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	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

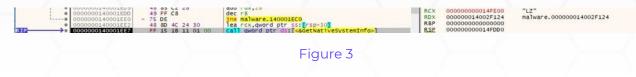
The malware is a 64-bit executable that uses the XOR algorithm to decrypt a DLL file, as highlighted below:

2112			140001 140001 140001 140001 140001 140001 140001 140001 140001 140001	353 356 358 350 360 363 365 365 368 368 368 368 368	42434474447	F B6 0 2A 0 8 1 32 0 03 9 3B 8 FF 9 FF 5 D0 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8	CG CD CA C2 C7 CB	24 3	10	-	sub and xor add cmp jb inc inc dec jne	al, al, byt rcx rcx rdx rdx rdi r11 mal	8 r8b (,rbp (,r10 (are.	r ds	0139	0 40		+30]			>
byte ptr [r al=4C 'L' .text:00000																					
al=4C 'L'	000140			re.ex	e:\$1		75 B	010	Dump	5	6	Wato	h 1	[X=]	Loca	ls	2	Struct			
al=4C 'L'	000140	00135B	malwa	re.ex	e:\$1	35B #	75 B	819	Dump	5	6	Wato	h 1	1. St. 1	Loca	ls	2	Struct			

Figure 1

Address	He	(ASCII
000000014002EDD0	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZÿÿ
000000014002EDE0	88	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	00	
000000014002EE10	OE	1F	BA	OE	00	B4	09	CD	21	88	01	4C	CD	21	54	68	º'.1!L1!Th
000000014002EE20	69	73	20	70	72	6F	67	72	61	GD	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	OD	0D	0A	24	00	00	00	00	00	00	00	mode\$
000000014002EE50	C7	2A	54	0C	83	4B	3A	5F	83	4B	3A	5F	83	4B	3A	5F	C*TK:K:K:_
000000014002EE60	97	20	3C	5E	82	4B	3A	5F	A4	8D	41	5F	81	4B	3A	5F	. <^.K:_¤.AK:_
000000014002EE70	97	20	3B	5E	95	4B	3A	5F	83	4B	3B	5F	03	4B	3A	5F	. ;^.K:K;K:_
000000014002EE80	7F	3C	83	5F	80	4B	3A	5F	1D	EB	FD	5F	82	4B	3A	5F	. <k:ëýk:_< td=""></k:ëýk:_<>
000000014002EE90	3D	ЗA	3F	5E	82	4B	3A	5F	83	4B	3A	5F	82	4B	ЗA	5F	=:?^.K:K:K:_
000000014002EEA0	18	39	3E	5E	90	4B	3A	5F	18	39	3A	5E	82	4B	3A	5F	.9>^.K:9:^.K:_
000000014002EEB0	18	39	38	5E	82	4B	3A	5F	52	69	63	68	83	4B	ЗA	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	B7	B 3	61	00	00	00	00	PEdj.*a
000000014002EEE0	00	00	00	00	FO	00	22	20	OB	02	OE	10	00	78	00	00	ð"x
000000014002EEF0	00	BA	00	00	00	00	00	00	38	68	00	00	00	10	00	00	.°8k

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



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

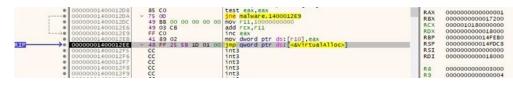


Figure 4

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

0000000140001C3F 0000000140001C44 0000000140001C44 0000000140001C47	49 80 4C 05 00 FF 56 38 4C 88 E0	lea rcx,qword ptr ds:[r13+rax] call qword ptr ds:[rs1+38] mov r12,rax	_	x87Tagword FFFF	
0000000140001C4A	48 85 CO	test rax, rax	, v	Default (x64 fastcall)	▼ 5 ¢ 🗌 Unlocke
gword ptr [rsi+38]=[00000000025234	C8 <&LoadLibraryA>]=<	malware.LoadLibraryA>	,	1: rcx 0000010180009F88 "ntdll.dll"	

Figure 5

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

0000000140001CBF 0000000140001CC4	4A 8D 54 28 02 FF 56 40	<pre>lea rdx,qword ptr ds:[rax+r13+2] call gword ptr ds:[rs1+40]</pre>	x87Tagword FFFF	
0000000140001CC7	48 89 04 1F	mov gword ptr ds:[rdi+rbx],rax	Default (x64 fastcall)	▼ 5 🗢 🗆 Unlocke
qword ptr [rsi+40]=[000000002523	4D0 <&GetProcAddress	>]= <malware.getprocaddress></malware.getprocaddress>	1: rcx 00007FF881080000 n 2: rdx 0000010180009F6A "	tdll.00007FF8B10B0000 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:

		01400021B5 01400021B7	FF D0 85 C0	test e		>
rax=0000010	0180006838					
.text:00000	0001400021B	5 malware.ex	e:\$2185 #1585			
Dump 1	Dump 2	Dump 3	💭 Dump 4	📖 Dump 5 🛛 🧐 Wat	ch 1 🛛 [x=] Locals 🛛 🖉 Struct	
Dump 1 Address	Dump 2	💭 Dump 3	Dump 4	💭 Dump 5 🛛 👹 Wat	ch 1 [x= Locals 22 Struct	

<u>PE-sieve</u> 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.d]]
[*] Scanning: C:\Windows\System32\oleaut32.dll
[*] Scanning: C:\Windows\System32\msvcp_win.d]]
[*] Scanning: C:\Windows\System32\ucrtbase.dll
[*] Scanning: C:\Windows\System32\combase.d]]
[*] 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.dl1
[*] Scanning: C:\Windows\System32\win32u.dl]
[*] Scanning: C:\Windows\System32\sechost.dll
[*] Scanning: C:\Windows\System32\imm32.d]]
[*] Scanning: C:\Windows\System32\sh]wapi.d]]
[*] 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\SHČore.dll
[*] Scanning: C:\Windows\System32\windows.storage.dll
[*] Scanning: C:\Windows\System32\advapi32.d]]
[*] Scanning: C:\Windows\System32\kernel.appcore.dll
[*] Scanning: C:\Windows\System32\powrprof.dll
[*] Scanning: C:\Windows\System32\profapi.d]]
[*] 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.d]]
[*] Scanning: C:\Windows\System32\ads1dpc.dl1
[*] 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.dl] as REALIGNEE
[!] Virtual section size is out ouf bounds: 200
[!] Truncated to maximal size: 0, buffer size: 17000
[*] Dumped module to: C:\Users\ \Desktop\\process_4896\10180000000.d]] 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 ; Runk
.c:0000010180005E50	call cs:GetCommandLineW
.c:0000010180005E56	mov rcx, rax
.c:0000010180005E59	call sub 10180005818

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.

