

A Detailed Analysis of the Quantum Ransomware

Prepared by: Vlad Pasca, Senior Malware &
Threat Analyst



[SecurityScorecard.com](https://www.securityscorecard.com)
info@securityscorecard.com

Tower 49
12 E 49th Street
Suite 15-001
New York, NY 10017
[1.800.682.1707](tel:18006821707)

Table of contents

Executive summary	2
Analysis and findings	2
Stopping targeted services	13
Killing targeted processes	14
Thread activity – sub_10180008CA0 function	16
Thread activity – sub_10180005014 function	19
Running with the /LOGIN= /PASSWORD= /NETWORK-w (-s) /PARAMS= /CONSOLE parameters	22
Running with the /NODEL parameter	24
Running with the /NOKILL parameter	24
Running with the /NOLOG parameter	24
Running with the /SHAREALL parameter	25
Running with the /TARGET= parameter	25
Running with the /FAST= parameter	25
Running with the /MIN= or /MAX= parameter	25
Running with the /FULLPD parameter	25
Running with the /MARKER= parameter	26
Running with the /NOLOCK=-L, -N, -S parameter	26
Indicators of Compromise	27

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:

```

0000000140001350 0F B6 C1 movzx eax,cl
0000000140001353 40 2A C6 sub al,si
0000000140001356 24 08 and al,s
0000000140001358 41 32 C0 xor al,r8b
000000014000135B 30 01 xor byte ptr ds:[rcx],al
000000014000135D 48 03 CD add rcx,rbp
0000000140001360 49 3B CA cmp rcx,r10
0000000140001363 72 EB jnb malware.140001350
0000000140001365 48 FF C2 inc rdx
0000000140001368 48 FF C7 inc rdi
000000014000136B 49 FF CB dec r11
000000014000136E 75 D0 jne malware.140001340
0000000140001370 48 8B 5C 24 30 mov rbx,qword ptr ss:[rsp+30]
    
```

Address	Hex	ASCII
000000014002EDD0	01 66 D9 3F 3D 3C 49 3A 42 34 37 43 CA CA 33 41	fU?=<I;B47CÉE3A
000000014002EDE0	F6 3F 3A 48 49 3D 4D 48 79 45 44 35 38 32 32 34	07:KI=MkyED58224
000000014002EDF0	38 4C 3E 4A 49 4C 38 3C 39 45 44 34 41 37 36 34	!L>JL!;<9EDAA764
000000014002EE00	49 3A 4E 3C 3F 48 3D 3D 33 41 46 37 32 42 41 35	I:N<?K==3AF72BAS
000000014002EE10	43 54 88 43 4C 89 39 F7 13 8C 32 08 FB 63 15 2C	CT.L.9+.2.üc.,
000000014002EE20	52 4F 11 3D 3E 53 2E 4D 57 59 61 51 27 5A 59 2C	RO.=>S.MWYaq'ZY,
000000014002EE30	49 1D 59 2C 6E 4D 4F 25 61 5C 2B 63 7D 0A 17 15	I.Y,rM0%a\+c}...
000000014002EE40	5D 55 5E 59 15 41 33 40 65 44 33 34 39 45 44 34]U^Y.A3#eD349ED4

Figure 1

Address	Hex	ASCII
000000014002EDD0	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....yy..
000000014002EDE0	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....
000000014002EDF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000014002EE00	00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 0001.....
000000014002EE10	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..*...I!.LI!Th
000000014002EE20	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
000000014002EE30	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
000000014002EE40	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....\$......
000000014002EE50	C7 2A 54 0C 83 48 3A 5F 83 48 3A 5F 83 48 3A 5F	C*T..K:..K:..K:..
000000014002EE60	97 20 3C 5E 82 48 3A 5F A4 8D 41 5F 81 48 3A 5F	. <^..K:..K:..K:..
000000014002EE70	97 20 38 5E 95 48 3A 5F 83 48 38 5F 03 48 3A 5F	. ;^..K:..K:..K:..
000000014002EE80	7F 3C 83 5F 80 48 3A 5F 1D EB FD 5F 82 48 3A 5F	.<..K:..ey..K:..
000000014002EE90	3D 3A 3F 5E 82 48 3A 5F 83 48 3A 5F 82 48 3A 5F	=: ?^..K:..K:..K:..
000000014002EEA0	18 39 3E 5E 9C 48 3A 5F 1B 39 3A 5E 82 48 3A 5F	.9>^..K:..9:..K:..
000000014002EEB0	1B 39 38 5E 82 48 3A 5F 52 69 63 68 83 48 3A 5F	.98^..K:..Rich..K:..
000000014002EEC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000014002EED0	50 45 00 00 64 86 08 00 A1 87 83 61 00 00 00 00	PE..d...i..*a.....
000000014002EEE0	00 00 00 00 F0 00 22 20 0B 02 0E 1C 00 78 00 000.".....x..
000000014002EEF0	00 BA 00 00 00 00 00 00 38 68 00 00 00 10 00 008k.....

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:

