A Detailed Analysis of the Quantum Ransomware

Prepared by: Vlad Pasca, Senior Malware & Threat Analyst

SecurityScorecard.com
info@securityscorecard.com

Tower 49
12 E 49th Street
Suite 15-001
New York, NY 10017
1.800.682.1707

Table of contents

Executive summary

Quantum ransomware, a rebrand of the MountLocker ransomware, was discovered in August 2021. The malware stops a list of processes and services, and can encrypt the machines found in the Windows domain or the local network, as well as the network shared resources. It logs all of its activities in a file called ".log" and computes a Client Id that is the XOR-encryption of the computer name.

The files are encrypted using the ChaCha20 algorithm, with the key being encrypted using a global ChaCha20 key, which is eventually encrypted with a public RSA-2048 key. The extension of the encrypted files is changed to .quantum by the ransomware.

Analysis and findings

SHA256: 91E66F0EDFA5F0277E127B599517B497CF0204B181F32CE1AAB8F9FAA749EC40

The malware is a 64-bit executable that uses the XOR algorithm to decrypt a DLL file, as highlighted below:

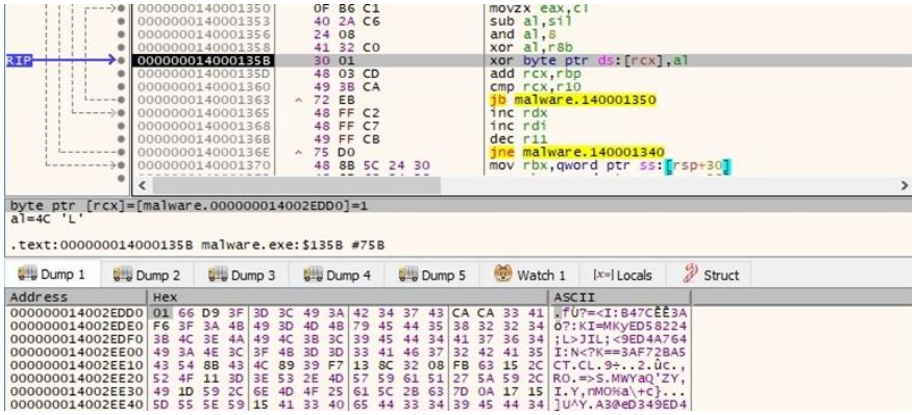


Figure 1

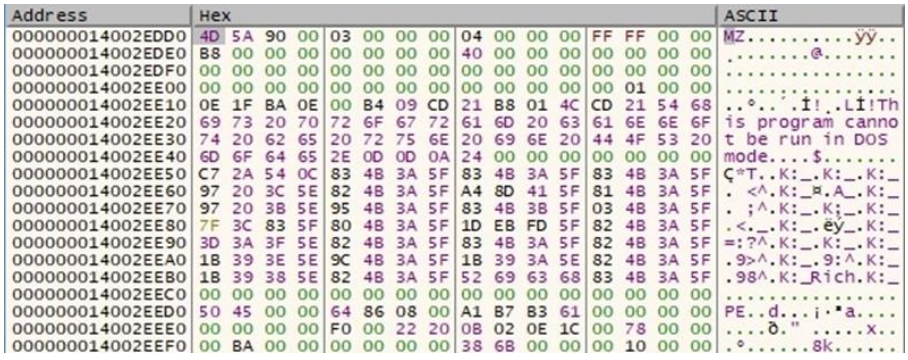


Figure 2

The GetNativeSystemInfo API is utilized to retrieve information about the current system:

Figure 3

The binary allocates new memory areas by calling the VirtualAlloc function (0x3000 = **MEM_COMMIT** | **MEM_RESERVE**, 0x4 = **PAGE_READWRITE**):

Figure 4

The executable loads the following DLLs into the address space of the process:

- ntdll.dll OLEAUT32.dll ole32.dll SHLWAPI.dll MPR.dll SHELL32.dll msvcrt.dll
- KERNEL32.dll USER32.dll ADVAPI32.dll NETAPI32.dll ACTIVEDS.dll

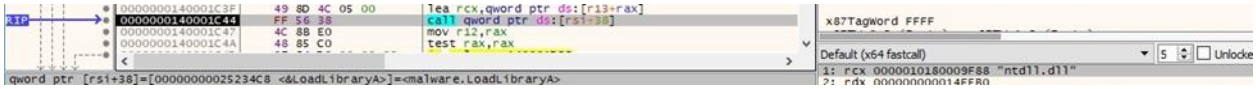


Figure 5

GetProcAddress is used to obtain the address of multiple export functions such as "RtlGetNativeSystemInformation":

Figure 6

The malicious binary changes the protection of a memory area via a function call to VirtualProtect (0x20 = **PAGE_EXECUTE_READ**):

Figure 7

The execution flow is transferred to the DLL file decrypted above:

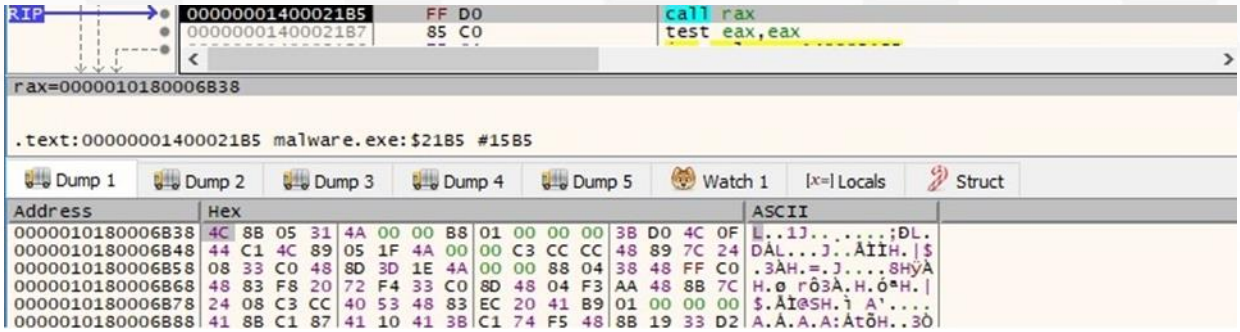


Figure 8

PE-sieve is utilized to dump the DLL file from the current process, as shown in the figure below:

```
C:\Users\   \Desktop>pe-sieve64.exe /pid 4896
PID: 4896
Output filter: no filter: dump everything (default)
Dump mode: autodetect (default)
[*] Using raw process!
[*] Scanning: C:\Users\   \Desktop\malware.exe
[*] Scanning: C:\Windows\System32\ntdll.dll
[*] Scanning: C:\Windows\System32\kernel32.dll
[*] Scanning: C:\Windows\System32\KERNELBASE.dll
[*] Scanning: C:\Windows\System32\oleaut32.dll
[*] Scanning: C:\Windows\System32\msvcp_win.dll
[*] Scanning: C:\Windows\System32\ucrtbase.dll
[*] Scanning: C:\Windows\System32\combase.dll
[*] Scanning: C:\Windows\System32\rpcrt4.dll
[*] Scanning: C:\Windows\System32\bcryptPrimitives.dll
[*] Scanning: C:\Windows\System32\ole32.dll
[*] Scanning: C:\Windows\System32\gdi32.dll
[*] Scanning: C:\Windows\System32\gdi32full.dll
[*] Scanning: C:\Windows\System32\user32.dll
[*] Scanning: C:\Windows\System32\win32u.dll
[*] Scanning: C:\Windows\System32\sechost.dll
[*] Scanning: C:\Windows\System32\imm32.dll
[*] Scanning: C:\Windows\System32\shlwapi.dll
[*] Scanning: C:\Windows\System32\msvcrt.dll
[*] Scanning: C:\Windows\System32\mpr.dll
[*] Scanning: C:\Windows\System32\shell32.dll
[*] Scanning: C:\Windows\System32\cfgmgr32.dll
[*] Scanning: C:\Windows\System32\SHCore.dll
[*] Scanning: C:\Windows\System32\windows.storage.dll
[*] Scanning: C:\Windows\System32\advapi32.dll
[*] Scanning: C:\Windows\System32\kernel.appcore.dll
[*] Scanning: C:\Windows\System32\powrprof.dll
[*] Scanning: C:\Windows\System32\profapi.dll
[*] Scanning: C:\Windows\System32\netapi32.dll
[*] Scanning: C:\Windows\System32\wkscli.dll
[*] Scanning: C:\Windows\System32\bcrypt.dll
[*] Scanning: C:\Windows\System32\srvcli.dll
[*] Scanning: C:\Windows\System32\netutils.dll
[*] Scanning: C:\Windows\System32\logoncli.dll
[*] Scanning: C:\Windows\System32\activeds.dll
[*] Scanning: C:\Windows\System32\adsldpc.dll
[*] Scanning: C:\Windows\System32\Wldap32.dll
Scanning workingset: 254 memory regions.
[*] Workingset scanned in 15 ms
[+] Report dumped to: process_4896
[*] Dumped module to: C:\Users\   \Desktop\\process_4896\140000000.malware.exe as REALIGNED
[*] Dumped module to: C:\Users\   \Desktop\\process_4896\7ff8b10b0000.ntdll.dll as REALIGNED
[*] Dumped module to: C:\Users\   \Desktop\\process_4896\7ff8ae140000.KERNELBASE.dll as REALIGNED
[!] Virtual section size is out of bounds: 200
[!] Truncated to maximal size: 0, buffer size: 17000
[*] Dumped module to: C:\Users\   \Desktop\\process_4896\10180000000.dll as UNMAPPED
[+] Dumped modified to: process_4896
[+] Report dumped to: process_4896
```

Figure 9

The DLL file has two export functions called “RunW” and “runW,” which execute the same code.