00000101800040F9 00000101800040FC	48 88 C8 48 80 55 20	<pre>mov rcx,rax lea rdx,qword ptr ss:[rbp+20] call qword ptr ds:[<&CommandLineToArgvW>]</pre>		x87TagWord FFFF	
	FF 15 5A 52 00 00	<pre>call qword ptr ds:[<&CommandLineToArgvw>]</pre>	> ×	Default (x64 fastcall)	🗸 💈 💭 Unlocke
qword ptr [0000010180009360 <&Com	andLineToArgvw>]= <she< td=""><td>1132.CommandLineToArgvW></td><td></td><td>1: rcx 00000000252285A L"\"C:\\ 2: rdx 00000000014FE20</td><td>Users\\\\Desktop\\maiwark</td></she<>	1132.CommandLineToArgvW>		1: rcx 00000000252285A L"\"C:\\ 2: rdx 00000000014FE20	Users\\\\Desktop\\maiwark

Figure 11

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

<pre> 0000010180004114 41 88 04 01 00 00</pre>	x87TagWord FFFF
312 000001010100004124 FF 15 B6 51 00 00 call qword ptr ds:[<&GetModuleFileNamew>]	> Default (x64 fastcall) • 5 🗘 Unlock
qword ptr [00000101800092E0 <&GetModuleFileNamew>]= <kernel32.getmodulefilenamew></kernel32.getmodulefilenamew>	1: rCX 0000010180000000 2: rCX 0000010180008030 3: rS 00000000000000104

Figure 12

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

0010180004F02	EB 84 48 88 4		48	Watch 1	[x=] Locals	rsp+48]	
0010180004F02	48 8B 4			mov rax,qwo	ord ptr ss:		
0010180004F02		4 24	48			rsp+48	
0010180004F02		4 24	48			rsp+48	
0010180004F02		44 24	48			rsp+48	
0010180004F00 ^							
0010180004EFD	88 04 0	DA		imp 1018000		cx],al	
0010180004EF8	48 8B 5	4 24		mov rdx, qwo	ord ptr ss:	rsp+48	
		1C 24	20			[rsn+20]	
0010180004EEC	OF 86 4		28	movzx ecx, b	yte ptr ss:		
			40				
0010180004EDE	OF 87 4	44 24		movzx eax, w	ord ptr ss:	[rsp+20]	
	010180004EE3 010180004EE8 010180004EC 010180004EF1 0010180004EF3 0010180004EF3	010180004EDE OF B7 010180004EE3 48 8B 010180004EE3 0F B6 010180004EE4 0F B6 010180004EE5 0F B6 010180004EE4 0F B6 010180004EE5 0F B6 010180004EE1 33 C1 0010180004EF3 0F B7 0010180004EF3 48 8B	010180004EDE OF B7 44 24 010180004EE3 48 88 42 24 010180004EE3 0F B6 04 01 010180004EE4 0F B6 04 01 010180004EE1 33 C1 010180004EF1 33 C1 0010180004EF3 0F B7 4C 24 0010180004EF3 48 85 54 24	010180004EDE 0F B7 44 24 20 010180004EE3 48 88 4C 24 40 0010180004EE3 0F 86 04 01 0010180004EE5 0F 86 4C 24 28 0010180004EE1 33 C1 0010180004EF3 0F 87 4C 24 20 0010180004EF1 33 C1 0010180004EF3 0F 87 4C 24 20	0010180004EDE OF B7 44 24 20 movzx eax, movzx 0010180004EE3 48 88 42 24 40 movzx eax, eax, b movzx eax, b movz eax,	0010180004EDE 0F B7 44 24 20 movzx eax,word ptr ss 0010180004EE3 48 88 42 40 mov rcx,qword ptr ss 0010180004EE3 0F 86 04 01 movzx eax,byte ptr ds 0010180004EEC 0F 86 04 01 movzx ecx,byte ptr ds 0010180004EE1 33 C1 xor eax,ecx xor eax,ecx 0010180004EF3 0F 87 42 20 movzx ecx,word ptr ss 0010180004EF3 36 42 48 movzx ecx,word ptr ss movzx ecx,word ptr ss 0010180004EF3 48 85 42 48 movzx ecx,word ptr ss	0010180004EDE 0F B7 44 24 20 movzx eax,word ptr ss: [rsp+20] 0010180004EE2 48 88 42 40 movzx eax,word ptr ss: [rsp+40] 0010180004EE2 0F B6 04 01 movzx eax,byte ptr ss: [rsp+40] 0010180004EE2 0F B6 4C 24 28 movzx eax,byte ptr ss: [rsp+28] 0010180004EF1 33 C1 Xor eax,ecx movzx ecx,word ptr ss: [rsp+20] 0010180004EF3 0F 87 4C 42 00 movzx ecx,word ptr ss: [rsp+20] 0010180004EF8 48 85 54 24 48 movzx ecx,word ptr ss: [rsp+40]

 0000010180006588 000001018000658C 000001018000658F 	49 88 0C DE 44 88 C5 49 88 D7	mov rcx,qword ptr ds:[r14+rbx°8] mov r8d,ebp mov rdx,r15		x87Tagword FFFF	
112 → 00000101800065C2	FF 15 C0 2D 00 00	<pre>call qword ptr ds:[<&StrCmpNIw>]</pre>	>		Cunlocke
qword ptr [0000010180009388 <&Str	CmpNIW>]= <sh1wapi.strc< td=""><td>mpNIW></td><td></td><td>1: rcx 00000000253CC70 L"C:\\Users\\ 2: rdx 00000000014FDC0 L"/LOGIN=" 3: r8 000000000000007</td><td>op\\malware.</td></sh1wapi.strc<>	mpNIW>		1: rcx 00000000253CC70 L"C:\\Users\\ 2: rdx 00000000014FDC0 L"/LOGIN=" 3: r8 000000000000007	op\\malware.

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 00000101800035F8 	33 D2 33 C9	xor edx,edx xor ecx,ecx		x87TagWord FFFF	
RIP	00000101800035FA	FF 15 30 5E 00 00	<pre>call qword ptr ds:[<&CoInitializeEx>]</pre>	> ×	Default (x64 fastcall)	▼ 5 ‡ 🗌 Unlocker
qword ptr	[0000010180009430 <&CoIr	itializeEx>]= <combase< td=""><td>.CoInitializeEx></td><td></td><td>1: rcx 00000000000000000000000000000000000</td><td></td></combase<>	.CoInitializeEx>		1: rcx 00000000000000000000000000000000000	