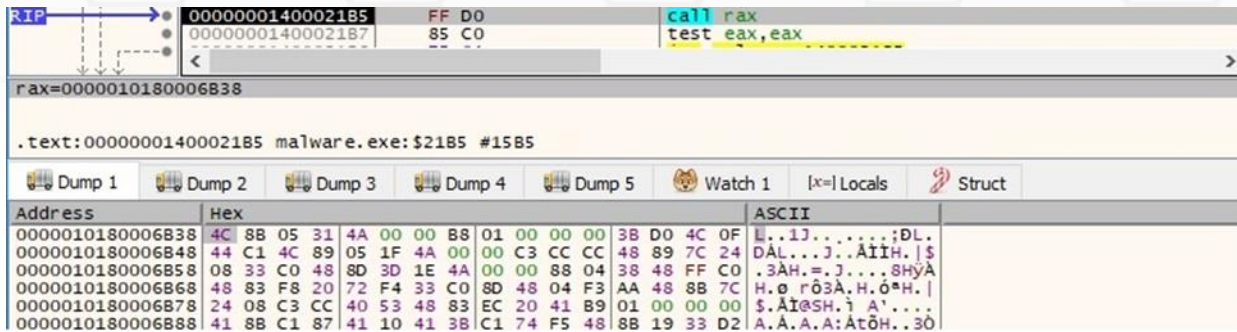


Figure 8

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

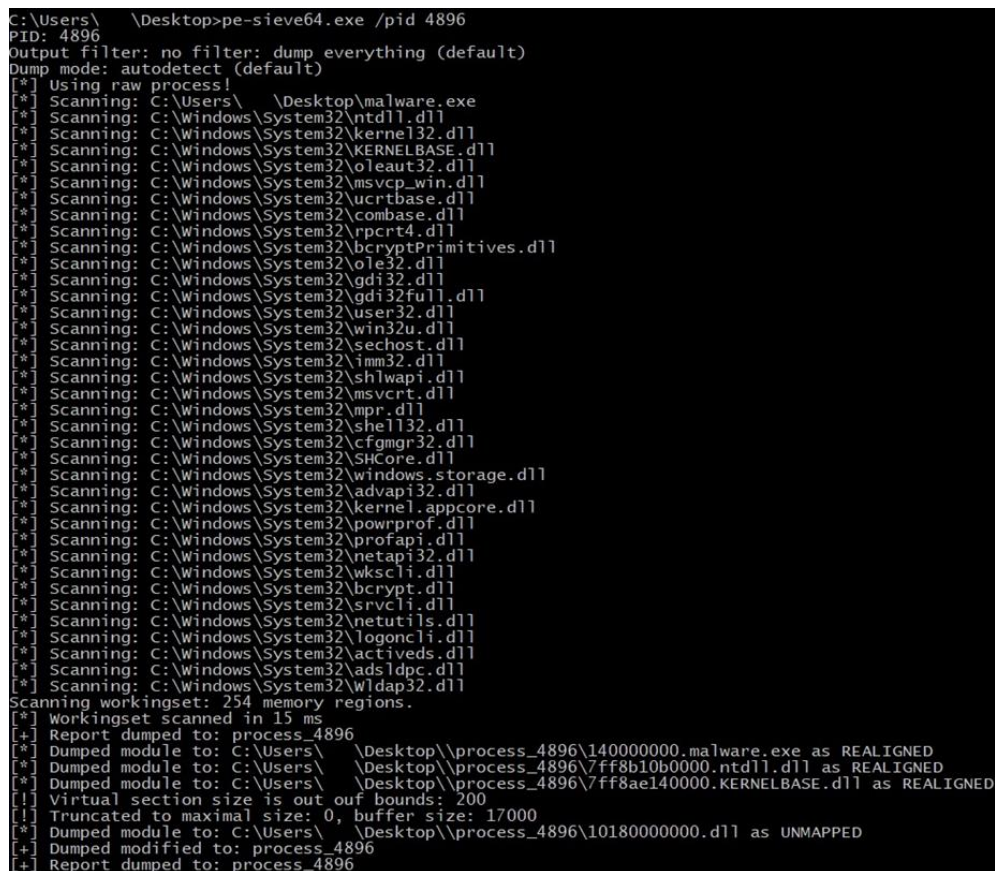


Figure 9

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**):

```

00000101800035F6 33 D2 xor edx,edx
00000101800035F8 33 C9 xor ecx,ecx
00000101800035FA FF 15 30 5E 00 00 call qword ptr ds:[<&CoInitializeEx>]
qword ptr [0000010180009430 <&CoInitializeEx>]=ccombase.CoInitializeEx>
x87Tagword FFFF
Default (x64 fastcall)
1: rcx 0000000000000000
2: rdx 0000000000000000
  
```

Figure 15

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

```

0000010180003600 48 83 64 24 40 00 and qword ptr ss:[rsp+40],0
0000010180003606 8F 03 00 00 00 mov edi,3
0000010180003608 83 64 24 38 00 and dword ptr ss:[rsp+38],0
0000010180003610 45 33 C9 xor r3d,r3d
0000010180003613 48 83 64 24 30 00 and qword ptr ss:[rsp+30],0
0000010180003619 45 33 C0 xor r3d,r3d
000001018000361C 89 7C 24 28 mov dword ptr ss:[rsp+28],edi
0000010180003620 83 CA FF or edx,FFFFFFFF
0000010180003623 83 64 24 20 00 and dword ptr ss:[rsp+20],0
0000010180003628 33 C9 xor ecx,ecx
000001018000362A FF 15 F8 5D 00 00 call qword ptr ds:[<&CoInitializeSecurity>]
RCX 0000000000000000
RDX 00000000FFFFFFFF
RBP 0000000000000000
RSP 000000000014FD80 "LA"
RSI 0000000000000000
RDI 0000000000000003
R8 0000000000000000
R9 0000000000000000
R10 0000000000000000
  
```

Figure 16

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

```

0000010180003630 FF 15 32 5D 00 00 call qword ptr ds:[<&IsUserAnAdmin>]
0000010180003636 89 05 8C 80 00 00 mov dword ptr ds:[1018000B6C81.eax]
  
```

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:

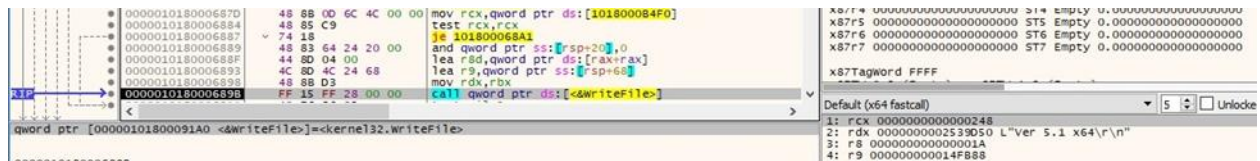


Figure 19

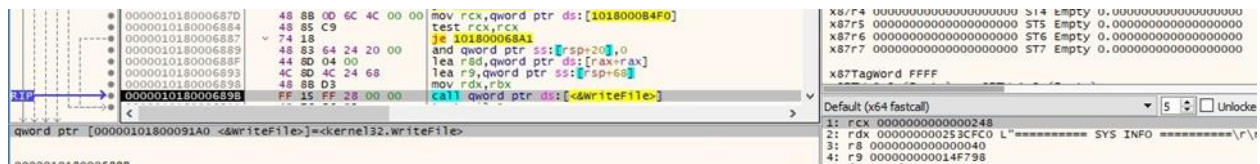


Figure 20

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

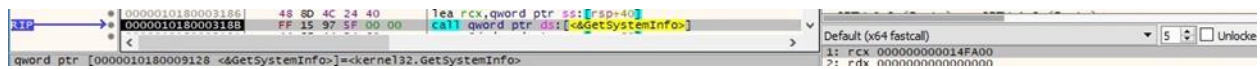


Figure 21

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**):

```

0000010180003217 45 33 C9          xor r9d,r9d
000001018000321A 48 8D 54 24 30   lea rdx,qword ptr ss:[rsp+30]
000001018000321F 48 8D 41 0C       lea r8d,qword ptr ds:[r9+C]
0000010180003223 41 8D 49 01       lea ecx,qword ptr ds:[r9+1]
RIP -> 0000010180003224 FF 15 D3 61 00 00 call qword ptr ds:[<&RtlGetNativeSystemInformation>]
qword ptr [0000010180009400 <&RtlGetNativeSystemInformation>]=<ntdl>.RtlGetNativeSystemInformation>

```

Figure 24

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

```

0000010180003258 48 8D 95 F0 02 00 00 lea rdx,qword ptr ss:[rbp+2F0]
000001018000325F 48 8D 8D D0 00 00 00 lea rdx,qword ptr ss:[rbp+D0]
0000010180003266 89 F0 F0 02 00 00 00 mov dword ptr ss:[rbp+2F0],ebx
RIP -> 0000010180003267 FF 15 A6 5D 00 00 00 call qword ptr ds:[<&GetUserName>]
qword ptr [0000010180009018 <&GetUserName>]=<advapi32.GetUserNeme>

```

Figure 25

```

000001018000329A 48 8D 95 F0 02 00 00 lea rdx,qword ptr ss:[rbp+2F0]
00000101800032A7 48 8D 8D D0 00 00 00 lea rdx,qword ptr ss:[rbp+D0]
RIP -> 00000101800032AE FF 15 64 5E 00 00 00 call qword ptr ds:[<&GetComputerName>]
qword ptr [0000010180009118 <&GetComputerName>]=<kernel32.GetComputerName>

```

Figure 26

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

```

0000010180007714 33 DB          xor ebx,ebx
0000010180007716 4C 8D 40 08     lea r8,qword ptr ds:[rax+8]
000001018000771A 48 8D 50 10     lea rdx,qword ptr ds:[rax+10]
000001018000771E 89 58 08       mov dword ptr ds:[rax+8],ebx
0000010180007721 33 C9          xor ecx,ecx
0000010180007723 48 89 58 10     mov qword ptr ds:[rax+10],rbx
RIP -> 0000010180007724 E8 C1 B1 FF FF call <NetGetJoinInformation>
<NetGetJoinInformation>