The ransomware extracts the command-line string for the process using GetCommandLineW:

```
.c:0000010180005E4C ; Exported entry   1. RunW
.c:0000010180005E4C ; Exported entry   2. runW
.c:0000010180005E4C
.c:0000010180005E4C
.c:0000010180005E4C ; Attributes: noreturn
.c:0000010180005E4C
.c:0000010180005E4C public runW
.c:0000010180005E4C runW proc near
.c:0000010180005E4C sub     rsp, 28h        ; RunW
.c:0000010180005E50 call    cs:GetCommandLineW
.c:0000010180005E56 mov     rcx, rax
.c:0000010180005E59 call    sub_10180005818
```

Figure 10

The CommandLineToArgvW routine is used to parse the command line string and to return pointers to the command line arguments, as displayed in figure 11.

Figure 11

The malware retrieves the path of the current executable via a function call to GetModuleFileNameW:

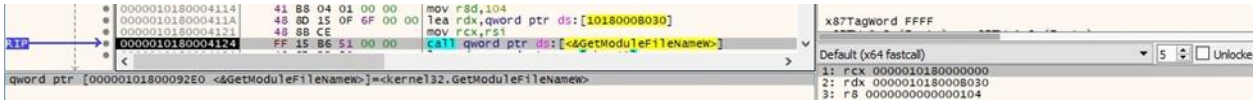


Figure 12

The binary decrypts a list of arguments and compares them with the values extracted above:

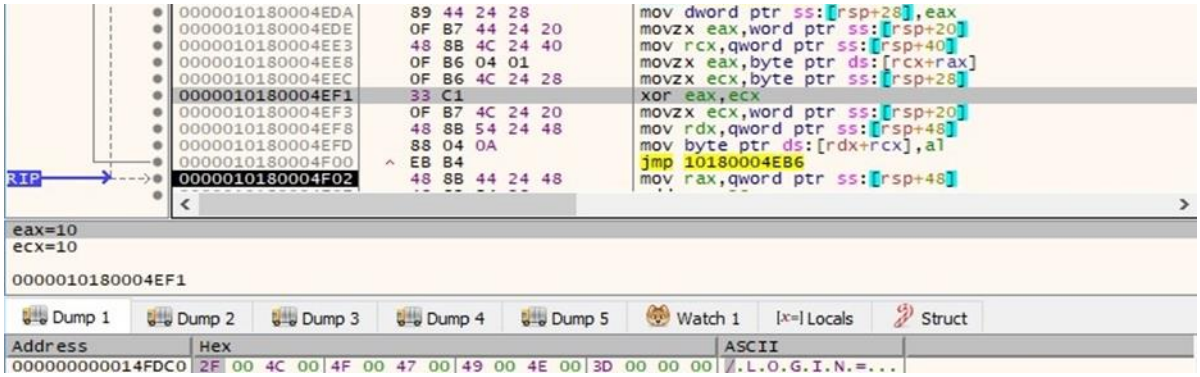


Figure 13

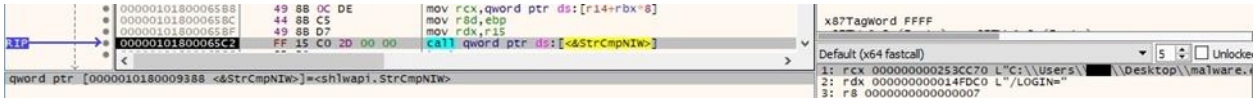


Figure 14

We explain each command-line parameter in the table below.

Parameter	Explanation
/LOGIN=	Username used to propagate to other machines
/PASSWORD=	Password used to propagate to other machines
/CONSOLE	Logging using the Windows console
/NODEL	Do not delete itself
/NOKILL	Do not stop the targeted processes and services
/NOLOG	No difference in execution
/SHAREALL	Encrypt all shared resources excepting "\ADMIN$"
/NETWORK	-w Use WMI to move laterally -s Create a remote service to run the ransomware
/PARAMS=	Parameters that the malware run with when performing lateral movement
/TARGET=	Encrypt a specific file/directory
/FAST=	Size for fast encryption (default value = 0x10000000 bytes)
/MIN=	Minimum size of a file to be encrypted
/MAX=	Maximum size of a file to be encrypted
/FULLPD	Do not avoid to encrypt the "Program Files", "Program Files (x86)", and "ProgramData" folders
/MARKER=	Create a marker file in a drive to be encrypted
/NOLOCK=	-L Do not encrypt local drives -N Do not target other computers in the network -S Do not encrypt network shared resources

The ransomware initializes the COM library for use by the current thread (0x0 = **COINIT_MULTITHREADED**):

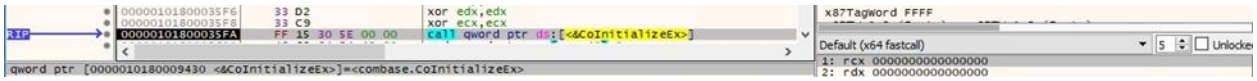


Figure 15

The process registers security and sets default security values using the CoInitializeSecurity API (0x3 = **RPC_C_IMP_LEVEL_IMPERSONATE**):

Figure 16

IsUserAnAdmin is utilized to verify whether the current user is a member of the local Administrators group:

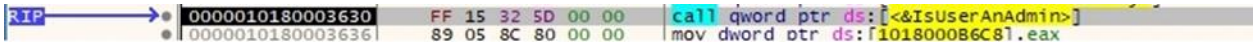


Figure 17

The malware creates a logging file called ".log" in the current directory, which will be populated with different information about the local machine and its actions (0xC0000000 =

GENERIC_READ | **GENERIC_WRITE**, 0x3 = **FILE_SHARE_READ** | **FILE_SHARE_WRITE**):

Figure 18

The Quantum ransomware's version 5.1 and a custom "system information header" are written to the logging file using the WriteFile function:

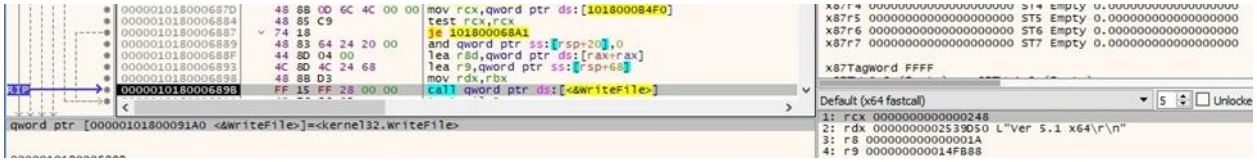


Figure 19

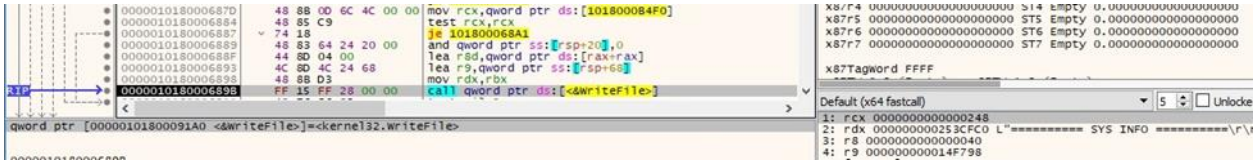


Figure 20

The malicious executable obtains information about the current system by calling the GetSystemInfo routine:

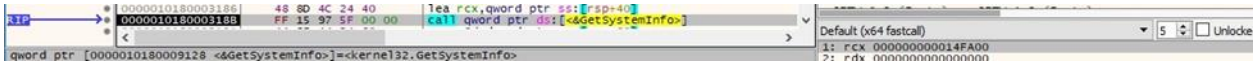


Figure 21

GlobalMemoryStatus is used to retrieve information about the system's usage of physical and virtual memory (see figure 22).