Figure 15

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

 0000010180003600 0000010180003606 0000010180003608 0000010180003610 	48 83 64 24 40 00 BF 03 00 00 00 83 64 24 38 00	and qword ptr ss:[rsp+40],0 mov ed1,3 and dword ptr ss:[rsp+38],0	RCX RDX RBP	00000000000000000000000000000000000000	
 0000010180003613 0000010180003619 0000010180003612 	45 33 C9 48 83 64 24 30 00 45 33 C0 89 7C 24 28	<pre>xor r9d,r9d and qword ptr ss:[rsp+30],0 xor r8d,r8d mov dword ptr ss:[rsp+28],edi</pre>	RSP RSI RDI	000000000014FDB0 00000000000000000 00000000000000000	"L ^A "
0000010180003620 0000010180003623 0000010180003623 0000010180003628	83 CA FF 83 64 24 20 00 33 C9 FF 15 F8 5D 00 00	<pre>or edx,FFFFFFFF and dword ptr ss:[rsp+20],0 xor ecx,ecx call gword ptr ds:[<&CoInitializeSecurity>]</pre>	R8 R9	000000000000000000000000000000000000000	

Figure 16

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

RIP 0000010180003630		call gword ptr ds:[
• 0000010180003636	89 05 8C 80 00 00	mov dword ptr ds: [1	018000B6C81.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 =

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:

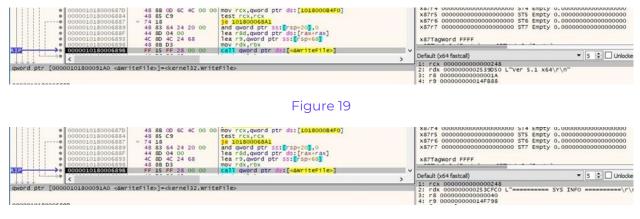


Figure 20

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

	• 0000010180003186	48 80 4C 24 40	lea rcx, gword ptr ss: rsp+40			
RIP	0000010180003188	FF 15 97 5F 00 00	call qword ptr ds:[<&GetSystemInfo>]		Default (x64 fastcall)	▼ 5 ¢ 🗌 Unlocke
qword pt	r [0000010180009128 <&Get5	SystemInfo>]= <kernel32< th=""><th>.GetSystemInfo></th><th>,</th><th>1: rcx 00000000014FA00 2: rdx 00000000000000000</th><th></th></kernel32<>	.GetSystemInfo>	,	1: rcx 00000000014FA00 2: rdx 00000000000000000	

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:

	<pre>00000101500031D8 00000101800031DF</pre>	48 80 40 80 89 50 80	lea rcx, qword ptr ss: rbp-50 mov dword ptr ss: rbp-50, ebx		x87TagWord FFFF	4.6
RIP	00000101800031E2	FF 15 20 62 00 00	call qword ptr ds:[<&RtlGetVersion>]	×	Default (x64 fastcall)	🕶 💈 🗖 Unlocke
qword p	tr [0000010180009408 <&Rt]G	etVersion>]= <ntdll.rt< td=""><td>1GetVersion></td><td></td><td>1: rcx 00000000014FA70 2: rdx 000000000000000</td><td></td></ntdll.rt<>	1GetVersion>		1: rcx 00000000014FA70 2: rdx 000000000000000	



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

0000010180003217 000001018000321A 00001018000321F 000001018000321F 0000010180003223	45 33 C9 48 8D 54 24 30 45 8D 41 0C 41 8D 49 01	<pre>xor r9d,r9d lea rdx,qword ptr ss: [rsp+30] lea r8d,qword ptr ds: [r9+C] lea ecx,qword ptr ds: [r9+1]</pre>	x87r7 0000000000000000000 ST7 Er x87TagWord FFFF	· · ·
0000010180003227 < qword ptr [0000010180009400 <&Rt](FF 15 D3 61 00 00 GetNativeSystemInforma	<pre>call qword ptr ds:[caRtIGetNativeSystemInformation> v</pre>	Default (x64 fastcal) 1: rCX 00000000000000 2: rdX 000000000014 F9F0 3: r8 0000000000000 4: r9 0000000000000	▼ S I Unlocke
		Figure 24		

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

• 00	 0000010180003258 48 80 95 F0 02 00 00 lea rdx,qword ptr ss: rbp+2F0 0000010180003256 48 80 20 D0 00 00 lea rcx,qword ptr ss: rbp+00 0000010180003266 89 05 P0 02 00 00 mov dword ptr ss: rbp+2F0 				x87Tagword FFFF		
• <	00001018000326C	erNamew>]= <advap132.getusern< th=""><th>word ptr ds:[<&GetUserNameW>]</th><th>></th><th>Default (x64 fastcal) 1: rcx 00000000014FB90 L"Ver %s x64\r\n" 2: rdx 00000000014FDB0</th><th>▼ 5 🗘 Unlock</th></advap132.getusern<>	word ptr ds:[<&GetUserNameW>]	>	Default (x64 fastcal) 1: rcx 00000000014FB90 L"Ver %s x64\r\n" 2: rdx 00000000014FDB0	▼ 5 🗘 Unlock	
			Figure 25				
	00001018000329A 0000101800032A1 0000101800032A7 00000101800032AE	48 SD SD DO 00 00 00 lea r	word ptr ss: rbp+2F01,ebx	~	x87Tagword FFFF Default (x64 fastcal)	 ▼ 5 ↓ Unlocks 	
gword ptr [000000	10180009118 <&GetCo	mputerNamew>]= <kernel32.getc< td=""><td>omputerNamew></td><td></td><td>1: rcx 00000000014FB90 L"</td><td></td></kernel32.getc<>	omputerNamew>		1: rcx 00000000014FB90 L"		

Figure 26

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

	 0000010180007714 0000010180007716 000001018000771A 000001018000771E 0000010180007721 0000010180007723 	33 D8 4C 8D 40 08 48 8D 50 10 89 58 08 33 C9 48 89 58 10	<pre>xor ebx,ebx lea rdx,qword ptr ds:[rax+8] lea rdx,qword ptr ds:[rax+10] mov dword ptr ds:[rax+1],ebx xor ecx,ecx mov gword gtr ds:[rax+1].rbx</pre>		x87F5 000000000000000000000000000000000000	Empty 0.0000000000000000000 Empty 0.00000000000000000000000000000000000
RIP	> 0000010180007727	E8 C1 B1 FF FF	call <netgetjoininformation></netgetjoininformation>	, v	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
<netgetjoi< td=""><td>nInformation></td><td></td><td></td><td></td><td>1: rcx 000000000000000 2: rdx 00000000014F9C8 3: r8 00000000014F9C0</td><td></td></netgetjoi<>	nInformation>				1: rcx 000000000000000 2: rdx 00000000014F9C8 3: r8 00000000014F9C0	