```

Figure 27

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

```

0000010180004AF5 48 83 C9 FF     or rcx,FFFFFFFFFFFFFFFF
0000010180004AF9 8B D7          mov edx,edi
0000010180004AFB 4C 8D 44 24 40 lea r8,qword ptr ss:[rsp+40]
RIP -> 0000010180004B00 FF 15 3A 45 00 00 00 call qword ptr ds:[<&OpenProcessToken>]
qword ptr [0000010180009040 <&OpenProcessToken>]=<advapi32.OpenProcessToken>

```

Figure 28

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

```

000001018000486F 44 8B 80 80 03 00 00 mov r9d,dword ptr ss:[rbp+380]
0000010180004876 48 8D 85 80 03 00 00 lea rax,qword ptr ss:[rbp+380]
000001018000487D 48 8B 4C 24 40   mov rcx,qword ptr ss:[rsp+40]
0000010180004882 4C 8B C3        mov r8,rbx
0000010180004885 8B D6          mov edx,esi
0000010180004887 48 89 44 24 20   mov qword ptr ss:[rsp+20],rax
RIP -> 000001018000488C FF 15 96 44 00 00 00 call qword ptr ds:[<&GetTokenInformation>]
qword ptr [0000010180009028 <&GetTokenInformation>]=<advapi32.GetTokenInformation>

```

Figure 29

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

```

00000101800048BA 4C 8D 80 90 03 00 00 lea r9,qword ptr ss:[rbp+390]
00000101800048C1 48 8D 85 88 03 00 00 lea rax,qword ptr ss:[rbp+388]
00000101800048C8 C7 85 88 03 00 00 00 mov dword ptr ss:[rbp+388],100
00000101800048D2 48 89 44 24 28      mov qword ptr ss:[rsp+28],rax
00000101800048D7 4C 8D 44 24 50      lea r8,qword ptr ss:[rsp+50]
00000101800048DC 48 8D 85 50 01 00 00 lea rax,qword ptr ss:[rbp+150]
00000101800048E3 8B F7              mov esi,edi
00000101800048E5 48 03 F6          add rsi,rsi
00000101800048E8 48 89 44 24 20      mov qword ptr ss:[rsp+20],rax
00000101800048ED 33 C9              xor ecx,ecx
00000101800048EF 48 8B 54 F3 08      mov rdx,qword ptr ds:[rbx-rsi*8+8]
00000101800048F4 FF 15 26 44 00 00  call qword ptr ds:[<&LookupAccountSid>]

```

Figure 30

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

```

00000101800049B3 89 01 00 00 00      mov ecx,1
00000101800049B8 FF 15 EA 48 00 00  call qword ptr ds:[<&SetErrorMode>]

```

Figure 31

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

```

00000101800053C2 4C 8D 44 24 34      lea r8,qword ptr ss:[rsp+34]
00000101800053C7 48 8B D3            mov rdx,rbx
00000101800053CA 33 C9              xor ecx,ecx
00000101800053CC FF 15 5E 3C 00 00  call qword ptr ds:[<&LookupPrivilegeValueA>]

```

Figure 32

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

```

00000101800053DE 4C 8D 44 24 30      lea r8,qword ptr ss:[rsp+30]
00000101800053E3 48 83 64 24 20 00  and qword ptr ss:[rsp+20],0
00000101800053E9 45 33 C9            xor r9d,r9d
00000101800053EC 48 8B 4C 24 58      mov rcx,qword ptr ss:[rsp+58]
00000101800053F1 33 D2              xor edx,edx
00000101800053F3 C7 44 24 30 01 00 00 mov dword ptr ss:[rsp+30],1
00000101800053F8 C7 44 24 3C 02 00 00 mov dword ptr ss:[rsp+3C],2
0000010180005408 FF 15 2F 3C 00 00  call qword ptr ds:[<&AdjustTokenPrivileges>]

```

Figure 33

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

```

000001018000815C 40 53              push rbx
000001018000815E 48 83 EC 10        sub rsp,10
0000010180008162 33 C9              xor ecx,ecx
0000010180008164 B8 01 00 00 00    mov eax,1
0000010180008169 0F A2             cpuid
000001018000816B 0F BA EA 0A        bts edx,A
000001018000816F 89 0D 57 33 00 00  mov dword ptr ds:[101800084CC],ecx
0000010180008175 89 15 4D 33 00 00  mov dword ptr ds:[101800084C8],edx
000001018000817B 48 83 C4 10        add rsp,10

```

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:

```

000001018000886C 4C 8B C9 mov r9,r8d
000001018000886F 45 33 C0 xor r8d,r8d
RIP → 0000010180008872 0F 31
0000010180008874 48 C1 E2 20 shl rdx,20
0000010180008878 4C 8D 15 29 2C 00 00 lea r10,qword ptr ds:[1018000B4A8]
000001018000887F 48 08 C2 or rax,rdx
0000010180008882 43 88 14 10 mov edx,dword ptr ds:[r8+r10]
0000010180008886 FF C2 inc edx
0000010180008888 8B C8 mov ecx,eax
000001018000888A 83 E1 07 and ecx,7
000001018000888D D3 C2 rol edx,c1
000001018000888F 33 D0 xor edx,eax
0000010180008891 43 89 14 10 mov dword ptr ds:[r8+r10],edx
0000010180008895 43 89 14 01 mov dword ptr ds:[r9+r8],edx
0000010180008899 49 83 C0 04 add r8,4
000001018000889D 49 83 F8 20 cmp r8,20
00000101800088A1 ^ 72 CF jnb 10180008872

```

Figure 35

Address	Hex	ASCII
0000010180008580	CD D3 85 CD 84 AB F2 C1 01 A4 0B 39 A6 8F ED D7	IÖ.I.«òÀ.Æ.9'.ix
0000010180008590	B3 A6 16 D9 5E E6 FA 37 6F 3A E4 B4 AE 28 15 45	*'.U^æú7o:ä'è+.E

Figure 36

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

```

0000010180005958 41 B9 01 00 00 00 mov r3d,1
0000010180005961 87 44 24 20 00 00 mov dword ptr ss:[rsp+20],F0000000
0000010180005969 4C 8D 45 D7 lea r8,qword ptr ss:[rbp-29]
000001018000596D 33 D2 xor edx,edx
000001018000596E 49 8D 40 C7 lea rcx,qword ptr ss:[rbp-39]
RIP → 0000010180005973 FF 15 37 37 00 00 call qword ptr ds:[<&CryptAcquireContextw>]
qword ptr [00000101800090B0-<&CryptAcquireContextw>]=<advapi32.CryptAcquireContextw>

```

Figure 37

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

```

000001018000597D 48 8B 4D C7 mov rcx,qword ptr ss:[rbp-39]
0000010180005981 48 8D 45 7F lea rax,qword ptr ss:[rbp-79]
0000010180005985 48 89 44 24 28 mov qword ptr ss:[rsp+28],rax
000001018000598A 45 33 C3 xor r3d,r3d
000001018000598D 83 64 24 20 00 and dword ptr ss:[rsp+20],0
0000010180005992 41 8B 14 01 00 00 mov r8d,14
0000010180005995 48 8B D3 mov rdx,r8b
RIP → 000001018000599B FF 15 FF 36 00 00 call qword ptr ds:[<&CryptImportKey>]
qword ptr [00000101800090A0-<&CryptImportKey>]=<advapi32.CryptImportKey>
0000010180005998

