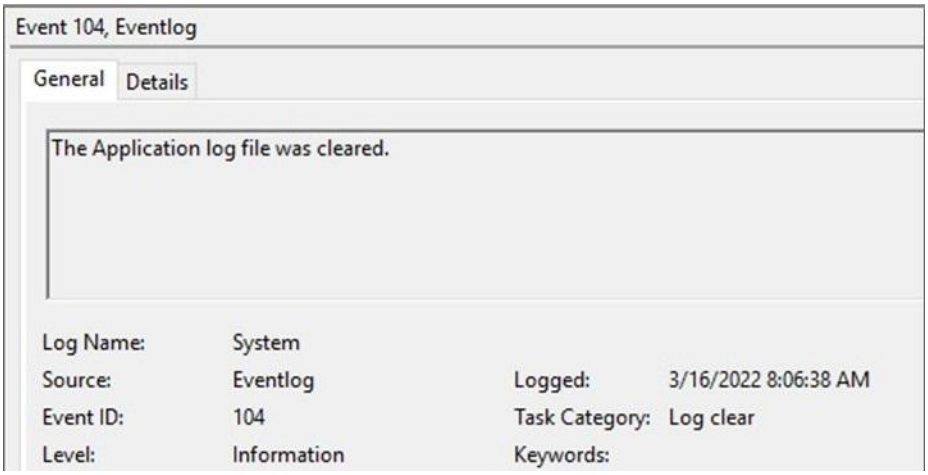


Figure 73

The channels enumeration continues using the same API as above:

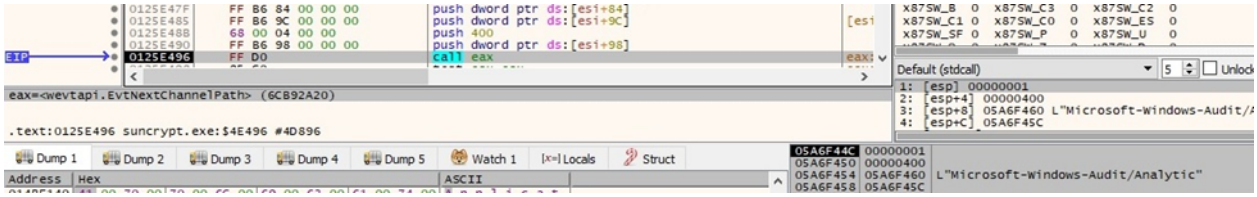


Figure 74

The ransomware opens the "SYSTEM\CurrentControlSet\Services\EventLog" registry key using the RegOpenKeyA function (0x80000002 = **HKEY_LOCAL_MACHINE**):

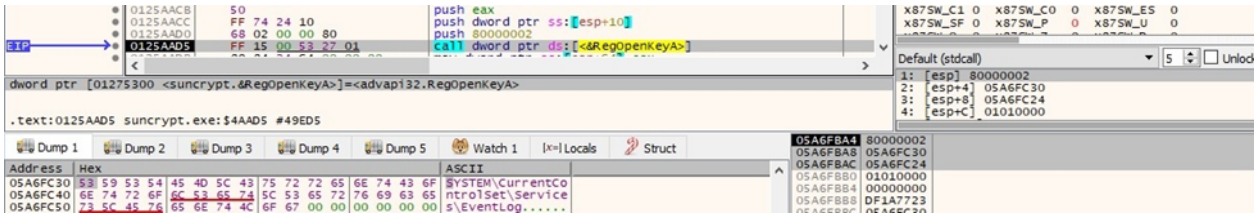


Figure 75

The file enumerates the subkeys of the above registry key using RegEnumKeyA:

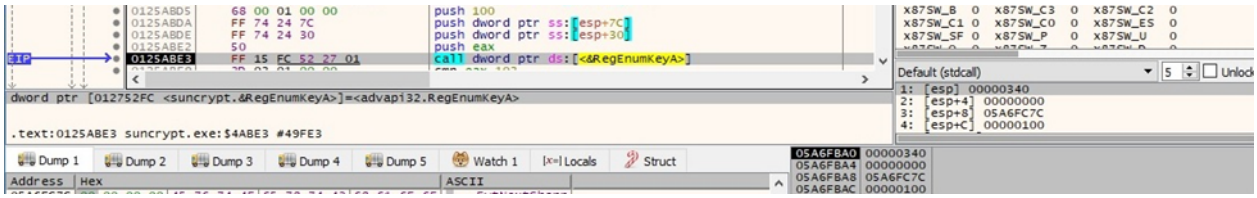


Figure 76

SunCrypt opens a handle to the “Application” event log via a function call to OpenEventLogA:

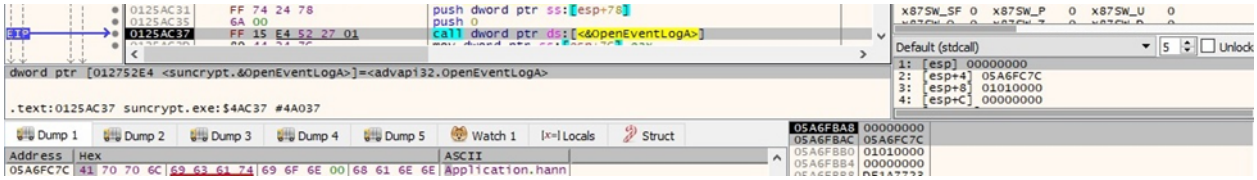


Figure 77

The malware clears the “Application” event log using the ClearEventLogA API. This is the 2nd method employed by SunCrypt to clear all event logs:

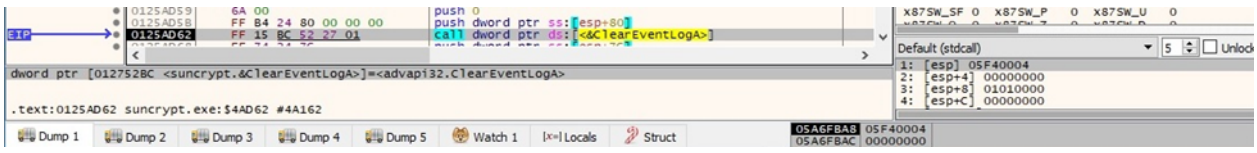


Figure 78

Thread activity – sub_12363D0 function

A worker thread responsible for file encryption dequeues an I/O completion packet from the IOCP using GetQueuedCompletionStatus:

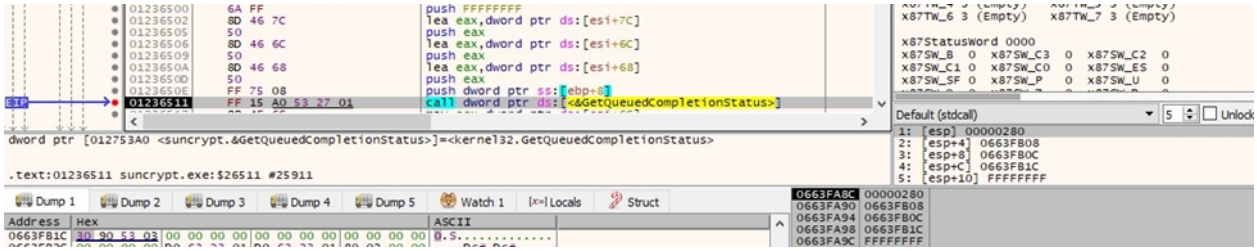


Figure 79

SunCrypt passes the ChaCha20 key along with the encrypted file name to the encryption function:

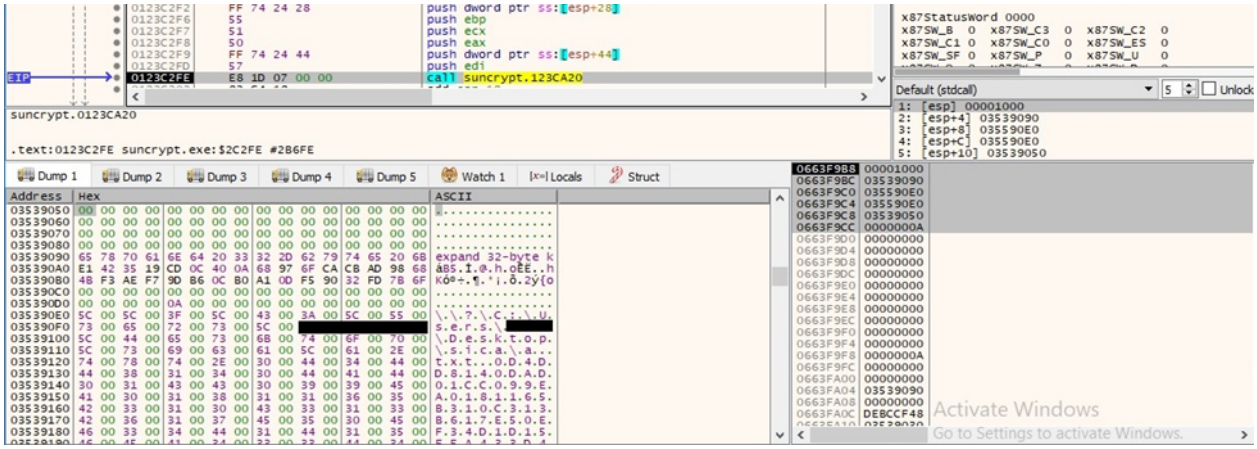


Figure 80

The ChaCha20 algorithm implementation is manual, and it doesn't rely on Windows APIs, as highlighted below:

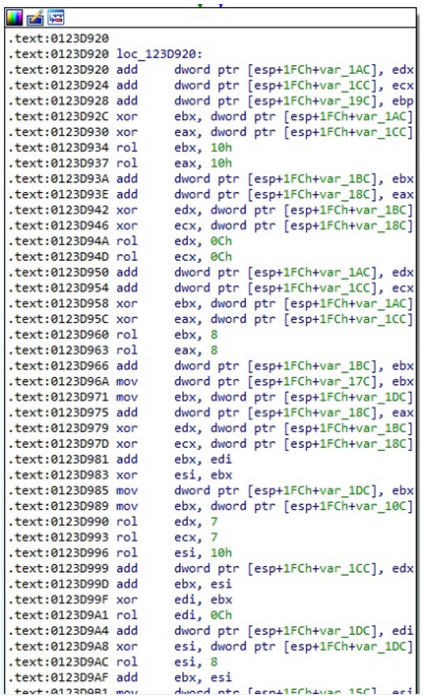


Figure 81

The encrypted content is written to the file by calling the WriteFile API (see figure 82). The targeted files should be at least 512 bytes long; otherwise they will not be encrypted by SunCrypt.

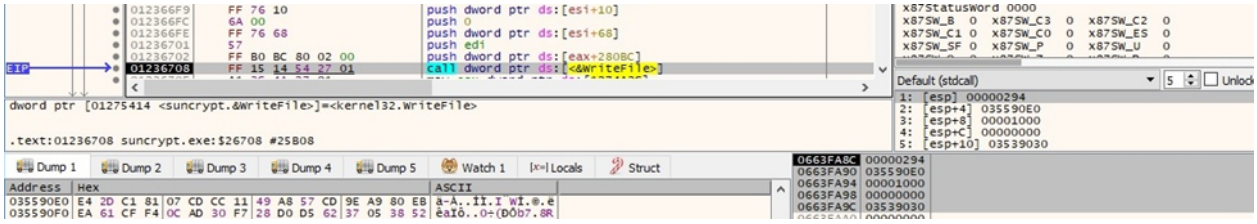


Figure 82

We continue with the analysis of the main thread. It's worth mentioning that the events log deletion operation is repeated in the main thread with an identical execution flow.