Figure 27

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

	 0000010180004AF5 0000010180004AF9 0000010180004AF8 	48 83 C9 FF 88 D7 4C 8D 44 24 40	or rcx,FFFFFFFFFFFFF mov edx,edi lea r8,qword ptr ss:[rsp+40]		x87Tagword FFFF	
RIP	> 0000010180004800	FF 15 3A 45 00 00	call gword ptr ds:[<@OpenProcessToken>]	```	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
qword pt	r [0000010180009040 <&Open	ProcessToken>]= <advap< td=""><td>132.0penProcessToken></td><td>,</td><td>1: rcx FFFFFFFFFFFFFF 2: rdx 000000000000008 3: r8 00000000014F580</td><td></td></advap<>	132.0penProcessToken>	,	1: rcx FFFFFFFFFFFFFF 2: rdx 000000000000008 3: r8 00000000014F580	

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:

	 000001018000488A 0000101800048C1 00000101800048C3 0000010180004802 0000010180004802 0000010180004807 0000010180004863 0000010180004853 0000010180004853 0000010180004853 0000010180004853 0000010180004853 0000010180004853 0000010180004853 0000010180004853 0000010180004853 	48 80 85 88 03 00 00 C7 85 88 03 00 00 48 89 44 24 28 4C 8D 44 24 50 48 80 85 50 01 00 00 88 F7 48 03 F6 48 89 44 24 20 33 C9	<pre>lea r9,qword ptr ss: [rbp+390] lea rax,qword ptr ss: [rbp+388] mov dword ptr ss: [rbp+388],100 mov qword ptr ss: [rbp+28],rax lea r8,qword ptr ss: [rbp+10] lea rax,qword ptr ss: [rbp+10] dd rsi;rsi mov qword ptr ss: [rbp+20],rax xor ecx,eck mov rdx,qword ptr ds: [rbx+rsi=848] call qword ptr ds: [rbx+rsi=848]</pre>		x87r1 000000000000000000 5T1 Er x87r2 00000000000000000 5T2 Er x87r3 000000000000000 5T3 Er x87r4 0000000000000000 5T4 Er x87r5 0000000000000000 5T5 Er x87r6 0000000000000000 5T5 Er x87r7 0000000000000000 5T6 Er x87r7 00000000000000000 5T7 Er	npty 0.00000000000000000000000000000000000
BIP	> 00000101800048F4	FF 15 26 44 00 00	[cal] gword ptr ds:[<&LookupAccountS1dw>]	2 1	Default (x64 fastcall)	▼ 5 🗢 🗆 Unlocke
gword ptr	[0000010180009020 <&Look	<pre>kupAccountSidw>]=<advapi< pre=""></advapi<></pre>	32.LookupAccountS1dw>		1: rCX 00000000000000000 2: rCX 00000000254F5E8 3: r8 000000000014F590 4: r9 000000000014F9D0	

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:

00000101800053C2 00000101800053C7 00000101800053CA	4C 8D 44 24 34 48 88 D3 33 C9	lea r8,qword ptr ss:[rsp+34] mov rdx,rbx xor ecx,ecx		x87Tagword FFFF	
81P 00000101800053CC	FF 15 5E 3C 00 00	call gword ptr ds:[<&LookupPrivilegeValueA>]		Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocker
qword ptr [0000010180009030 <&Loo	(upPrivilegeValueA>]=<	advapi32.LookupPrivilegeValueA>	ŕ	1: rcx 000000000000000 2: rdx 00000000014FCE0 "SeResto 3: r8 00000000014FCA4	prePrivilege"

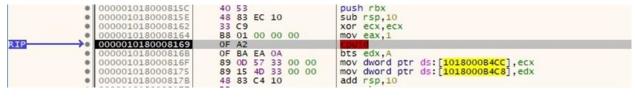


The binary enables the above privileges in the access token via a function call to AdjustTokenPrivileges:

 00000101800053DE 00000101800053E3 0000011800053E9 0000011800053E9 00000101800053E1 00000101800053F1 00000101800053F8 0000001080053F8 	4C 8D 44 24 30 lea r8,qword ptr 5s:[rsp+20] 48 83 64 24 20 00 and qword ptr ss:[rsp+20] 48 83 64 24 50 xor r9d,r9d 78 33 D2 xor r9d,r9d 74 24 30 01 00 00 mov dword ptr ss:[rsp+30] 74 24 30 01 00 00 mov dword ptr ss:[rsp+30] 74 24 30 02 00 mov dword ptr ss:[rsp+30]	,0 x87r5 000000000000000000000000000000000000
RIP 0000010180005403	FF 15 2F 3C 00 00 call qword ptr ds [<&Adju	istTokenPrivi]eges>]
qword ptr [0000010180009038 <&Adj	ustTokenPrivileges>]= <advapi32.adjusttokenprivile< td=""><td>ges> 2: r1X 000000000000000 3: r8 00000000014FCA0 4: r9 00000000000000</td></advapi32.adjusttokenprivile<>	ges> 2: r1X 000000000000000 3: r8 00000000014FCA0 4: r9 00000000000000

Figure 33

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





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	nov r9.rcx	
	000001018000886F		kor r8d,r8d	
RIP >> >>	0000010180008872	OF 31		
•	0000010180008874		shl rdx,20	
	0000010180008878	4C 8D 15 29 2C 00 00	lea r10, gword ptr ds: [1018000B4A8]	
	000001018000887F	48 0B C2	pr rax,rdx	
	0000010180008882	43 8B 14 10	nov edx, dword ptr ds: [r8+r10]	
	0000010180008886	FF C2	inc edx	
	0000010180008888	8B C8	nov ecx,eax	
	000001018000888A	83 E1 07	and ecx,7	
0	000001018000888D	D3 C2	rol edx,cl	
i •	000001018000888F	33 D0	kor edx,eax	
	0000010180008891	43 89 14 10	nov dword ptr ds:[r8+r10],edx	
	0000010180008895	43 89 14 01	nov dword ptr ds:[r9+r8],edx	
	0000010180008899	49 83 C0 04	add r8,4	
	000001018000889D		cmp r8,20	
L	00000101800088A1		ib 10180008872	



Address	He	x															ASCII
0000010180008580	CD	D3	85	CD	84	AB	F2	C1	01	A4	0B	39	A6	8F	ED	D7	10.1.«òA.¤.9¦.1x
0000010180008590	B3	A6		D9	5E	EG	FA	37	6F	3A	E4	84	AE	28	15	45	▪!.U^æú7o:ä′≋+.E

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

 0000010150005958 0000010180005961 0000010180005969 0000010180005969 0000010180005966 	4C 8D 45 D7 33 D2 48 8D 4D C7	<pre>mov r9d.1 0 mov dword ptr ss:[rsp+20],F0000000 1ea r8,qword ptr ss:[rbp-29] xor edx,edx 1ea rcx,qword ptr ss:[rbp-39]</pre>		x87r6 000000000000000000000000000000000000	mpty 0.00000000000000000000000000000000000
RIP 0000010180005973	FF 15 37 37 00 00	call qword ptr ds:[<&CryptAcquireContextW>]	Ň	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
qword ptr [0000010180009080 <&Cry)tAcquireContextW>]= <ad< td=""><td>ivapi32.CryptAcquireContextW></td><td></td><td>1: rcx 00000000014FC20 2: rdx 00000000000000 3: r8 000000000014FC30 L"Microsof 4: r9 00000000000000</td><td>t Enhanced Cryptographic Pr</td></ad<>	ivapi32.CryptAcquireContextW>		1: rcx 00000000014FC20 2: rdx 00000000000000 3: r8 000000000014FC30 L"Microsof 4: r9 00000000000000	t Enhanced Cryptographic Pr