```

Figure 38

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

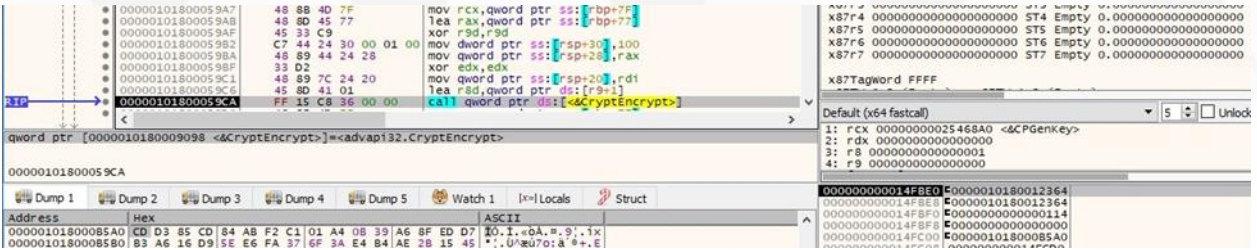


Figure 39

Address	Hex	ASCII
000001800085A0	9D CF E4 44 8F 8F 9F 3D 1B 8A 27 15 E3 8E BA 1D	.IAD...=.'.ä.°.
000001800085B0	A3 AE D2 D3 C9 5D 34 31 94 DD D0 80 27 15 90 7C	z°00E]41.YD'.'
000001800085C0	4C 3F 56 F7 78 7C 03 98 8A D9 59 F6 85 0B B4 57	L'v x ...Üyöu.'W
000001800085D0	76 92 03 AC 7D 54 0D 7C 13 28 34 73 EF D0 F5 F8	v.-~}T. .(4S1D00
000001800085E0	0D 18 DE AE 61 9A 74 4D 83 5C 6A 83 BA 45 62 51	.,p@a.tM. j.°EbQ
000001800085F0	BB FA 79 10 62 10 E0 47 8C 1A 54 75 D3 0F 32 58	»üy.b.äg..Tu0.2X
00000180008600	78 0B 9B 67 DD 72 5E 76 DB BB 88 11 87 58 55 CF	{.gYr^v0»...XUI
00000180008610	AF 83 2A F3 66 47 4E B8 6A BF 34 0D 81 85 9D 98	.°oFGN.jç4....
00000180008620	FB C6 66 0D 4A 72 9D BB 18 FB 0A 3E DF 9C EC A0	Ü&f.Jr.».u.»B.i.
00000180008630	25 76 17 68 E9 83 BB EA 19 B2 5E 47 62 10 B6 76	%v.he*»è.»Agb.¶v
00000180008640	F7 F6 FC 04 39 56 F7 C7 F7 BA A4 AF 5E 17 49 4E	+0U.9v+ç+°m ^.IN
00000180008650	57 8D 97 0B 58 37 48 C7 E1 FD EA 06 8C DE C6 F1	Wv...[7Hçäyè..b&ñ
00000180008660	69 76 AB 0E B7 0D 17 1F 0D D1 F5 FD DA 43 A8 68	iv«.NoyÜ h
00000180008670	B3 03 61 28 1F 32 9E 3F 32 F8 1C 26 B1 21 05 A5	*.a(.2.720.&±l.¥
00000180008680	9A F4 19 21 1D BB 48 30 08 CB 2D 2C 4A 0C EB 7B	.ö.!.»KO.E-.J.è{
00000180008690	A7 C4 1F 4F 95 76 1F 1A 0C 63 79 8D 41 CC 66 A5	§Ä.O.v...cy.AIf¶

Figure 40

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

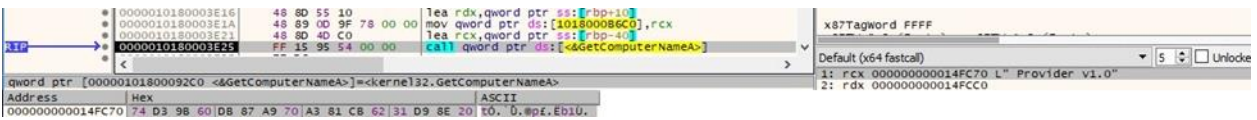


Figure 41

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
0000010180012508	30 66 32 63 64 39 31 64 34 31 36 33 35 31 30 33	0f2cd91d41635103
0000010180012518	61 39 39 39 66 35 65 64 66 65 66 35 30 63 62 36	a999f5edfef50cb6
0000010180012528	37 34 64 33 39 62 36 30 64 62 38 37 61 39 37 30	74d39b60db87a970
0000010180012538	61 33 38 31 63 62 36 32 33 31 64 39 38 65 32 30	a381cb6231d98e20
0000010180012548	0D 0A 09 20 20 0D 0A 09 09 09 3C 2F 70 72 65 3E</pre>
0000010180012558	0D 0A 09 09 3C 2F 62 3E 0D 0A 09 09 3C 68 72 2F<chr/
0000010180012568	3E 0D 0A 54 68 69 73 20 6D 65 73 73 61 67 65 20	>..This message
0000010180012578	63 6F 6E 74 61 69 6E 73 20 61 6E 20 69 6E 66 6F	contains an info
0000010180012588	72 6D 61 74 69 6F 6E 20 68 6F 77 20 74 6F 20 66	rmation how to f
0000010180012598	69 78 20 74 68 65 20 74 72 6F 75 62 6C 65 73 20	ix the troubles
00000101800125A8	79 6F 75 27 76 65 20 67 6F 74 20 77 69 74 68 20	you've got with
00000101800125B8	79 6F 75 72 20 6E 65 74 77 6F 72 68 2E 3C 62 72	your network.<br
00000101800125C8	3E 3C 62 72 3E 0D 0A 0D 0A 46 69 6C 65 73 20 6F	> ...Files o
00000101800125D8	6E 20 74 68 65 20 77 6F 72 68 73 74 61 74 69 6F	n the workstatio
00000101800125E8	6E 73 20 69 6E 20 79 6F 75 72 20 6E 65 74 77 6F	ns in your netwo
00000101800125F8	72 6B 20 77 65 72 65 20 65 6E 63 72 79 70 74 65	rk were encrypte
0000010180012608	64 20 61 6E 64 20 61 6E 79 20 79 6F 75 72 20 61	d and any your a
0000010180012618	74 74 65 6D 70 74 20 74 6F 20 63 68 61 6E 67 65	ttempt to change
0000010180012628	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:

The screenshot shows a debugger window with assembly code on the left and a registry key value on the right. The assembly code includes instructions like `mov dword ptr [esi+20], 2`, `lea r9, qword ptr [rsi+30]`, `add eax, eax`, `lea rcx, qword ptr ds:[10180010658]`, `mov rdx, 1`, `mov dword ptr [rsi+20], eax`, and `call qword ptr [dsi+65*RegSetUSValue]`. The registry key value is `explorer.exe README_TO_DECRYPT.html`.

Figure 44

The screenshot shows the Windows Registry Editor window. The path is `Computer\HKEY_CURRENT_USER\Software\Classes\quantum\shell\Open\command`. The registry value is `explorer.exe README_TO_DECRYPT.html`.