Figure 22

Quantum ransomware extracts the operating system version via a call to RtlGetVersion:

Figure 23

The system processor's architecture is extracted by calling the RtlGetNativeSystemInformation function (0x1 = **SystemProcessorInformation**):

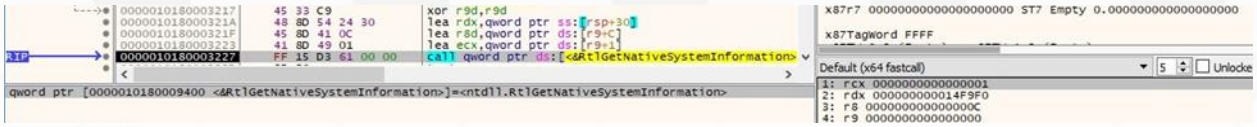


Figure 24

The process retrieves the username associated with the current thread and the NetBIOS name of the local computer:

Figure 25

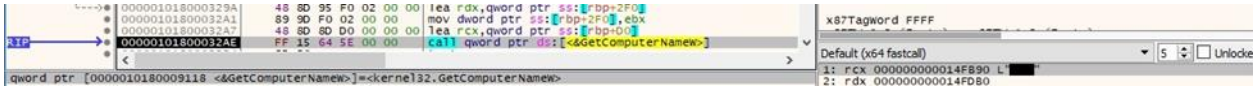


Figure 26

The executable obtains join status information for the local computer using the NetGetJoinInformation API:

Figure 27

The OpenProcessToken routine is used to open the access token associated with the current process (0x8 = **TOKEN_QUERY**):

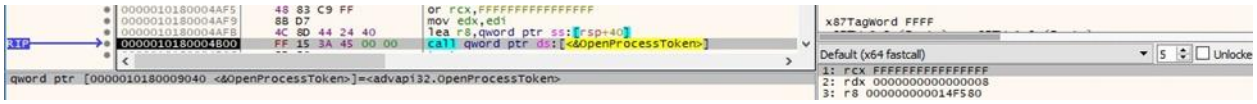


Figure 28

The ransomware extracts a TOKEN_GROUPS structure containing the group accounts associated with the above token (0x2 = **TokenGroups**):

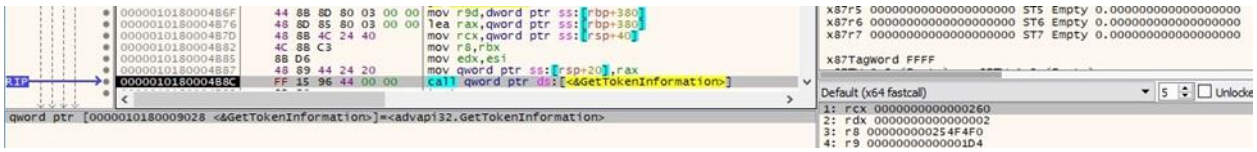


Figure 29

The LookupAccountSidW API is utilized to obtain the name of the group corresponding to a security identifier (SID) passed as a parameter:

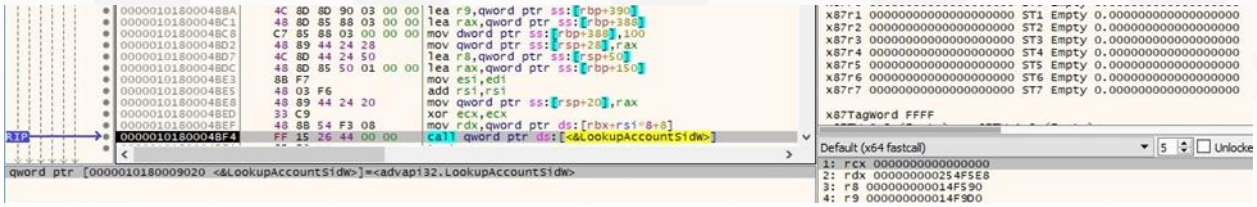


Figure 30

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

Figure 31

LookupPrivilegeValueA is used to extract the locally unique identifier (LUID) corresponding to the “SeRestorePrivilege” and “SeDebugPrivilege” privileges:

Figure 32

The binary enables the above privileges in the access token via a function call to AdjustTokenPrivileges:

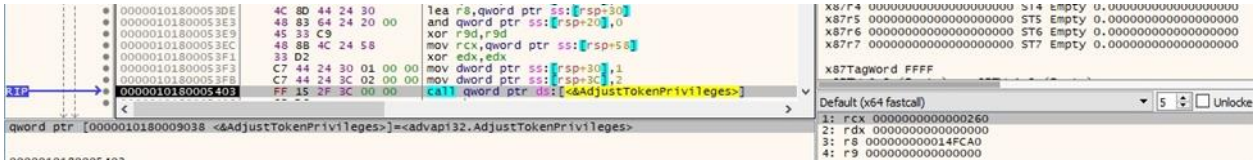


Figure 33

The cpuid instruction returns processor information that is stored in the EAX, EBX, ECX, and EDX registers:

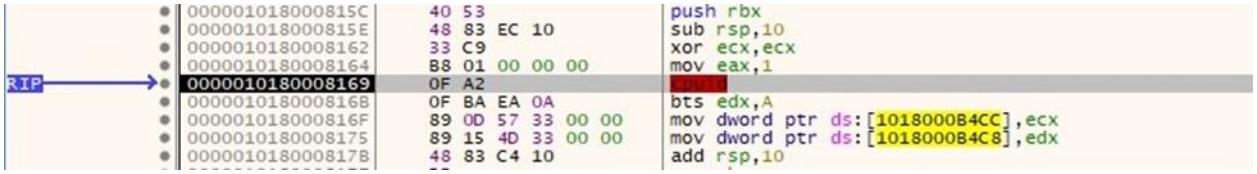


Figure 34

Quantum ransomware generates 32 random bytes using the rdtsc instruction, which reads the current value of the processor’s time stamp. This operation is performed ten times, and the resulting buffer represents the global ChaCha20 key that will be used later on:

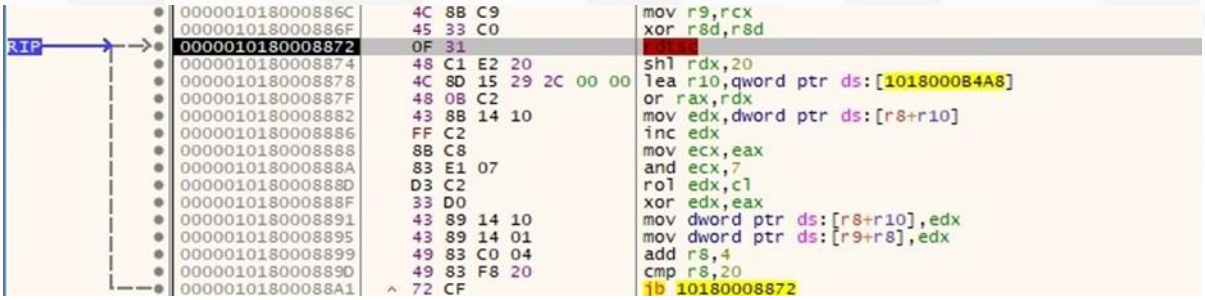


Figure 35

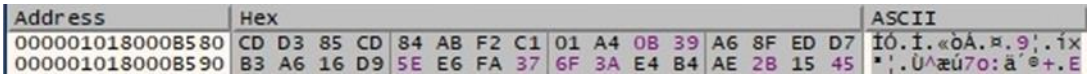


Figure 36

CryptAcquireContextW is utilized to acquire a handle to a key container within a cryptographic service provider (0x1 = **PROV_RSA_FULL**):

Figure 37

The ransomware imports a public RSA key via a call to CryptImportKey:

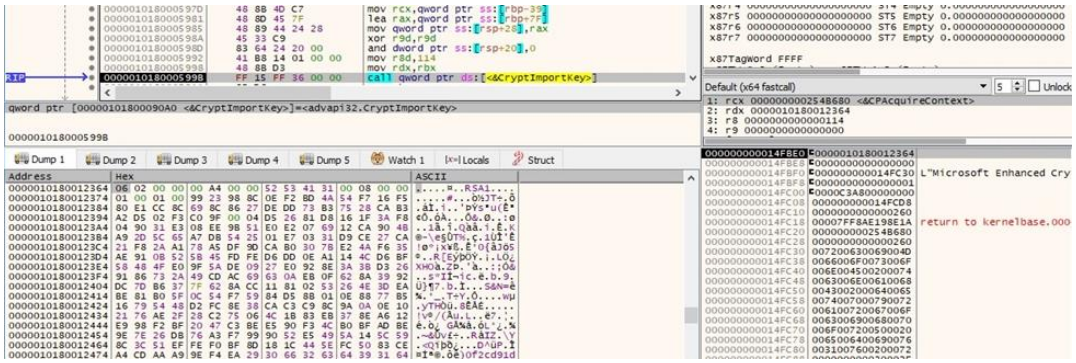
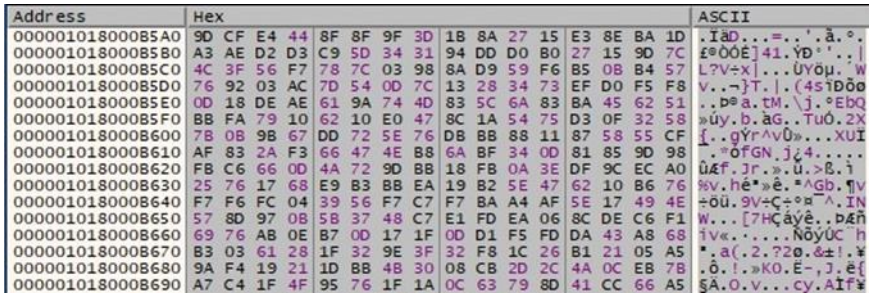


Figure 38

The public RSA key is used to encrypt the global ChaCha20 key generated before:

Figure 39



Address	Hex	ASCII
00000101800085A0	9D CF E4 44 8F 8F 9F 3D 1B 8A 27 15 E3 8E BA 1D	.ÏäD...=..'.ã.°.
00000101800085B0	A3 AE D2 D3 C9 5D 34 31 94 DD D0 B0 27 15 9D 7C	£®ÒÓÉ]41.ÝÐ°'..|
00000101800085C0	4C 3F 56 F7 78 7C 03 98 8A D9 59 F6 B5 0B B4 57	L?V÷x|...ŠÙYöµ.´W
00000101800085D0	76 92 03 AC 7D 54 0D 7C 13 28 34 73 EF D0 F5 F8	v..¬}T.|.(4sïÐõø
00000101800085E0	0D 18 DE AE 61 9A 74 4D 83 5C 6A 83 BA 45 62 51	..Þ®a.tM.\j.ºEbQ
00000101800085F0	BB FA 79 10 62 10 E0 47 8C 1A 54 75 D3 0F 32 58	»úy.b.àG..TuÓ.2X
0000010180008600	7B 0B 9B 67 DD 72 5E 76 DB BB 88 11 87 58 55 CF	{..gÝr^vÛ»...XUÏ
0000010180008610	AF 83 2A F3 66 47 4E B8 6A BF 34 0D 81 85 9D 98	¯.*ófGN¸j¿4.....
0000010180008620	FB C6 66 0D 4A 72 9D BB 18 FB 0A 3E DF 9C EC A0	ûÆf.Jr.».û.>ß.ì
0000010180008630	25 76 17 68 E9 B3 BB EA 19 B2 5E 47 62 10 B6 76	%v.hé³»ê.²^Gb.¶v
0000010180008640	F7 F6 FC 04 39 56 F7 C7 F7 BA A4 AF 5E 17 49 4E	÷öü.9V÷Ç÷º¤¯^.IN
0000010180008650	57 8D 97 0B 5B 37 48 C7 E1 FD EA 06 8C DE C6 F1	W...[7HÇáýê..ÞÆñ
0000010180008660	69 76 AB 0E B7 0D 17 1F 0D D1 F5 FD DA 43 A8 68	iv«.·.....ÑõýÚC¨h
0000010180008670	B3 03 61 28 1F 32 9E 3F 32 F8 1C 26 B1 21 05 A5	³.a(.2.?2ø.&±!.¥
0000010180008680	9A F4 19 21 1D BB 4B 30 08 CB 2D 2C 4A 0C E8 7B	.ô.!.»K0.Ë-,J.è{
0000010180008690	A7 C4 1F 4F 95 76 1F 1A 0C 63 79 8D 41 CC 66 A5	§Ä.O.v...cy.AÌf¥

Figure 40

The process obtains the computer name and then encrypts it using the XOR operator, as highlighted in the figure below:

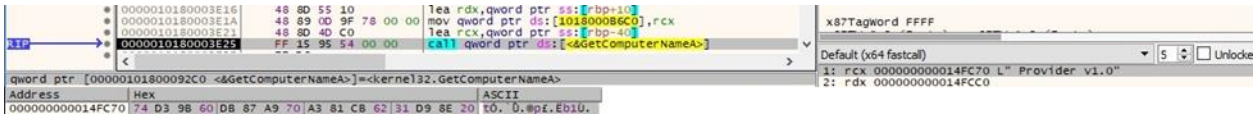


Figure 41

A hard-coded 16-byte buffer used to encrypt the computer name together with the encrypted result represent the client ID that is written to the ransom note:

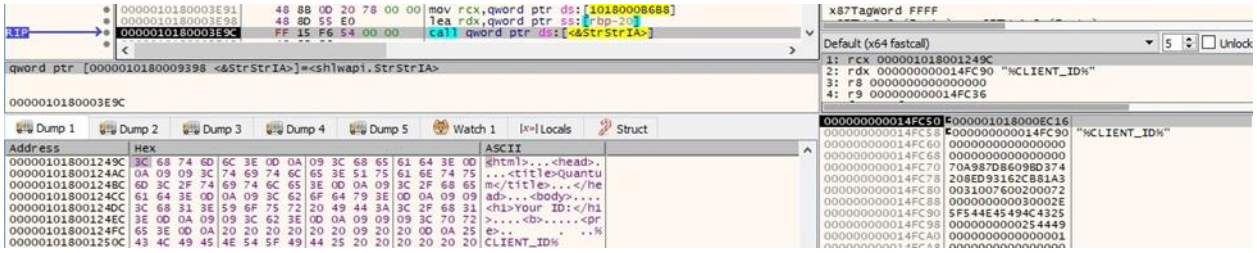
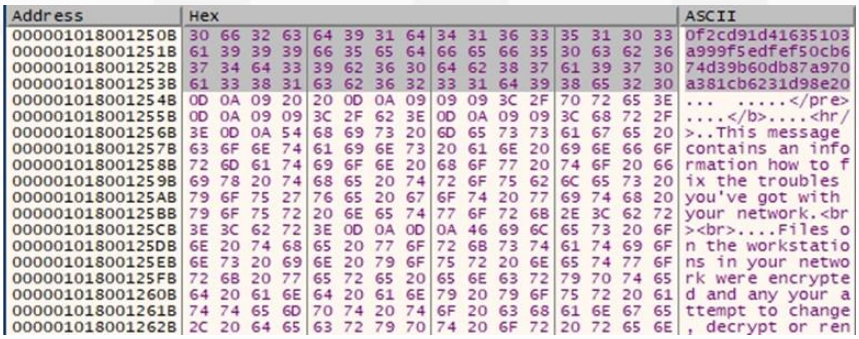


Figure 42



Address	Hex	ASCII
000001018001250B	30 66 32 63 64 39 31 64 34 31 36 33 35 31 30 33	0f2cd91d41635103
000001018001251B	61 39 39 39 66 35 65 64 66 65 66 35 30 63 62 36	a999f5edfef50cb6
000001018001252B	37 34 64 33 39 62 36 30 64 62 38 37 61 39 37 30	74d39b60db87a970
000001018001253B	61 33 38 31 63 62 36 32 33 31 64 39 38 65 32 30	a381cb6231d98e20
000001018001254B	0D 0A 09 20 20 0D 0A 09 09 09 3C 2F 70 72 65 3E	... .</pre>
000001018001255B	0D 0A 09 09 3C 2F 62 3E 0D 0A 09 09 3C 68 72 2F	........<hr/
000001018001256B	3E 0D 0A 54 68 69 73 20 6D 65 73 73 61 67 65 20	>..This message
000001018001257B	63 6F 6E 74 61 69 6E 73 20 61 6E 20 69 6E 66 6F	contains an info
000001018001258B	72 6D 61 74 69 6F 6E 20 68 6F 77 20 74 6F 20 66	rmation how to f
000001018001259B	69 78 20 74 68 65 20 74 72 6F 75 62 6C 65 73 20	ix the troubles
00000101800125AB	79 6F 75 27 76 65 20 67 6F 74 20 77 69 74 68 20	you've got with
00000101800125BB	79 6F 75 72 20 6E 65 74 77 6F 72 6B 2E 3C 62 72	your network.<br
00000101800125CB	3E 3C 62 72 3E 0D 0A 0D 0A 46 69 6C 65 73 20 6F	> ....Files o
00000101800125DB	6E 20 74 68 65 20 77 6F 72 6B 73 74 61 74 69 6F	n the workstatio
00000101800125EB	6E 73 20 69 6E 20 79 6F 75 72 20 6E 65 74 77 6F	ns in your netwo
00000101800125FB	72 6B 20 77 65 72 65 20 65 6E 63 72 79 70 74 65	rk were encrypte
000001018001260B	64 20 61 6E 64 20 61 6E 79 20 79 6F 75 72 20 61	d and any your a
000001018001261B	74 74 65 6D 70 74 20 74 6F 20 63 68 61 6E 67 65	ttempt to change
000001018001262B	2C 20 64 65 63 72 79 70 74 20 6F 72 20 72 65 6E	, decrypt or ren

Figure 43

The binary creates a registry key called “Software\Classes\.quantum\shell\Open\command” and its default value is set to a process that displays the ransom note:

Figure 44

Figure 45

Quantum ransomware decrypts a list of files/folders that will not be encrypted:

- ":\Windows\" ":\System Volume Information\" ":\$RECYCLE.BIN\" ":\SYSTEM.SAV" ":\WINNT" ":\$WINDOWS.~BT\"
- ":\Windows.old\" ":\PerfLog\" ":\PerfLogs\" ":\Program Files\" ":\Program Files (x86)\" ":\Boot"
- ":\ProgramData\Microsoft\" ":\ProgramData\Packages\" ":\EFI" ":\ProgramData" "$\Windows\" "$\System Volume Information\"
- "$\$RECYCLE.BIN\" "$\SYSTEM.SAV" "$\WINNT" "$\$WINDOWS.~BT\" "$\Windows.old\" "$\PerfLog\" "$\PerfLogs\" "$\Program Files\"
- "$\Program Files (x86)\" "$\Boot" "$\ProgramData\Microsoft\" "$\ProgramData\Packages\" "$\EFI" "$\ProgramData" "\WindowsApps\"
- "\Microsoft\Windows\" "\Local\Packages\" "\Windows Defender" "\microsoft shared\" "\Google\Chrome\" "\Mozilla Firefox\"
- "\Mozilla\Firefox\" "\Internet Explorer\" "\MicrosoftEdge\" "\Tor Browser\" "\AppData\Local\Temp\" "\AppData" "\All Users"

- "\Boot" "\Google" "\Mozilla" "\autorun.inf" "\boot.ini" "\bootfont.bin" "\bootsect.bak" "\bootmgr" "\bootmgr.efi" "\bootmgfw.efi"
- "\iconcache.db" "\desktop.ini" "\ntldr" "\ntuser.dat" "\ntuser.dat.log" "\ntuser.ini" "\thumbs.db"

It also decrypts a list of extensions that will be avoided:

- "exe" "dll" "sys" "msi" "mui" "inf" "cat" "bat" "cmd" "ps1" "vbs" "ttf" "fon" "lnk"
- ".386" ".adv" ".ani" ".bin" ".cab" ".com" ".cpl" ".cur" ".deskthemepack" ".diagcab"
- ".diagcfg" ".diagpkg" ".drv" ".hlp" ".icl" ".icns" ".ico" ".ics" ".idx" ".ldf" ".mod"
- ".mpa" ".mp4" ".mp3" ".msc" ".msp" ".msstyles" ".msu" ".nls" ".nomedia" ".ocx" ".prf"
- ".rom" ".rtp" ".scr" ".shs" ".spl" ".theme" ".themepack" ".wpx" ".lock" ".key" ".hta"

Stopping targeted services

The OpenSCManagerA API is utilized to open the service control manager database (0xF003F = **SC_MANAGER_ALL_ACCESS**):

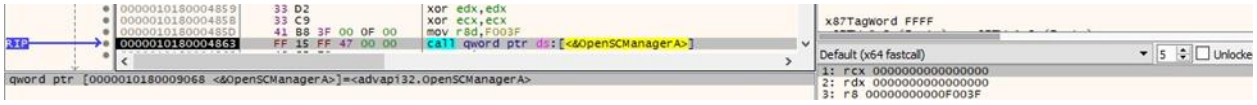


Figure 46

The malware extracts a list of active services using EnumServicesStatusA (0x30 = **SERVICE_WIN32**, 0x1 = **SERVICE_ACTIVE**):

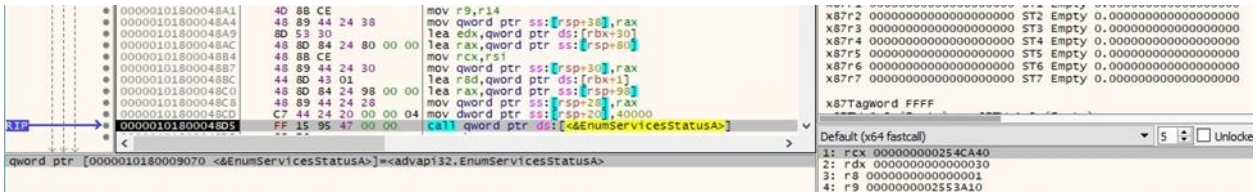


Figure 47

The services whose name contains "SQL", "database", and "msexchange" are targeted by the ransomware:

Figure 48

The executable opens a targeted service by calling the OpenServiceA routine (0x20 = **SERVICE_STOP**):

Figure 49

The service is stopped using ControlService (0x1 = **SERVICE_CONTROL_STOP**):

Figure 50

The QueryServiceStatusEx function is used to verify whether the service was successfully stopped:

Figure 51

Killing targeted processes

The binary retrieves a list of processes by calling the RtlGetNativeSystemInformation native API (0x5 = **SystemProcessInformation**):

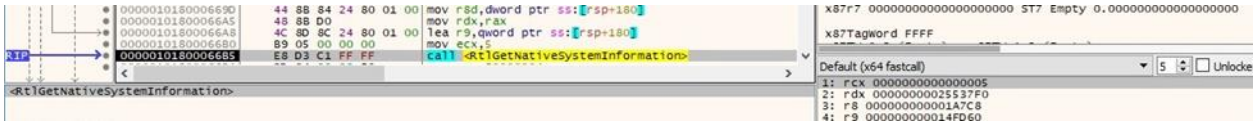


Figure 52

The ransomware constructs a list of processes to terminate:

- "msftesql.exe" "sqlagent.exe" "sqlbrowser.exe" "sqlwriter.exe" "oracle.exe" "ocssd.exe" "dbsnmp.exe"
- "synctime.exe" "agntsvc.exe" "isqlplussvc.exe" "xfssvccon.exe" "sqlservr.exe" "encsvc.exe" "ocautoupds.exe"
- "mydesktopservice.exe" "firefoxconfig.exe" "tbirdconfig.exe" "mydesktopqos.exe" "ocomm.exe" "mysqld.exe"
- "mysqld-nt.exe" "mysqld-opt.exe" "dbeng50.exe" "sqbcoreservice.exe" "excel.exe" "infopath.exe" "msaccess.exe"
- "mspub.exe" "onenote.exe" "outlook.exe" "powerpnt.exe" "sqlservr.exe" "thebat.exe" "steam.exe" "thebat64.exe"
- "thunderbird.exe" "visio.exe" "winword.exe" "wordpad.exe" "QBW32.exe" "QBW64.exe" "ipython.exe" "wpython.exe"
- "python.exe" "dumpcap.exe" "procmon.exe" "procmon64.exe" "procexp.exe" "procexp64.exe"

Figure 53

The malware opens a targeted process using the OpenProcess routine (0x1 = **PROCESS_TERMINATE**):

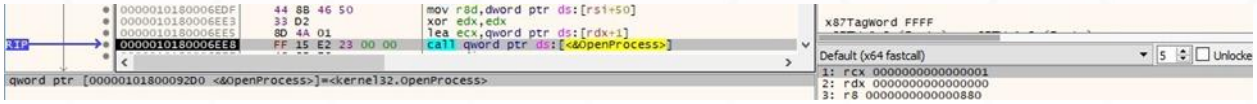


Figure 54

The process is killed by calling the TerminateProcess API:

Figure 55

An example of a log file is displayed in the figure below:

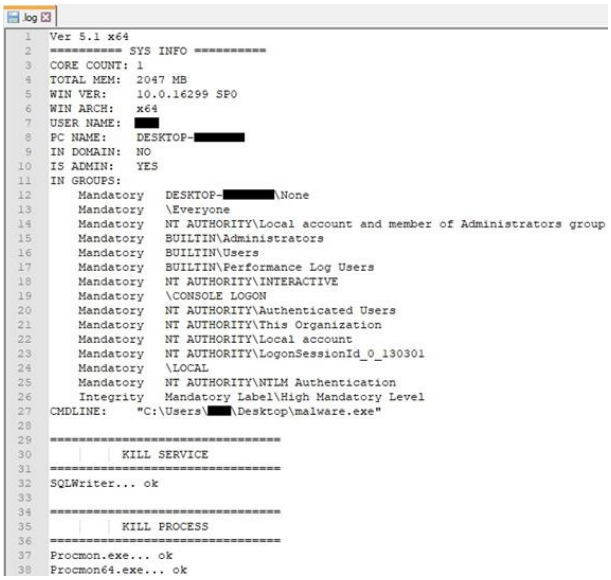


Figure 56

The process calls the GetVolumeInformationW API with the drives ranging from A: to Z: (see figure 57).

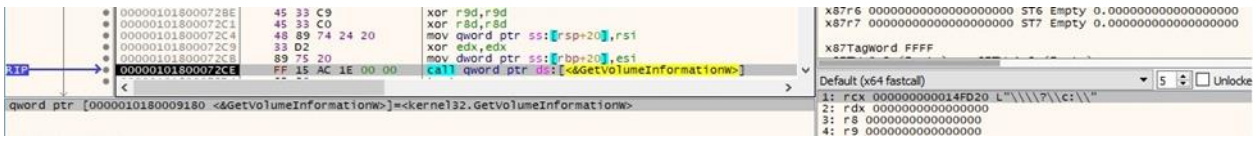


Figure 57

The drive type is retrieved using the GetDriveTypeW function, and the malware expects a value different than 0x4 (**DRIVE_REMOTE**):

Figure 58

The ransomware creates a thread that handles the local drives encryption and another one that handles the network shares encryption. The responsible function is the same, sub_10180008CA0:

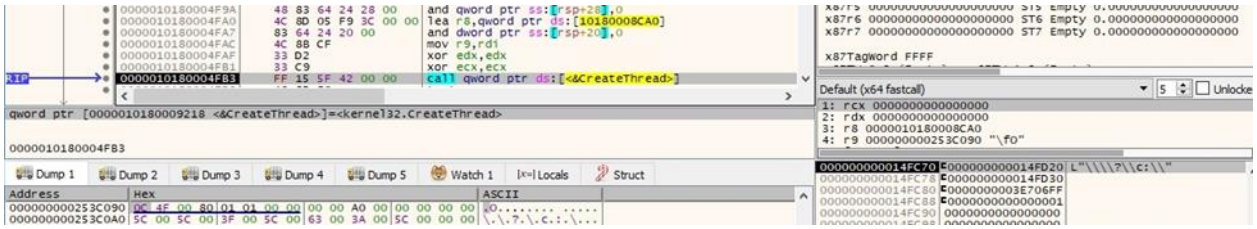


Figure 59

Thread activity – sub_10180008CA0 function

The process creates two unnamed event objects via a function call to CreateEventA:

Figure 60

An unnamed semaphore object is also created by the malware:

Figure 61

The binary creates two threads that will perform the files' encryption. The responsible function is sub_10180005014, and the current thread gives a filename to encrypt to the encryption threads:

Figure 62

Quantum ransomware starts enumerating the network resources by calling the WNetOpenEnumW API (0x2 = **RESOURCE_GLOBALNET**):

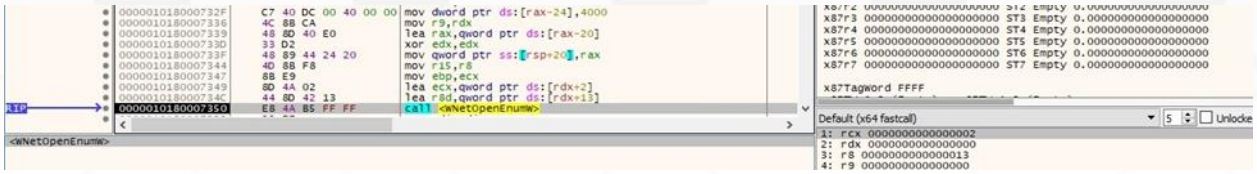


Figure 63

WNetEnumResourceW is utilized to continue the enumeration of network resources:

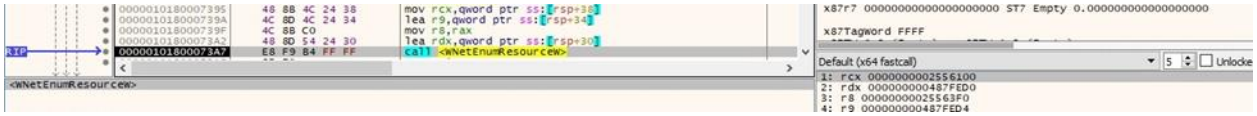


Figure 64

The malicious process retrieves information about the shared resources on the local machine:

Figure 65

The ransomware doesn't target the ADMIN$ share, as highlighted in figure 66.