Figure 37

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

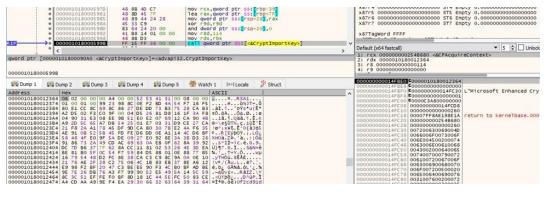


Figure 38

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

 0000010180005947 48 80 40 7F mov FCx, quord ptr ss: [rbp+77] 0000010180005947 48 00 46 77 radiation for st: [rbp+77] 0000010180005947 48 00 42 40 00 01 00 mov dword ptr ss: [rbp+77] 0000010180005984 48 94 42 42 8 0000010180005984 48 94 42 42 8 0000010180005984 48 97 72 42 00 0000010180005981 48 97 72 42 00 0000010180005981 48 97 72 42 00 0000010180005981 48 97 72 42 00 mov qword ptr ss: [rsp+20], rdt 0000010180005981 48 97 72 42 00 mov qword ptr ds: [rsp+20], rdt 0000010180005981 48 97 72 42 00 mov qword ptr ds: [rsp+20], rdt 0000010180005981 48 97 72 42 00 mov qword ptr ds: [rsp+20], rdt 	x87r5 00000000000000000 x87r6 000000000000000000000000000000000000	000 514 Empty 0.00000000000000000000000000000000055 Empty 0.00000000000000000000000000000000000
31P >> 00000101800059CA FF 15 C8 36 00 00 call gword ptr ds:[<&CryptEncrypt>]	Default (x64 fastcall)	▼ 5 🗢 🗌 Unloc
<pre>qword ptr [0000010180009098 <&CryptEncrypt>]=<advap132.cryptencrypt> 00000101800059CA</advap132.cryptencrypt></pre>	1: rcx 0000000025468A0 2: rdx 0000000000000000000 3: r8 0000000000000000 4: r9 000000000000000000	<&CPGenKey>
🗱 Dump 1 🗱 Dump 2 🗱 Dump 3 🗱 Dump 4 🗱 Dump 5 😻 Watch 1 🖾 Locals 🦻 Struct	00000000014FBE0 5000001	0180012364 0180012364
Address Hex ASCII 00000101800085A0 (20 D3 85 CD 84 AB F2 C1 01 A4 08 39 A6 8F ED D7 10,1+0A,=,9',ix 0000010180085A0 83 A5 L5 D9 EF EF EF 83 F3 84 AF 28 13 45 '' Usay707.5' # F	000000000014F8F0 000000 0000000014F8F8 000000	0000000114 0000000000 01800085A0

Address	He	x															ASCII
000001018000B5A0	9D	CF	E4	44	8F	8F	9F	3D	18	8A	27	15	E3	8E	BA	1D	.IaD='.à.°.
0000010180008580	A3	AE	D2	D3	C9	5D	34	31	94	DD	DO	BO	27	15	9D	7C	£000É]41.Ýа'
000001018000B5C0	4C	3F	56	F7	78	7C	03	98	8A	D9	59	F6	85	OB	B4	57	L?V÷x]UYöµ. W
																	V}T. . (451D00
000001018000B5E0	OD	18	DE	AE	61	9A	74	4D	83	5C	6A	83	BA	45	62	51	pea.tM. \j. °EbQ
000001018000B5F0	BB	FA	79	10	62	10	EO	47	8C	1A	54	75	D3	OF	32	58	»úy.b.aGTuÓ.2X
000001018000B600	7B	OB	9B	67	DD	72	5E	76	DB	BB	88	11	87	58	55	CF	{gýr^v0»XUI
000001018000B610	AF	83	2A	F3	66	47	4E	88	6A	BF	34	OD	81	85	9D	98	. * of GN. j24
000001018000B620	FB	C6	66	OD	4A	72	9D	BB	18	FB	0A	3E	DF	90	EC	AO	ûÆf.Jr.».ů.>B.ì
																	96v.hé*»ê.*^Gb.¶v
000001018000B640	F7	F6	FC	04	39	56	F7	C7	F7	BA	A4	AF	5E	17	49	4E	÷öü.9V÷C÷°¤^^.IN
000001018000B650	57	8D	97	OB	5B	37	48	C7	E1	FD	EA	06	8C	DE	C6	F1	W[7HCáýêÞÆñ
000001018000B660	69	76	AB	OE	B7	OD	17	1F	OD	D1	F5	FD	DA	43	AS	68	iv«Noýúc h
000001018000B670	B3	03	61	28	1F	32	9E	3F	32	F8	10	26	B1	21	05	A5	*.a(.2.720.&±1.¥
000001018000B680	9A	F4	19	21	1D	BB	4B	30	08	CB	2D	2C	4A	OC	EB	7B	.ô.!.»KO.E-,J.ë{
0000010180008690	A7	C4	1F	4F	95	76	1F	14	OC	63	79	8D	41	CC	66	A5	§A.O.VCV.AIf¥

Figure 40

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

	<pre>0000010180003E16 0000010180003E1A 0000010180003E21</pre>	48 89 0D 9F 78 00 00 m 48 8D 4D C0 1	ea rdx,qword ptr ss: rbp+10 ov gword ptr ds: [101800086C0],rcx ea rcx,gword ptr ss: rbp-40		x87TagWord FFFF	
RIP	>> 0000010180003E25 <	FF 15 95 54 00 00	all qword ptr'ds:[<&GetComputerNameA>]	>	Default (x64 fastcall)	▼ 5 C Unlocke
qword pt	r [00000101800092C0 <&GetC	ComputerNameA>]= <kernel32.< td=""><td>.GetComputerNameA></td><td></td><td>1: rcx 00000000014FC70 L" Provider v1.0" 2: rdx 00000000014FCC0</td><td></td></kernel32.<>	.GetComputerNameA>		1: rcx 00000000014FC70 L" Provider v1.0" 2: rdx 00000000014FCC0	
Address 00000000	Hex 0014FC70 74 D3 98 60 D8 87	7 A9 70 A3 81 CB 62 31 D9	ASCII 8E 20 to. 0.mpf.Eb10.			