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" ".cp1" ".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" "dbnmp.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**):

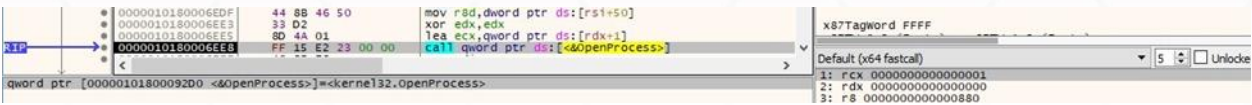


Figure 54

The process is killed by calling the TerminateProcess API:



Figure 55

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

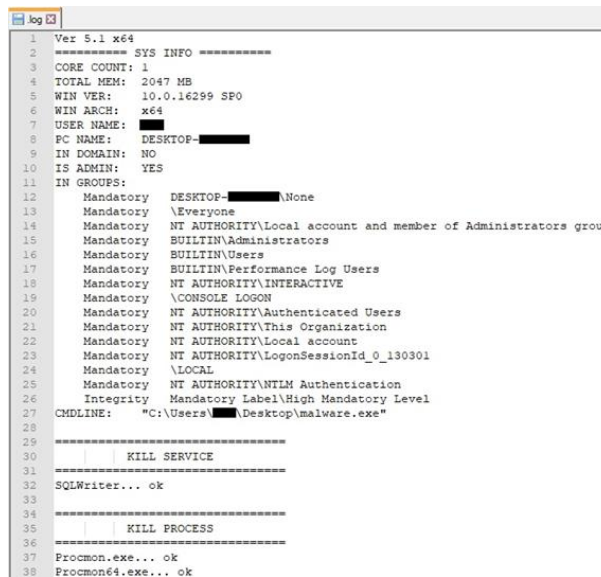


Figure 56

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**):

```

000001018000732F C7 4D DC 00 40 00 00 mov dword ptr ds:[rax-24],4000
0000010180007330 4C 8B CA          mov r9,rdx
0000010180007331 48 80 E0          lea rax,qword ptr ds:[rax-20]
0000010180007332 33 D2            xor edx,edx
0000010180007333 48 89 44 24 20    mov qword ptr ss:[rsp+20],rax
0000010180007334 4D 8B F8          mov r15,r8
0000010180007335 8B E9            mov ebp,ecx
0000010180007336 80 44 02          lea ecx,qword ptr ds:[rdx+2]
0000010180007337 44 80 42 13      lea r8d,qword ptr ds:[rdx+13]
0000010180007338 E8 4A B5 FF FF    call <@NetOpenEnum>

```

Figure 63

WNetEnumResourceW is utilized to continue the enumeration of network resources:

```

0000010180007395 48 8B 4C 24 38    mov rcx,qword ptr ss:[rsp+38]
0000010180007396 4C 8B CA          lea r9,qword ptr ss:[rbp-10]
0000010180007397 4C 8B C0          mov r8,qword ptr ss:[rbp-10]
0000010180007398 48 80 54 24 30    lea rdx,qword ptr ss:[rsp+30]
0000010180007399 E8 F9 84 FF FF    call <@WNetEnumResourceW>

```

Figure 64

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

```

0000010180004600 83 65 7F 00      and dword ptr ss:[rbp-7F],0
0000010180004601 48 80 45 E7      lea rax,qword ptr ss:[rbp-10]
0000010180004602 48 83 65 EF 00   and qword ptr ss:[rbp-10],0
0000010180004603 4C 80 45 EF      lea r8,qword ptr ss:[rbp-10]
0000010180004604 48 89 44 24 30  mov qword ptr ss:[rsp+30],rax
0000010180004605 41 83 C9 FF      or r9d,FFFFFFFF
0000010180004606 48 80 45 EB      lea rax,qword ptr ss:[rbp-15]
0000010180004607 8A 01 00 00 00   mov edx,1
0000010180004608 48 89 44 24 28   mov qword ptr ss:[rsp+28],rax
0000010180004609 49 8B CF          mov rcx,r15
000001018000460A 48 80 45 7F      lea rax,qword ptr ss:[rbp+7F]
000001018000460B 48 89 44 24 20   mov qword ptr ss:[rsp+20],rax
000001018000460C E8 AE E2 FF FF    call <@NetShareEnum>

```

Figure 65

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

```

00000101800069F8 48 80 54 24 20    lea rdx,qword ptr ss:[rsp+20]
0000010180006A00 FF 15 72 29 00 00 mov rcx,rdx
0000010180006A01 FF 15 72 29 00 00 call qword ptr ds:[<@StrStrIW>]

```

Figure 66

The files are enumerated using the FindFirstFileW and FindNextFileW functions:

```

00000101800088BE 48 8D 91 20 00 02 00 lea rdx,qword ptr ds:[rcx+20020]
00000101800088C5 33 ED            xor ebp,ebp
00000101800088C7 C7 44 59 20 2A 00 00 mov dword ptr ds:[rcx+rbx*2+20],2A
00000101800088E6 48 83 C1 20      add rcx,20
00000101800088D3 FF 15 97 08 00 00 call qword ptr ds:[<@FindFirstFileW>]

```

Figure 67

```

0000010180003A09 48 8B D5          mov rdx,rbp
0000010180003A0C 48 8B CE          mov rcx,r15
0000010180003A0F FF 15 63 S7 00 00 call qword ptr ds:[<@FindNextFileW>]

```

Figure 68

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

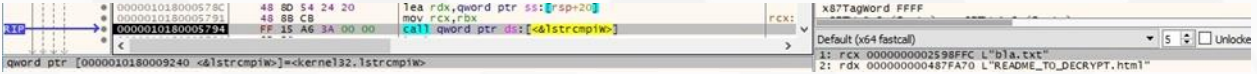


Figure 69

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**):

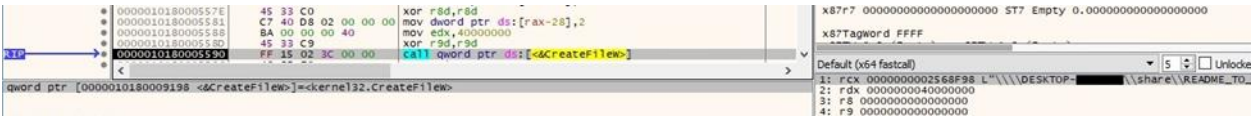


Figure 72

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**):

```

00000101800064D4 48 83 60 E8 00 and qword ptr ds:[rax-18],0
00000101800064D3 48 8B DA 00 00 mov rdx,rdx
00000101800064DC 83 60 E0 00 00 and dword ptr ds:[rax-20],0
00000101800064ED 48 8B F9 00 00 mov rdi,rcx
00000101800064E3 48 8B C8 00 00 mov rcx,rdx
00000101800064E6 C7 40 D8 03 00 00 00 mov dword ptr ds:[rax-28],3
00000101800064E4 45 33 C9 00 00 xor r9d,r9d
00000101800064F0 45 33 C0 00 00 xor r8d,r8d
00000101800064E3 8A 00 00 01 C0 00 mov edx,C0010000
00000101800064F8 FF 15 9A 2C 00 00 call qword ptr ds:[<&CreateFileW>]
qword ptr [0000010180009198 <&CreateFileW>]=<kernel32.CreateFileW>

```

Figure 75

The size of the file is obtained using GetFileSizeEx:

```

000001018000652F 48 8D 57 08 00 lea rdx,qword ptr ds:[rdi+8]
0000010180006536 FF 15 1C 2D 00 00 call qword ptr ds:[<&GetFileSizeEx>]
qword ptr [0000010180009258 <&GetFileSizeEx>]=<kernel32.GetFileSizeEx>

```

Figure 76

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

```

0000010180004CD1 44 88 88 48 00 10 00 mov r9d,qword ptr ds:[rax+100048]
0000010180004CD8 4C 8D 80 38 00 10 00 lea r8,qword ptr ds:[rax+100038]
0000010180004CDF 48 8B 08 00 00 00 00 mov rcx,qword ptr ds:[rax]
0000010180004CE2 BA 03 00 00 00 00 00 mov edx,3
0000010180004CE7 46 8D 0C 4D 18 00 00 lea r9d,qword ptr ds:[r9+2+18]
0000010180004CEF FF 15 78 45 00 00 call qword ptr ds:[<&SetFileInformationByHandle>]
qword ptr [0000010180009270 <&SetFileInformationByHandle>]=<kernel32.SetFileInformationByHandle>
0000010180004CEF

```

Figure 77

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

Address	Hex	ASCII
000001018000B4A8	29 6A 37 59 07 9C 9E DB B2 A8 3F 98 9F D2 D3 20	}j7Y...0* ?.00
000001018000B4B8	A9 BC 38 B9 57 43 CC A2 58 52 AA B4 1F CD 28 98	@48'wCieXR'.I.