Figure 66

The files are enumerated using the FindFirstFileW and FindNextFileW functions:

Figure 67

Figure 68

The ransomware doesn't encrypt the ransom note, if present:

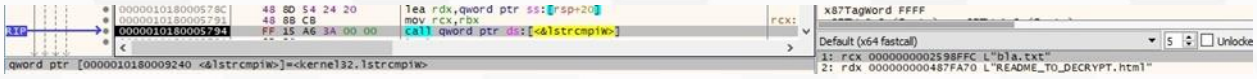


Figure 69

The file's extension is compared with the list that will be avoided, as shown below:

Figure 70

One of the events created earlier is signaled by calling the SetEvent API, which means that a file is ready to be encrypted.

Figure 71

The malware creates a ransom note called "README_TO_DECRYPT.html" in every directory that is encrypted (0x40000000 = **GENERIC_WRITE**):

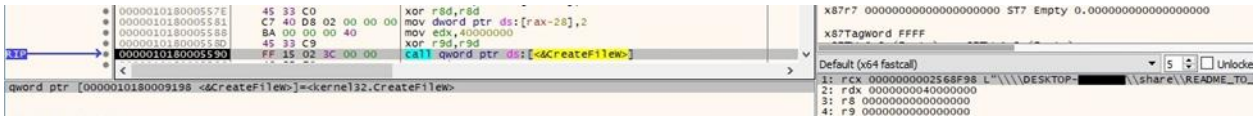


Figure 72

The ransom note is populated using the WriteFile routine:

Figure 73

Thread activity – sub_10180005014 function

ReleaseSemaphore is utilized to release the semaphore created earlier:

Figure 74

The process opens a targeted file by calling the CreateFileW routine (0xC0010000 = **GENERIC_READ** | **GENERIC_WRITE** | **DELETE**):

Figure 75

The size of the file is obtained using GetFileSizeEx:

Figure 76

The SetFileInformationByHandle API is used to append the ".quantum" extension to an encrypted file name (0x3 = **FileRenameInfo**):

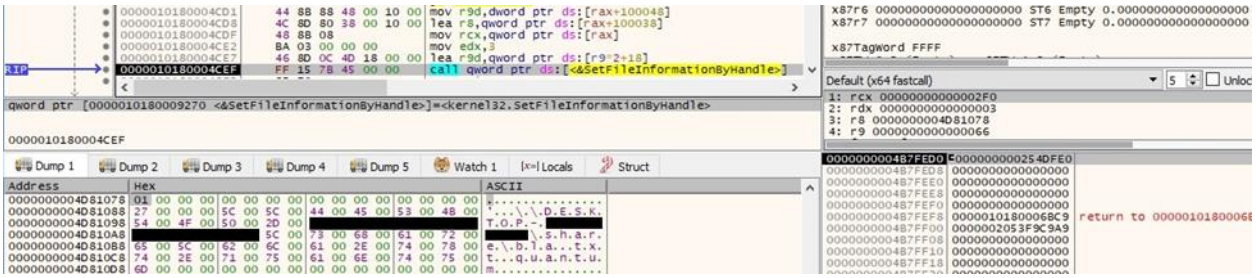


Figure 77

The process generates another ChaCha20 key with the same rdtsc instruction. This key will be used to encrypt the file's content, as we'll see later on:

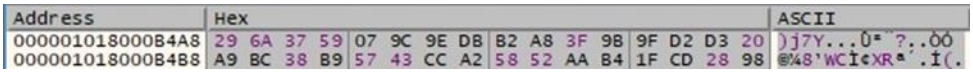


Figure 78

The ransomware constructs the initial ChaCha20 state using the global ChaCha20 key presented in figure 36:

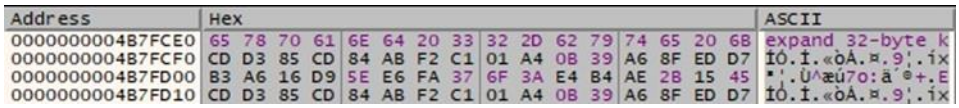


Figure 79

Finally, the ChaCha20 key generated above is encrypted with the global ChaCha20 key and will be stored in the encrypted file:

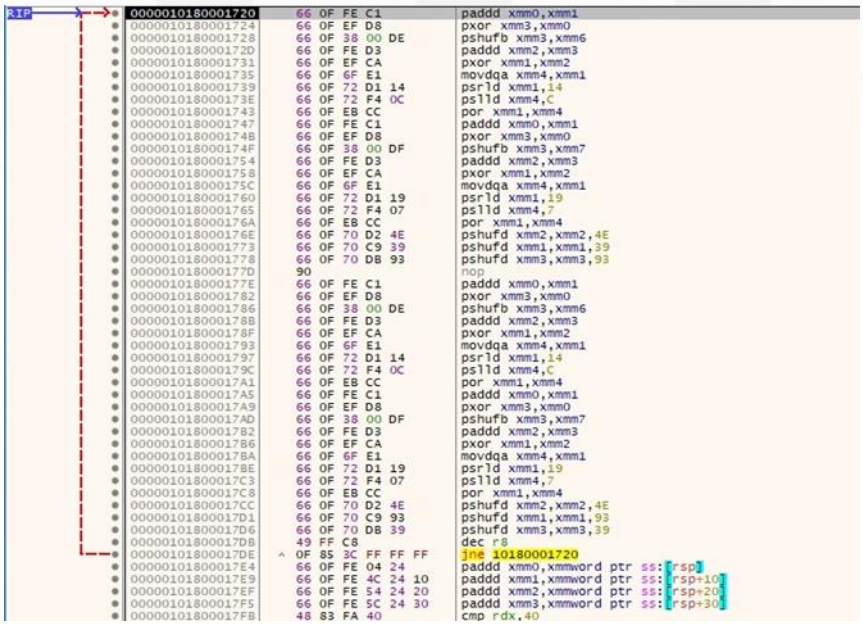
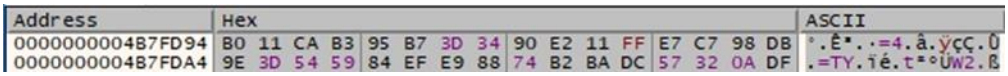


Figure 80



Address	Hex	ASCII
0000000004B7FD94	B0 11 CA B3 95 B7 3D 34 90 E2 11 FF E7 C7 98 DB	°.Ê³.·=4.â.ÿçÇ.Û
0000000004B7FDA4	9E 3D 54 59 84 EF E9 88 74 B2 BA DC 57 32 0A DF	.=TY.ïé.t²ºÜW2.ß

Figure 81

The binary moves the file pointer to the end of the file using SetFilePointerEx (0x2 = **FILE_END**):

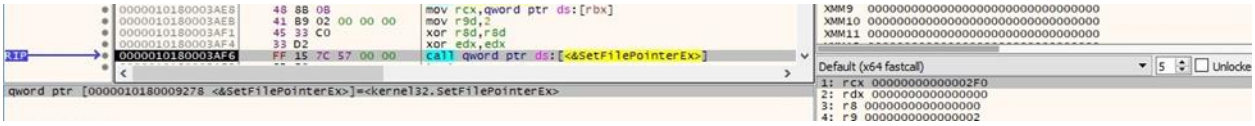


Figure 82

The ransomware writes the fast encryption size (0x10000000 bytes), the encrypted ChaCha20 key, and the RSA-encrypted global ChaCha20 key to the encrypted file:

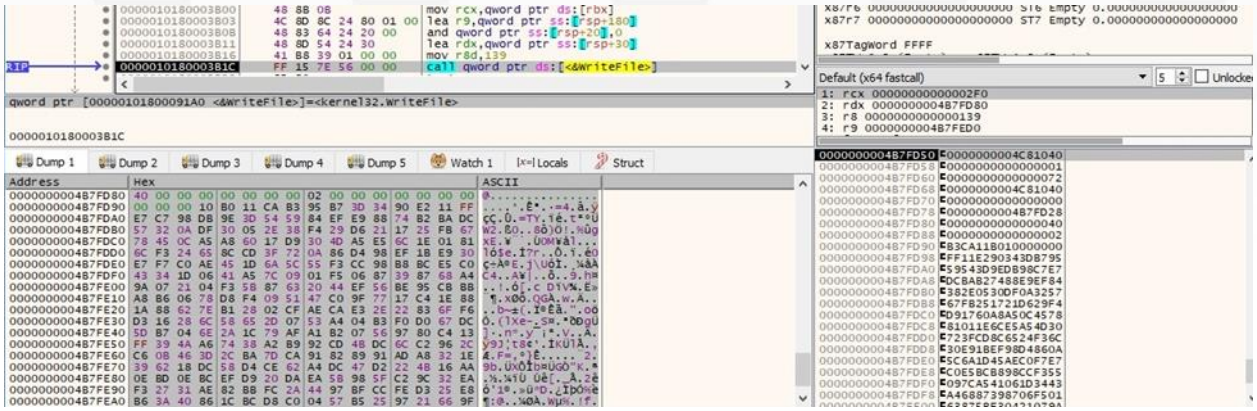


Figure 83

The file content is read by calling the ReadFile API (see figure 84).