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:

	 0000010180003E91 0000010180003E98 	48 8D 55 E0 lea rdx	,qword ptr ds:[1018000B6B8] ,qword ptr ss:[rbp-20]		x87TagWord FFFF	
RIP		FF 15 F6 54 00 00 call qw	ord ptr ds:[<&StrStrIA>]	, *	Default (x64 fastcall)	👻 5 🗘 🗌 Unlock
qword ptr		StrIA>]= <sh]wapi.strstria></sh]wapi.strstria>			1: rCX 000001018001249C 2: rdX 00000000014FC90 "%CLIENT_1 3: r8 000000000000000 4: r9 00000000014FC36	ID%"
Dump 1	Ump 2 Ump 3	📖 Dump 4 🛛 🗱 Dump 5 🛛 🛞 Watd			00000000014EC50 C000001018000EC16 00000000014EC58 C0000000014EC90	
0000010180 0000010180 0000010180 0000010180 0000010180 0000010180	00124AC 0A 09 09 3C 74 6 00124BC 60 3C 2F 74 69 7 00124CC 61 64 3E 00 0A 0 00124DC 3C 68 31 3E 59 6 00124EC 3E 0D 0A 09 09 33 00124EC 65 3E 0D 0A 20 2	E 00 0A 09 3C 68 65 61 64 3E 00 9 74 6C 65 3E 51 75 61 6E 74 75 4 6C 65 3E 00 0A 09 3C 2F 68 65 9 3C 62 6F 64 79 3E 00 0A 09 3C 2F 68 65 9 3C 62 6F 64 79 3E 00 0A 09 39 70 7 57 72 20 49 44 3A 3C 2F 68 31 C 62 8E 00 A9 99 09 3C 70 72 0 20 20 20 20 92 02 00 00 A 25 4 5F 49 44 25 20 20 10 20 20 20 20	<tile>Quantu m</tile> ad> body> <hl>Your ID:> e> e> %</br></hl>	Î	000000000147661 00000000000000000000000000000000000	

Address	He	ĸ															ASCII
000001018001250B	30	66	32	63	64	39	31	64	34	31	36	33	35	31	30	33	Of2cd91d41635103
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	
000001018001255B	0D	0A	09	09	3C	2F	62	3E	OD	0A	09	09	3C	68	72	2F	<hr <="" td=""/>
000001018001256B	3E	0D	0A	54	68	69	73	20	GD	65	73	73	61	67	65	20	>This message
000001018001257B	63	6F	6E	74		69				61					66	6F	contains an info
000001018001258B	72	6D		74				20		6F				6F	20	66	rmation how to f
000001018001259B	69	78								6F		62	6C	65	73	20	ix the troubles
00000101800125AB	79	6F	75	27						74	20	77	69	74	68	20	you've got with
00000101800125BB	79	6F				6E								3C			your network. <br< td=""></br<>
00000101800125CB	3E	3C	62	72	3E		0A			46							> Files o
00000101800125DB	6E	20	74	68	65	20	77	6F	72	6B	73	74	61	74	69	6F	n the workstatio
00000101800125EB		73	20	69	GE					72		6E		74		6F	ns in your netwo
00000101800125FB	72	6B	20		65	72		20		6E						65	rk were encrypte
000001018001260B		20	61		64	20		6E		20					20	61	d and any your a
000001018001261B		74	65							20			61			65	ttempt to change
000001018001262B	2C	20	64	65	63	72	79	70	74	20	6F	72	20	72	65	6E	, decrypt or ren

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:

	s:[rsp+30] ds:[10180010658] rsp+20],eax	x87r5 00000 x87r6 00000 x87r7 00000 x87r7 00000 x87r7 00000 x87ragword Default (x64 fast 1 rcx 0000 2: rdx 0000 3: r8 00000	ttal)
U Dump 1 U Dump 2 U Dump 3 U Dump 4 U Dump 5 🛞 Watch 1 🗠 Loca	als 🐉 Struct	00000000014	4F880 = 0000010180000D06 4F888 = 000000000014F880 L"explorer.exe README_TO
Address Hex ASCII		00000000014	4F890 C00000000014FD20 L".quantum" 4F898 C000000002534C60
000000000014FC40 5C 00 43 00 6C 00 61 00 73 00 73 00 65 00 73 00 \.C.1.a.s. 0000000000004FC50 5C 00 28 00 71 00 75 00 61 00 66 00 74 00 75 00 \ 00000000000004FC50 6D 00 5C 00 73 00 68 00 66 00 6C 00 5C 00 m.\.s.h. 000000000004FC50 6D 00 5C 00 73 00 68 00 65 00 6C 00 5C 00 m.\.s.h. 0000000000014FC50 6D 00 61 00 6E 00 5G 00 50 00 00 00 00 00 00 m.a.n.d.	n.t.u. 1.1.\. c.o.m.	00000000014	4F8A5 E000000000000002 4F8B6 E00020070078005005 4F8B6 E007200650072006F 4F8C0 E006500780050002E 4F8C9 E0041004500530030
📑 Registry Editor			– 🗆 ×
File Edit View Favorites Help			
Computer\HKEY_CURRENT_USER\Software\Classes\.quantum\shell\Open\command			
> .ms-lockscreencomponent-primary ^ Name	Тур	e	Data
>mts		S_SZ	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" "ps]" "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**):

00000101\$0004859 0000010180004858 0000010180004858 0000010180004850	33 D2 33 C9 41 B8 3F 00 0F 00	xor edx,edx xor ecx,ecx mov r8d,F003F		x87Tagword FFFF	
RIP 0000010180004863	FF 15 FF 47 00 00	call gword ptr ds:[<&OpenSCManagerA>]	. ~	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
qword ptr [0000010180009068 <&Open	SCManagerA>]= <advap13< td=""><td>2.OpenSCManagerA></td><td>,</td><td>1: rcx 0000000000000000 2: rdx 00000000000000000000000000000000000</td><td></td></advap13<>	2.OpenSCManagerA>	,	1: rcx 0000000000000000 2: rdx 00000000000000000000000000000000000	

Figure 46

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

00000111800048A1 00000101800048A4 00000101800048A9 00000101800048A9 00000101800048A7 00000101800048A7 0000010180004857 0000010180004857 0000010180004555 0000010180004555 0000010180004555 0000010180004555	40 88 CE mov rg.ri4 48 89 44 24 38 68 62 mov qword ptr ds: [rbx+30] 68 62 64 24 68 94 24 80 00 00 68 62 42 80 00 00 68 62 42 80 00 00 68 62 42 80 00 00 64 80 43 01 1ea at.gword ptr ss: [rsp=90] 68 62 42 42 80 00 1ea rat.gword ptr ss: [rsp=91] 68 89 42 28 mov qword ptr ss: [rsp=20] rax 68 84 24 20 00 00 1ea rat.gword ptr ss: [rsp=20] 7 42 42 00 00 1pr ss: [rsp=20] rax	x87r2 000000000000000000000000000000000000
RIP 00000101800048D5	FF 15 95 47 00 00 call gword ptr ds [<&EnumServicesStatus	Default (x64 fastcall) 🔹 5 🗘 🗌 Unlocke
qword ptr [0000010180009070 <&Enu	mServicesStatusA>]= <advapi32.enumservicesstatusa></advapi32.enumservicesstatusa>	11 rcx 00000000254CA40 22 rdx 0000000000000 31 r8 0000000000001 42 r9 0000000000001