Figure 78

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

Address	Hex	ASCII
0000000004B7FCE0	65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 68	expand 32-byte k
0000000004B7FCF0	CD D3 85 CD 84 AB F2 C1 01 A4 0B 39 A6 8F ED D7	I0.I.«0A.0.9'.ix
0000000004B7FD00	B3 A6 16 D9 5E E6 FA 37 6F 3A E4 B4 AE 28 15 45	''.U^æu70:â@+.E
0000000004B7FD10	CD D3 85 CD 84 AB F2 C1 01 A4 0B 39 A6 8F ED D7	I0.I.«0A.0.9'.ix

Figure 79

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

```

RIP → 0000010180001720 66 0F FE C1 paddb xmm0,xmm1
      0000010180001724 66 0F EF D8 pxor xmm3,xmm0
      0000010180001728 66 0F 38 00 DE pshufb xmm3,xmm6
      000001018000172D 66 0F FE D3 paddb xmm2,xmm3
      0000010180001731 66 0F EF CA pxor xmm1,xmm2
      0000010180001735 66 0F 6F E1 movdqa xmm4,xmm1
      0000010180001739 66 0F 72 D1 14 psrlq xmm1,14
      000001018000173E 66 0F 72 F4 0C psllq xmm4,C
      0000010180001743 66 0F EB CC por xmm1,xmm4
      0000010180001747 66 0F FE C1 paddb xmm0,xmm1
      000001018000174B 66 0F EF D8 pxor xmm3,xmm0
      000001018000174F 66 0F 38 00 DF pshufb xmm3,xmm7
      0000010180001754 66 0F FE D3 paddb xmm2,xmm3
      0000010180001758 66 0F EF CA pxor xmm1,xmm2
      000001018000175C 66 0F 6F E1 movdqa xmm4,xmm1
      0000010180001760 66 0F 72 D1 19 psrlq xmm1,19
      0000010180001765 66 0F 72 F4 07 psllq xmm4,7
      000001018000176A 66 0F EB CC por xmm1,xmm4
      000001018000176E 66 0F 70 D2 4E pshufd xmm2,xmm2,4E
      0000010180001773 66 0F 70 C9 39 pshufd xmm1,xmm1,39
      0000010180001778 66 0F 70 DB 93 pshufd xmm3,xmm3,93
      000001018000177D 90 nop
      000001018000177E 66 0F FE C1 paddb xmm0,xmm1
      0000010180001782 66 0F EF D8 pxor xmm3,xmm0
      0000010180001786 66 0F 38 00 DE pshufb xmm3,xmm6
      000001018000178B 66 0F FE D3 paddb xmm2,xmm3
      000001018000178F 66 0F EF CA pxor xmm1,xmm2
      0000010180001793 66 0F 6F E1 movdqa xmm4,xmm1
      0000010180001797 66 0F 72 D1 14 psrlq xmm1,14
      000001018000179C 66 0F 72 F4 0C psllq xmm4,C
      00000101800017A1 66 0F EB CC por xmm1,xmm4
      00000101800017A5 66 0F FE C1 paddb xmm0,xmm1
      00000101800017A9 66 0F EF D8 pxor xmm3,xmm0
      00000101800017AD 66 0F 38 00 DF pshufb xmm3,xmm7
      00000101800017B2 66 0F FE D3 paddb xmm2,xmm3
      00000101800017B6 66 0F EF CA pxor xmm1,xmm2
      00000101800017BA 66 0F 6F E1 movdqa xmm4,xmm1
      00000101800017BE 66 0F 72 D1 19 psrlq xmm1,19
      00000101800017C3 66 0F 72 F4 07 psllq xmm4,7
      00000101800017C8 66 0F EB CC por xmm1,xmm4
      00000101800017D2 66 0F 70 D2 4E pshufd xmm2,xmm2,4E
      00000101800017D1 66 0F 70 C9 93 pshufd xmm1,xmm1,93
      00000101800017D6 66 0F 70 DB 39 pshufd xmm3,xmm3,39
      00000101800017DB 49 FF C8 dec r8
      00000101800017DE 66 0F 85 3C FF FF FF 39E 30180001720
      00000101800017E2 66 0F FE 04 24 10 paddb xmm0,xmmword ptr ss:[rsp]
      00000101800017E9 66 0F FE 4C 24 10 paddb xmm1,xmmword ptr ss:[rsp+10]
      00000101800017EF 66 0F FE 54 24 20 paddb xmm2,xmmword ptr ss:[rsp+20]
      00000101800017F5 66 0F FE 5C 24 30 paddb xmm3,xmmword ptr ss:[rsp+30]
      00000101800017FB 48 83 FA 40 cmp rdx,40

```

Figure 80

Address	Hex	ASCII
0000000004B7FD94	B0 11 CA B3 95 B7 3D 34 90 E2 11 FF E7 C7 98 DB	°.È*. .=4. à.ÿçç.0
0000000004B7FDA4	9E 3D 54 59 84 EF E9 88 74 B2 BA DC 57 32 0A DF	.,=TY. i.é.t*°Uw2.ß

Figure 81

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

```

0000010180003AE8 48 8B 0B mov rcx,qword ptr ds:[rbx]
0000010180003AEB 41 89 02 00 00 00 mov r9d,r9d
0000010180003AF1 45 33 C0 xor r9d,r9d
0000010180003AF4 33 D2 xor edx,edx
RIP → 0000010180003AF6 FF 15 7C 57 00 00 call qword ptr ds:[.<&SetFilePointerEx>]
qword ptr [0000010180009278 <&SetFilePointerEx>]=<kernel32.SetFilePointerEx>

```

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:

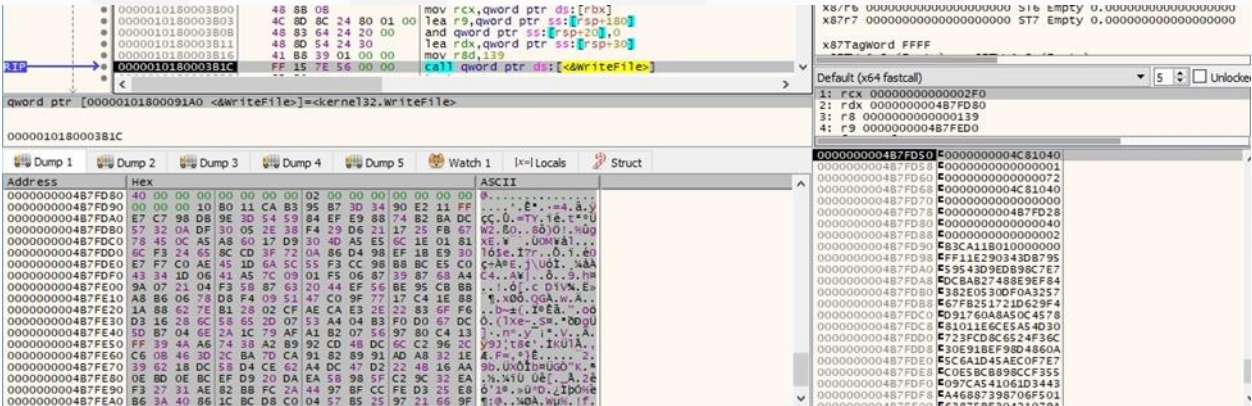


Figure 83

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
000000004B7FE00	65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 68	expand 32-byte k
000000004B7FE10	29 6A 37 59 07 9C 9E DB B2 A8 3F 98 9F D2 D3 20	}j7Y...0*?.\00
000000004B7FE20	A9 BC 38 B9 57 43 CC A2 58 52 AA B4 1F CD 28 98	@48'WCiEXR*.I(.
000000004B7FE30	29 6A 37 59 07 9C 9E DB B2 A8 3F 98 9F D2 D3 20	}j7Y...0*?.\00