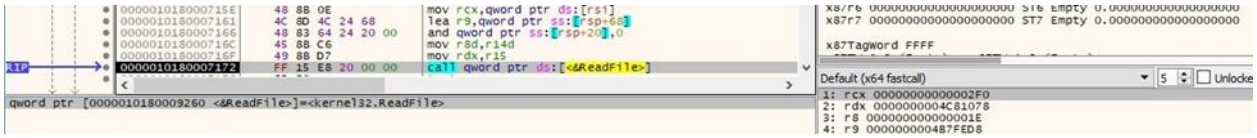


Figure 84

The content is encrypted using the ChaCha20 algorithm, and the encrypted data is written back to the file:

Address	Hex	ASCII
0000000004B7FE00	65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 6B	expand 32-byte k
0000000004B7FE10	29 6A 37 59 07 9C 9E DB B2 A8 3F 9B 9F D2 D3 20	)j7Y...Û² ?..ÒÓ
0000000004B7FE20	A9 BC 38 B9 57 43 CC A2 58 52 AA B4 1F CD 28 98	©¼8¹WCÌ¢XRª´.Í(.
0000000004B7FE30	29 6A 37 59 07 9C 9E DB B2 A8 3F 9B 9F D2 D3 20	)j7Y...Û² ?..ÒÓ

Figure 85

Figure 86

We continue with the analysis of the main thread.

The malware obtains the path of the %TEMP% folder using GetTempPathW:

Figure 87

A batch file is created in the %TEMP% directory. The file name is based on a GetTickCount function return value:

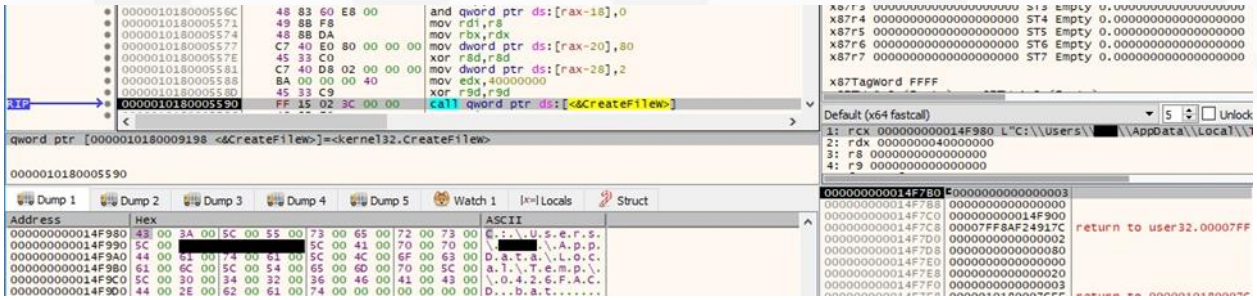


Figure 88

The script's purpose is to delete itself and the initial executable, as highlighted below:

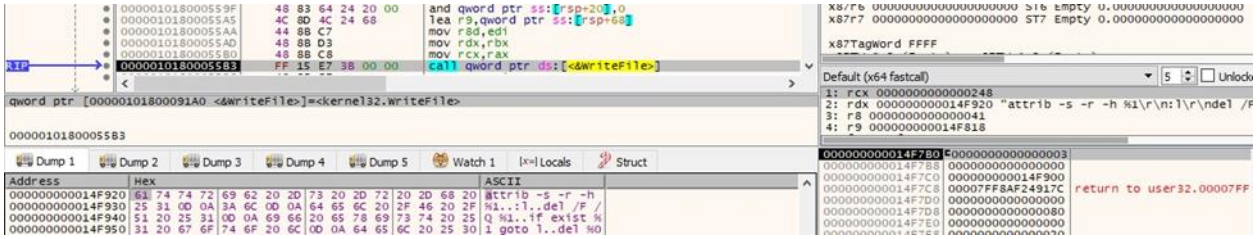


Figure 89

Finally, the batch file is executed by calling the CreateProcessW function (0x08000000 = **CREATE_NO_WINDOW**):

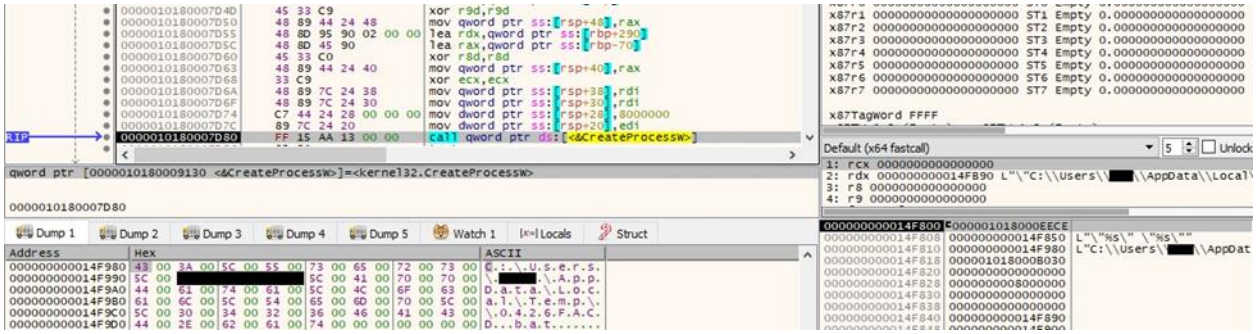


Figure 90

Running with the /LOGIN= /PASSWORD= /NETWORK-w (-s) /PARAMS= /CONSOLE parameters

Quantum ransomware creates a new console using the AllocConsole routine:

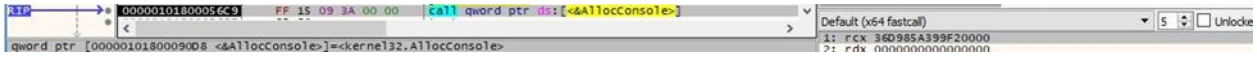


Figure 91

The executable retrieves a handle to the standard output device using GetStdHandle (0xFFFFFFF5 = **STD_OUTPUT_HANDLE**):

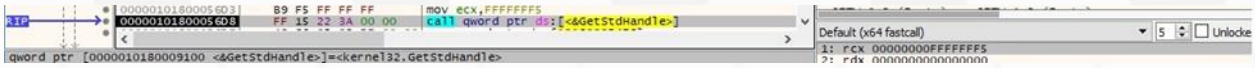


Figure 92

The process creates 8 threads that will enumerate the computers in the Windows domain as well as the local network:

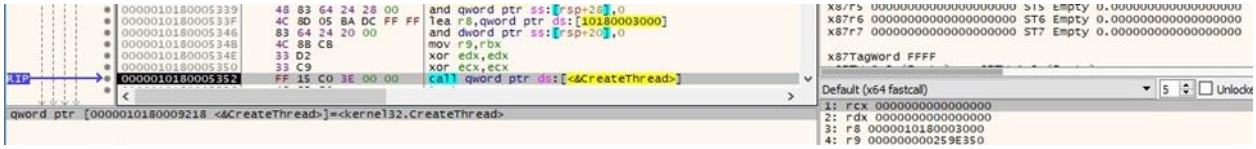


Figure 93

NetGetDCName is utilized to retrieve the name of the domain controller for the primary domain:

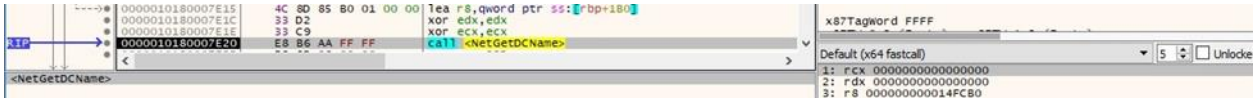


Figure 94

The WNetOpenEnumW function is used to start enumerating the resources on the local network (0x2 = **RESOURCE_GLOBALNET**):

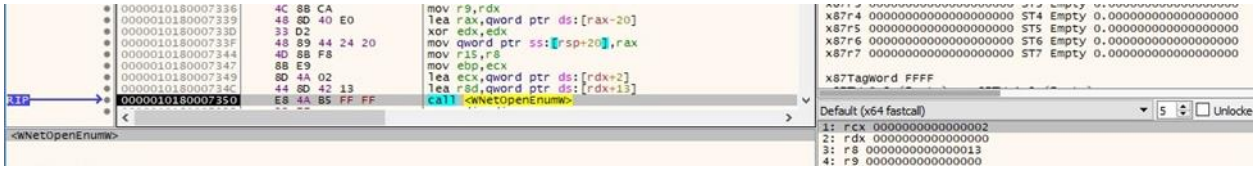


Figure 95

The malicious binary continues the enumeration of the network resources using WNetEnumResourceW:

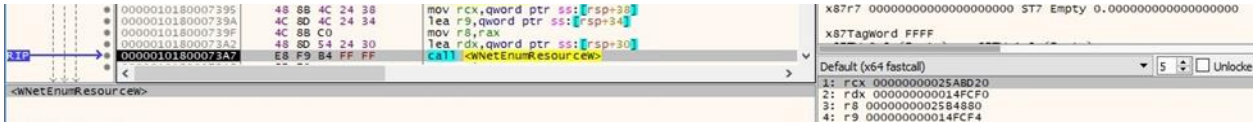


Figure 96

The malware makes a connection to the identified network resources using the username and password passed as parameters:

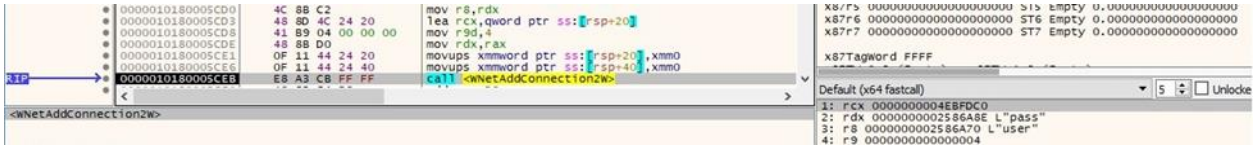


Figure 97

The malware doesn't target the ADMIN$ and IPC$ shares, however; the ransomware executable will be copied in the ProgramData directory found on the remote machine identified above:

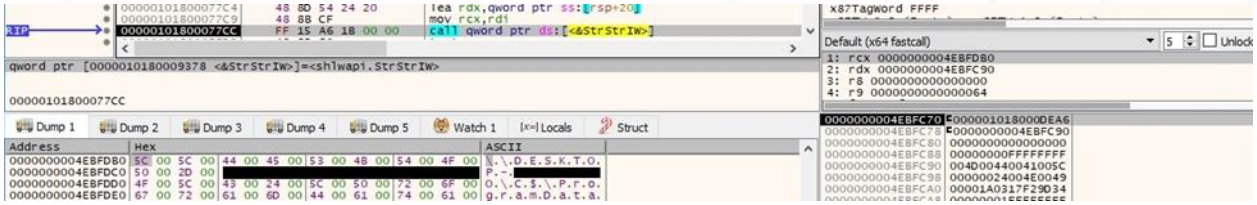


Figure 98

CopyFileW is utilized to copy the binary on the remote host. It will be executed via WMI (if the "-w" parameter is specified) or by creating a remote Windows service (if the "-s" parameter is specified):

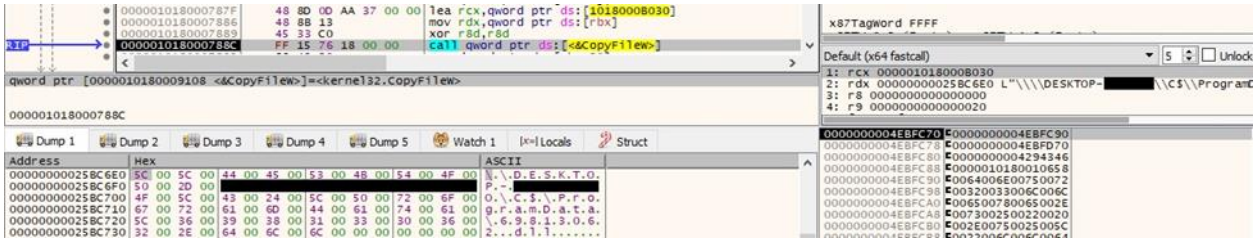


Figure 99

Quantum ransomware obtains information about the shared resources on the remote computer by calling the NetShareEnum API:

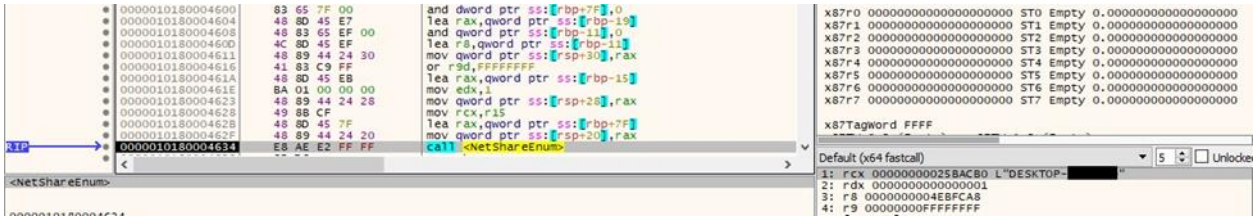


Figure 100

Running with the /NODEL parameter

In this case, the ransomware doesn't perform the self-deletion operation.

Running with the /NOKILL parameter

The malware doesn't stop the targeted processes and services.

Running with the /NOLOG parameter

Interestingly, the ransomware still creates the log file even if it's running with this parameter.

Running with the /SHAREALL parameter

In this case, the malware encrypts all shared resources except for “$ADMIN”.

Running with the /TARGET= parameter

Quantum ransomware only encrypts the file/directory passed as a parameter.

Running with the /FAST= parameter

The size for fast encryption is set to the value passed as the parameter. The last 5 bytes represent a marker that appears in every encrypted file:

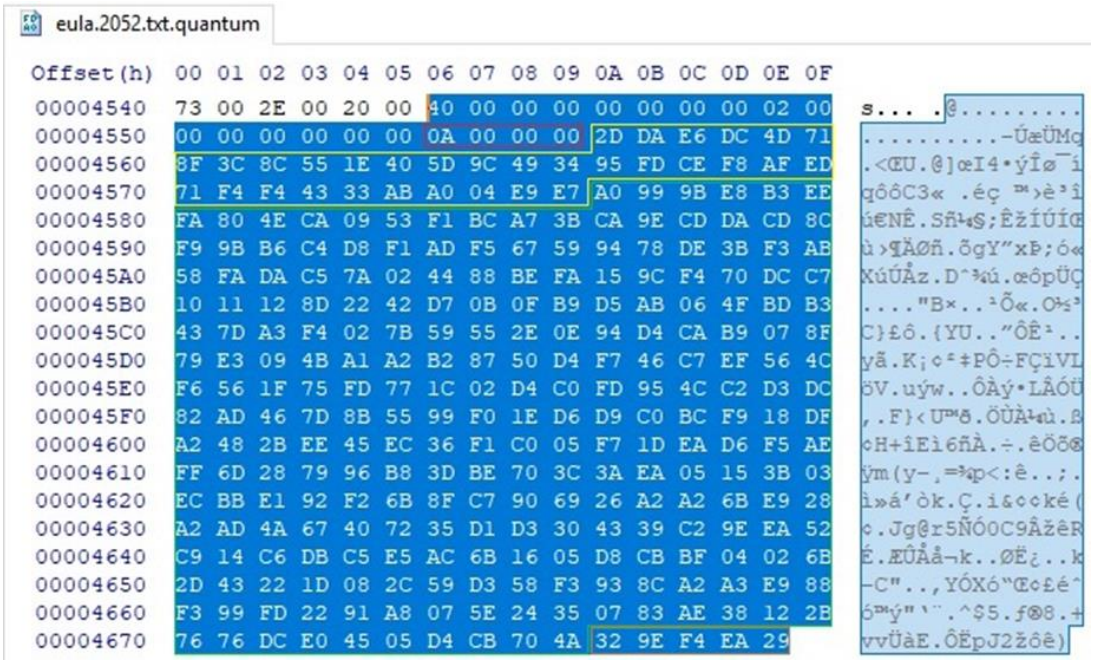


Figure 101

Running with the /MIN= or /MAX= parameter

If the file size is greater than MAX or lower than MIN, the file is not encrypted by Quantum ransomware.

Running with the /FULLPD parameter

The following directories will also be encrypted by the ransomware: “Program Files”, “Program Files (x86)”, and “ProgramData”.

Running with the /MARKER= parameter

The process creates an empty file on each drive that will be encrypted. The file name is passed

as a parameter (0xC0000000 = **GENERIC_READ** | **GENERIC_WRITE**):

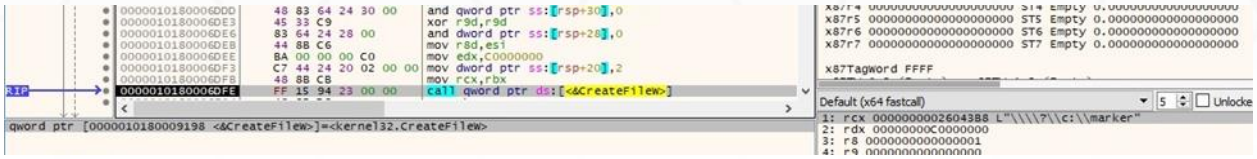


Figure 102

Running with the /NOLOCK=-L, -N, -S parameter

In this case, the malware splits its execution flow according to the parameter:

- do not encrypt local disks (“-L”)
- do not encrypt other machines in the network (“-N”)
- do not encrypt network shared resources (“-S”)

Indicators of Compromise

Quantum Ransom Note

README_TO_DECRYPT.html

Files created

%Temp%\<GetTickCount result>.bat

.log

Registry key created

Software\Classes\.quantum\shell\Open\command