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**):



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

0000010180004515 0000010180004515 0000010180004515 000001018000451F	4C 8D 44 24 30 8B D3 48 8B CF	lea r6.qword ptr ss:[rsp+30] mov edx.ebx mov rcx.rdi		x87Tagword FFFF	- Andrea
312 → 0000010180004522 <	FF 15 60 48 00 00	call qword ptr ds:[<&ControlService>]	>	Default (x64 fastcall)	👻 S 💠 🗌 Unlocke
qword ptr [0000010180009088 <&Con	trolService>]= <advapi3< td=""><td>2.ControlService></td><td></td><td>1: rcx 00000000254C560 2: rdx 000000000000000 3: r8 00000000014FC20</td><td></td></advapi3<>	2.ControlService>		1: rcx 00000000254C560 2: rdx 000000000000000 3: r8 00000000014FC20	
· · · · · ·				3. 18 00000000000000000000000000000000000	

Figure 50

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

000001018000454C 0000010180004552 0000010180004552 0000010180004557 000001018000455E 000001018000455E	41 B9 24 00 00 00 4C 8D 44 24 50 48 89 44 24 20 33 D2 48 8B CF	<pre>mov r9d;24 lea r8,qword ptr ss:[rsp+50] mov qword ptr ss:[rsp+20],rax xor edx,edx mov rcx,rdi</pre>		x87r6 000000000000000000 ST6 Empty 0.1 x87r7 00000000000000000 ST7 Empty 0.1 x87Tagword FFFF	
RIP 0000010180004561	FF 15 E9 4A 00 00	call qword ptr ds:[<&QueryServiceStatusEx>]	×	Default (x64 fastcall)	🔻 💈 🖨 Unlocke
qword ptr [0000010180009050 <&Query:	ServiceStatusEx>]= <a< td=""><td>dvapi32.QueryServiceStatusEx></td><td></td><td>1: rcx 00000000254C560 2: rdx 00000000000000 3: r8 00000000014FC40 &"SQLWriter" 4: r9 0000000000024</td><td></td></a<>	dvapi32.QueryServiceStatusEx>		1: rcx 00000000254C560 2: rdx 00000000000000 3: r8 00000000014FC40 &"SQLWriter" 4: r9 0000000000024	

Figure 51

Killing targeted processes

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

000001015000669D 00000101800066A5 00000101800066A5 00000101800066A8	44 88 84 24 80 01 00 mov r8d,dword ptr ss:[rsp+180] 43 88 D0 mov rdx,rax a 44 80 82 0 01 00 a r9,qword ptr ss:[rsp+180] 59 05 00 00 mov ecx,s mov ecx,s mov ecx,s mov ecx,s	x87r7 00000000000000000000000000000000000
81P 0000010180006685	ES D3 C1 FF FF call <rtlgetnativesysteminformation></rtlgetnativesysteminformation>	on> V Default (x64 fastcall) V 5 🗘 🗌 Unio
<pre>«RtlGetNativeSystemInformation></pre>		1 r cr 00000000000000 2 r dx 0000000253760 3 r c8 000000000147C8 4 r c9 000000000147C6



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" "procemon64.exe" "procexp.exe"
 "procexp64.exe"

	0000010180006EA5 0000010180006EAA	48 80 54 24 20 48 88 CF	lea rdx,qword ptr ss:[rsp+20] mov rcx,rdi		x87Tagword FFFF	
RIP	0000010150006EAD	FF 15 25 24 00 00	call qword ptr ds:[<&lstrcmpiA>]	×	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
qword pt	r [00000101800092D8 <&lstr	cmpiA>]= <kernel32.lst< td=""><td>rcmpiA></td><td>,</td><td>1: rcx 00000000014FC20 "smss.exe" 2: rdx 00000000014FBB0 "msftesql.e</td><td>exe"</td></kernel32.lst<>	rcmpiA>	,	1: rcx 00000000014FC20 "smss.exe" 2: rdx 00000000014FBB0 "msftesql.e	exe"

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

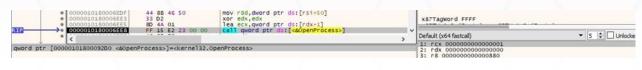


Figure 54

The process is killed by calling the TerminateProcess API:

0000010180006 0000010180006	5EF6 33 D2 5EF8 48 88 C8	xor edx,edx mov rcx,rax		x87Tagword FFFF	
RIP → 0000010180000	SEFE FF 15 C7 23 00 00	<pre>call qword ptr ds:[<&TerminateProcess>]</pre>	>	Default (x64 fastcall)	▼ S 😂 🗌 Unlocke
qword ptr [00000101800092C8	<&TerminateProcess>]= <kerne< td=""><td>132.TerminateProcess></td><td></td><td>1: rcx 000000000000270 2: rdx 000000000000000</td><td></td></kerne<>	132.TerminateProcess>		1: rcx 000000000000270 2: rdx 000000000000000	

Figure 55

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

1	Ver 5.1 x64
	SYS INFO
	CORE COUNT: 1
	TOTAL MEM: 2047 MB
	WIN VER: 10.0.16299 SP0
	WIN ARCH: x64
	USER NAME:
	PC NAME: DESKTOP-
	IN DOMAIN: NO
	IS ADMIN: YES
	IN GROUPS:
	Mandatory DESKTOP-
	Mandatory \Everyone
	Mandatory NT AUTHORITY\Local account and member of Administrators group
	Mandatory BUILTIN\Administrators
	Mandatory BUILTIN\Users
	Mandatory BUILTIN\Performance Log Users
	Mandatory NT AUTHORITY\INTERACTIVE
	Mandatory \CONSOLE LOGON
	Mandatory NT AUTHORITY\Authenticated Users
	Mandatory NT AUTHORITY\This Organization
	Mandatory NT AUTHORITY\Local account
	Mandatory NT AUTHORITY\LogonSessionId_0_130301
	Mandatory \LOCAL
	Mandatory NT AUTHORITY\NTLM Authentication
	Integrity Mandatory Label\High Mandatory Level
	CMDLINE: "C:\Users\\Desktop\malware.exe"
	KILL SERVICE
	SQLWriter ok

	KILL PROCESS
	Procmon.exe ok
	Procmon64.exe ok



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**):