Figure 85

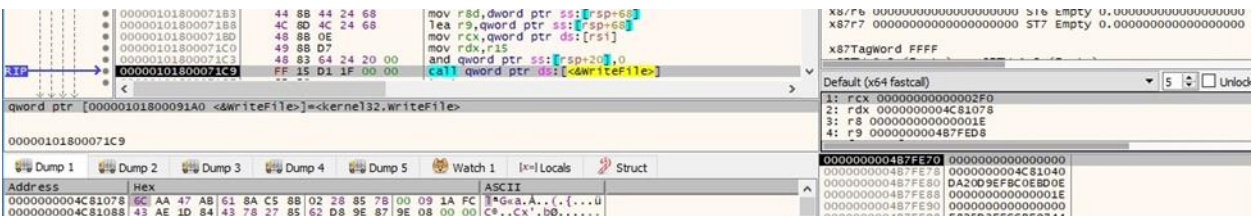


Figure 86

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:

```

000001018000556C 48 83 60 E8 00 and qword ptr ds:[rax-18],0
0000010180005571 49 88 F8      mov rdi,r8
0000010180005574 48 88 DA      mov rbx,rdx
0000010180005577 C7 40 E0 80 00 00 mov dword ptr ds:[rax-20],80
000001018000557E 45 33 C0      xor r8d,r8d
0000010180005581 C7 40 D8 02 00 00 00 mov dword ptr ds:[rax-28],2
0000010180005588 BA 00 00 40   mov edx,40000000
000001018000558D 45 33 C9      xor r9d,r9d
0000010180005590 FF 15 02 3C 00 00 call qword ptr ds:[<&CreateFileW>]

```

qword ptr [0000010180009198 <&CreateFileW>]=<kernel32.CreateFileW>

Address	Hex	ASCII
000000000014F980	48 00 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00	E...U...S...e...r...s...
000000000014F990	5C 00 00 00 00 00 00 00 00 00 41 00 70 00 70 00A.p.p.
000000000014F9A0	44 00 61 00 74 00 61 00 5C 00 4C 00 6F 00 63 00	D...a...t...a...L...o...c...
000000000014F9B0	61 00 6C 00 5C 00 54 00 65 00 6D 00 70 00 5C 00	a...l...t...e...m...p...l...
000000000014F9C0	5C 00 30 00 34 00 32 00 36 00 46 00 41 00 43 00	\...D...a...t...e...F...A...C...
000000000014F9D0	44 00 2E 00 62 00 61 00 74 00 00 00 00 00 00 00	D...b...a...t...e...t...e...t...

Figure 88

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

```

000001018000559F 48 83 64 24 20 00 and qword ptr ss:[rsp+20],0
00000101800055A5 4C 8D 4C 24 68 lea r9,qword ptr ss:[rsp+68]
00000101800055AA 44 88 C7      mov r8d,edi
00000101800055AD 48 8B D3      mov rdx,rbx
00000101800055B0 48 8B C8      mov rcx,rax
00000101800055B3 FF 15 E7 38 00 00 call qword ptr ds:[<&writeFile>]

```

qword ptr [00000101800091A0 <&writeFile>]=<kernel32.writeFile>

Address	Hex	ASCII
000000000014F920	61 74 74 72 69 62 20 2D 73 20 2D 72 20 2D 68 20	attrib -s -r -h
000000000014F930	25 31 0D 0A 3A 6C 0D 0A 64 65 6C 20 2F 46 20 2F	Q !...!.del /F /
000000000014F940	51 20 25 31 0D 0A 69 66 20 65 78 69 73 74 20 25	Q !...! if exist %
000000000014F950	31 20 67 6F 74 6F 20 6C 0D 0A 64 65 6C 20 25 30	I goto 1..del %0

Figure 89

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

```

0000010180007D4D 45 33 C9      xor r9d,r9d
0000010180007D50 48 89 44 24 48 mov qword ptr ss:[rsp+48],rax
0000010180007D55 48 8D 95 90 02 00 00 lea rdx,qword ptr ss:[rbp+290]
0000010180007D58 48 8D 45 90 lea rax,qword ptr ss:[rbp-70]
0000010180007D60 45 33 C0      xor r8d,r8d
0000010180007D63 48 89 44 24 40 mov qword ptr ss:[rsp+40],rax
0000010180007D68 33 C9 C9     xor ecx,ecx
0000010180007D6A 48 89 7C 24 38 mov qword ptr ss:[rsp+38],rdi
0000010180007D6F 48 89 7C 24 30 mov qword ptr ss:[rsp+30],rdi
0000010180007D74 C7 44 24 28 00 00 00 mov dword ptr ss:[rsp+28],80000000
0000010180007D7C 89 7C 24 20 mov dword ptr ss:[rsp+20],edi
0000010180007D80 FF 15 AA 19 00 00 call qword ptr ds:[<&CreateProcessW>]

```

qword ptr [0000010180009130 <&CreateProcessW>]=<kernel32.CreateProcessW>

Address	Hex	ASCII
000000000014F980	48 00 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00	E...U...S...e...r...s...
000000000014F990	5C 00 00 00 00 00 00 00 00 00 41 00 70 00 70 00A.p.p.
000000000014F9A0	44 00 61 00 74 00 61 00 5C 00 4C 00 6F 00 63 00	D...a...t...a...L...o...c...
000000000014F9B0	61 00 6C 00 5C 00 54 00 65 00 6D 00 70 00 5C 00	a...l...t...e...m...p...l...
000000000014F9C0	5C 00 30 00 34 00 32 00 36 00 46 00 41 00 43 00	\...D...a...t...e...F...A...C...
000000000014F9D0	44 00 2E 00 62 00 61 00 74 00 00 00 00 00 00 00	D...b...a...t...e...t...e...t...

Figure 90

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

Quantum ransomware creates a new console using the AllocConsole routine:

```

00000101800056C9 FF 15 09 3A 00 00 call qword ptr ds:[<&AllocConsole>]

```

qword ptr [00000101800090D8 <&AllocConsole>]=<kernel32.AllocConsole>

Figure 91

The executable retrieves a handle to the standard output device using GetStdHandle (0xFFFFFFFF5 = STD_OUTPUT_HANDLE):

```

00000101800056D3 89 F5 FF FF FF mov ecx,FFFFFFFF
00000101800056D8 FF 15 22 3A 00 00 call qword ptr ds:[<&GetStdHandle>]
qword ptr [0000010180009100 <&GetStdHandle>]=<kernel32.GetStdHandle>

```

Figure 92

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

```

0000010180005239 48 83 64 24 28 00 and qword ptr ss:[rsp+28],0
000001018000533F 4C 8D 05 BA DC FF FF lea r8,qword ptr ds:[10180003000]
0000010180005346 83 64 24 20 00 and dword ptr ss:[rsp+20],0
0000010180005348 4C 8B CB mov r9,rdx
000001018000534E 33 D2 xor edx,edx
0000010180005350 33 C9 xor ecx,ecx
0000010180005352 FF 15 C0 3E 00 00 call qword ptr ds:[<&CreateThread>]
qword ptr [0000010180009218 <&CreateThread>]=<kernel32.CreateThread>