The ransomware extracts the path of the current executable using GetModuleFileNameW:

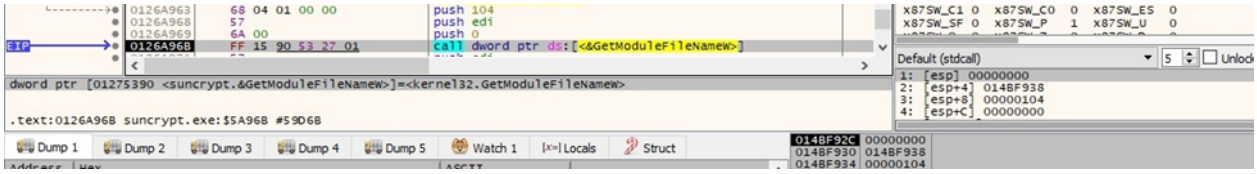


Figure 83

The ransomware deletes itself via a function call to CreateProcessW (0x8000000 = **CREATE_NO_WINDOW**):

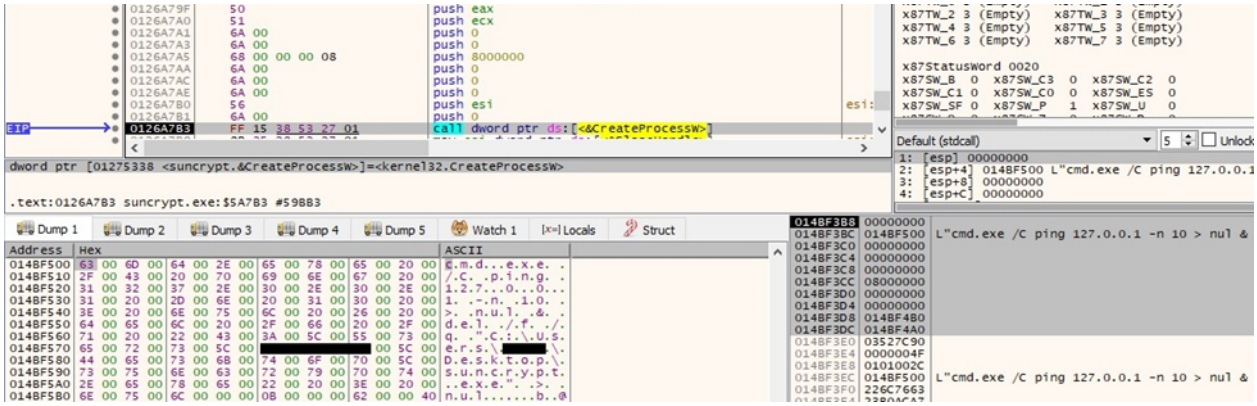


Figure 84

We want to provide some observations regarding the usage of command-line parameters.

Parameter	Explanation
-nomutex	No difference in execution
-noservices	No difference in execution
-noreport	No difference in execution
-vm	No difference in execution
-path	Encrypt a single directory
-noshares	Do not encrypt network shares
-keep_exe	Do not delete the executable
-justcrypt	Do not kill the targeted processes. Do not delete the Volume Shadow Copies

SunCrypt proves that a relatively low-level complexity code could still produce significant damages. As opposed to ransomware families such as LockBit or Conti, the encryption of a system takes tens of minutes and can be detected by monitoring the CPU usage for a longer time period.

Indicators of Compromise

Mutex

0c91c96fd7124f21a0193cf842e3495f6daf84a394f44013e92a87ad9d2ef4a0ceec9dd2e2eca22e

SunCrypt Ransom Note

YOUR_FILES_ARE_ENCRYPTED.HTML

Processes spawned

cmd.exe /C ping 127.0.0.1 -n 10 > nul & del /f /q \"<Path to executable>" > nul