```

Figure 93

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

```

0000010180007E15 4C 8D 85 80 01 00 00 lea r8,qword ptr ss:[rbp+180]
0000010180007E1C 33 D2 xor edx,edx
0000010180007E20 8B 8E AA FF FF call <NetGetDCName>
<NetGetDCName>

```

Figure 94

The WNetOpenEnumW function is used to start enumerating the resources on the local network (Ox2 = **RESOURCE_GLOBALNET**):

```

0000010180007336 4C 8B CA mov r9,rdx
0000010180007339 48 8D 40 E0 lea rax,qword ptr ds:[rax-20]
000001018000733D 33 D2 xor edx,edx
000001018000733F 48 89 44 24 20 mov qword ptr ss:[rsp+20],rax
0000010180007344 4D 8B F8 mov r15,r8
0000010180007347 8B E9 mov ebp,ecx
0000010180007349 8D 8A 02 lea ecx,qword ptr ds:[rdx+2]
000001018000734C 44 8D 42 13 lea r8d,qword ptr ds:[rdx+13]
0000010180007350 E8 4A B5 FF FF call <WNetOpenEnumW>
<WNetOpenEnumW>

```

Figure 95

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

```

0000010180007395 48 8B 4C 24 38 mov rcx,qword ptr ss:[rsp+38]
000001018000739A 4C 8D 4C 24 34 lea r9,qword ptr ss:[rsp+34]
000001018000739F 4C 8B C0 mov r8,rax
00000101800073A2 48 8D 54 24 30 lea rdx,qword ptr ds:[rsp+30]
00000101800073A7 E8 F9 B4 FF FF call <WNetEnumResourceW>
<WNetEnumResourceW>

```

Figure 96

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

```

0000010180005CD0 4C 8B C2 mov r8,rdx
0000010180005CD3 48 8D 4C 24 20 lea rcx,qword ptr ss:[rsp+20]
0000010180005CD8 41 B9 04 00 00 00 mov r9d,4
0000010180005CDE 48 8B D0 mov rdx,rax
0000010180005CE1 0F 11 44 24 20 movups xmmword ptr ss:[rsp+20],xmm0
0000010180005CE6 0F 11 44 24 20 movups xmmword ptr ss:[rsp+40],xmm0
0000010180005CEB E8 A3 C8 FF FF call <WNetAddConnection2W>
<WNetAddConnection2W>

```

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):

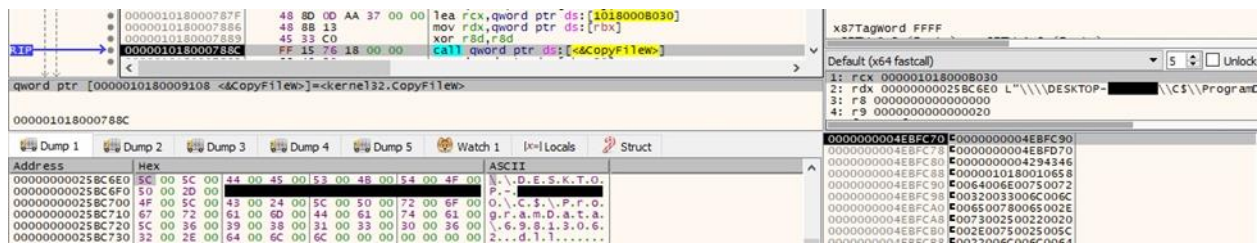


Figure 99

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

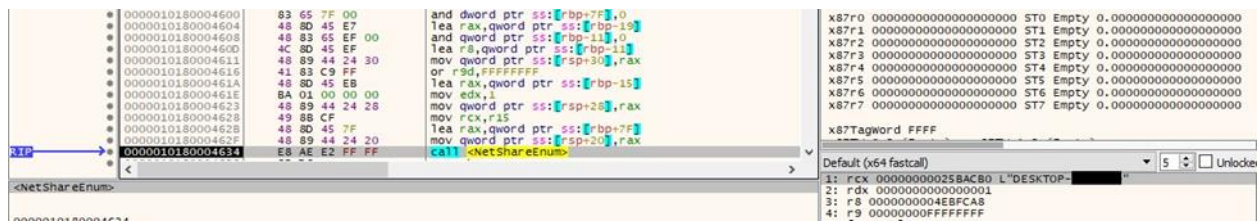


Figure 100

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:

```
eula.2052.txt.quantum
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00004540 73 00 2E 00 20 00 40 00 00 00 00 00 00 02 00 s...@
00004550 00 00 00 00 00 00 0A 00 00 00 2D DA E6 DC 4D 71 .....-ÛæÜMç
00004560 8F 3C 8C 55 1E 40 5D 9C 49 34 95 FD CE F8 AF ED .<EU.}@I4•ýÏø~i
00004570 71 F4 F4 43 33 AB A0 04 E9 E7 A0 99 9B E8 B3 EE qôóC3« .éc™)è'ì
00004580 FA 80 4E CA 09 53 F1 BC A7 3B CA 9E CD DA CD 8C éENÈ.Sñ+S;ÊzÍÚIG
00004590 F9 9B B6 C4 D8 F1 AD F5 67 59 94 78 DE 3B F3 AB ù>ŹÄ0ñ.õgY"xP;ó«
000045A0 58 FA DA C5 7A 02 44 88 BE FA 15 9C F4 70 DC C7 KúÚÁz.D^nú.œópÛÇ
000045B0 10 11 12 8D 22 42 D7 0B 0F B9 D5 AB 06 4F BD B3 ..."B×...²Ö«.O½³
000045C0 43 7D A3 F4 02 7B 59 55 2E 0E 94 D4 CA B9 07 8F C)£ó.(YU..°ÖÈ¹..
000045D0 79 E3 09 4B A1 A2 B2 87 50 D4 F7 46 C7 EF 56 4C yã.K;ç^+PÔ÷FÇ1VL
000045E0 F6 56 1F 75 FD 77 1C 02 D4 C0 FD 95 4C C2 D3 DC öV.uýw..ÔÁý•LÁÓÛ
000045F0 82 AD 46 7D 8B 55 99 F0 1E D6 D9 C0 BC F9 18 DF ,.F)<U™ð.ÖÜÁ+ù.ß
00004600 A2 48 2B EE 45 EC 36 F1 C0 05 F7 1D EA D6 F5 AE çH+iEi6ñA.÷.èÖöø
00004610 FF 6D 28 79 96 B8 3D BE 70 3C 3A EA 05 15 3B 03 ŷm(y-,=³q<:ê...;
00004620 EC BB E1 92 F2 6B 8F C7 90 69 26 A2 A2 6B E9 28 ì»á'òk.Ç.i&ocké(
00004630 A2 AD 4A 67 40 72 35 D1 D3 30 43 39 C2 9E EA 52 ç.Jg@r5ÑÓC9ÂžèR
00004640 C9 14 C6 DB C5 E5 AC 6B 16 05 D8 CB BF 04 02 6B È.ÆÜÁâ-k..øÈ¿..k
00004650 2D 43 22 1D 08 2C 59 D3 58 F3 93 8C A2 A3 E9 88 -C"...YÓXó"€œÉé~
00004660 F3 99 FD 22 91 A8 07 5E 24 35 07 83 AE 38 12 2B ó™ý"´.^$5.fø8.+
00004670 76 76 DC E0 45 05 D4 CB 70 4A 32 9E F4 EA 29 vvÜâE.ÖÈpJ2žöè)
```

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