RIP	 0000010180004072 0000010180004074 0000010180004077 0000010180004077 	8B F2 48 8B F9 FF 15 13 52 00 00 83 F8 04	<pre>mov esi,edx mov rdi,rcx call qword ptr ds:[<&GetDriveTypew>] cmp eax,4</pre>	_	x87r7 0000000000000000000 ST7 Empty 0.0 x87Tagword FFFF	000000000000000000000000000000000000000
	0000010180004080	* 74 3D	je 101800040BF	×	Default (x64 fastcall)	▼ 5 C Unlocke

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:

 0000010180004F9A 46 83 64 24 26 00 and qword ptr ss:["ssp-28],0 0000010180004FA7 83 64 24 20 00 and qword ptr ds:["ssp-28],0 and qword ptr ss:["ssp-28],0 and qword ptr ssin[["ssp-28],0 and qword ptr ssin[["ssp-28],0 and qword ptr ssin[["ssp-28],0 and qword ptr ssin	x87r7 0000000000000000000 ST7 Empty 0.00000000000000000000000000000000000
310 >>> 0000010180001838 FF 15 5F 42 00 call qword ptr ds:[<&createThread	d>) Default (x64 fastcall)
🗱 Dump 1 🗱 Dump 2 🗱 Dump 3 🗱 Dump 4 👯 Dump 5 👹 Watch 1 💷 Locals 🎾 Struct	00000000014FC70 F0000000014FD20 L"\\\\?\\c:\\" 00000000014FC78 F0000000014FD30
Address Hex ASCII 000000000253C0A0 SC 00 5C 0015F 00 5C 00163 00 3A 00100 00 00 00 00 00 00 00 00 00 00 00	000000000014FC80 F000000000001 00000000014FC80 F0000000000000 0000000014FC80 F0000000000000 0000000014FC90 F0000000000000

Figure 59

Thread activity – sub_10180008CA0 function

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





An unnamed semaphore object is also created by the malware:





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**):



WNetEnumResourceW is utilized to continue the enumeration of network resources:

0000010180007395 000001018000739A 000001018000739F 0000010180007342	48 8B 4C 24 38 4C 8D 4C 24 34 4C 8B C0 48 8D 54 24 30	<pre>mov rcx,qword ptr ss:[rsp+38] lea r9,qword ptr ss:[rsp+34] mov r8,rax lea rdx,qword ptr ss:[rsp+30]</pre>		x87r7 0000000000000000000 ST7 Emp x87Tagword FFFF	
RIP → 00000101800073A7	E8 F9 84 FF FF	call <wnetenumresourcew></wnetenumresourcew>	, ×	Default (x64 fastcall)	🔻 💈 📮 Unlocke
<pre><wnetenumresourcew></wnetenumresourcew></pre>				1: rcx 000000002556100 2: rdx 00000000487FED0 3: r8 0000000025563F0 4: r9 00000000487FED4	

Figure 64

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

	00010180004600 00010180004604 00010180004608 00010180004608 00010180004611 00010180004616 00010180004616 00010180004628 00010180004628 00010180004628	83 65 7F 00 48 80 45 67 48 80 45 67 48 83 65 6F 00 4C 80 45 6F 48 89 44 24 30 41 83 C9 FF 48 80 45 6B 8A 01 00 00 48 89 44 24 28 49 88 CF 48 80 45 7F 48 80 45 7F	<pre>and dword ptr ss[rbp+7F] 0 lea rax, gword ptr ss[rbp-13] and gword ptr ss[rbp-13], 0 lea r3, gword ptr ss[rbp-13], 0 lea r3, gword ptr ss[rbp-14], rax or rdd,FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</pre>	Pexi	x87r0 000000000000000000000000000000000000	00000000000000000000000000000000000000
	00010180004634	E8 AE E2 FF FF	call <netshareenum></netshareenum>	×	Default (x64 fastcall)	💌 S 💠 🗌 Unlocke
<netshareenum></netshareenum>				,	1: rcx 000000002568D50 L"DESKTOP- 2: rdx 000000000000001 3: r8 00000000487FD38 4: r9 00000000FFFFFFF	-

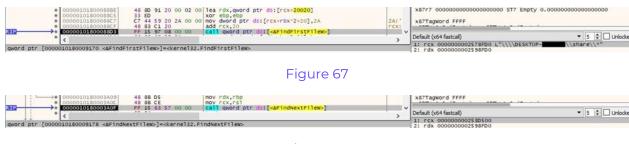
Figure 65

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

00000101800069F8 48 80 54 24 20 00000101800069F0 48 88 C8 00000101800069F0 FF 15 72 29 00 00	<pre>lea rdx,qword ptr ss:[rsp+20] mov rcx,rbx call qword ptr ds:[<&StrStrIw>]</pre>	rex:	x87TagWord FFFF
	ferri due a bei ant fease sei turi		Default (x64 fastcall) 🔻 5 🗘 🗌 Unlock
qword ptr [0000010180009378 <&StrStrIw>]= <sh]wapi.strst 0000010180006A00</sh]wapi.strst 	1105		2: rdx 000000000487FCA0 3: r8 0000000000000007 4: r9 00000000000000020
🖓 Dump 1 👹 Dump 2 👹 Dump 3 👹 Dump 4 👹 Dump 5	👹 Watch 1 🛛 🗱 Locals 🎾 Struct		00000000487FC80 F0000010180000EA6 00000000487FC88 F00000000487FCA0
Address Hex	ASCII	^	000000000487FC90 00000000487FCF8 &L"DE5KT0P-
000000000487FCA0 5C 00 41 00 44 00 4D 00 49 00 4E 00 24	00 00 00 N.A.D.M.I.N.\$		

Figure 66

The files are enumerated using the FindFirstFileW and FindNextFileW functions:





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

	000001018000578C 0000010180005791	48 8D 54 24 20 48 8B CB	lea rdx,qword ptr ss:[rsp+20] mov rcx,rbx	rex:	x87TagWord FFFF	
(P	0000010180005794	FF 15 A6 3A 00 00	call qword ptr ds:[<&lstrcmpiw>]		Default (x64 fastcall)	▼ 5 🗘 🗆 Unlocke
word ptr [0	000010180009240 <&lstrc	mpiW>]= <kernel32.lstr< th=""><th>cmp1W></th><th></th><th>1: rcx 000000002598FFC L"bla.tx 2: rdx 000000000487FA70 L"README</th><th>t"TODECRYPT.html"</th></kernel32.lstr<>	cmp1W>		1: rcx 000000002598FFC L"bla.tx 2: rdx 000000000487FA70 L"README	t"TODECRYPT.html"

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

0000010180006F72 0000010180006F76	48 88 14 01 48 80 4C 24 20	mov rdx, qword ptr ds: [rcx+rax] lea rcx, qword ptr ss: [rsp+20]	rdx:	x87TagWord FFFF	
P → 0000010180006F78	FF 15 FF 23 00 00	call qword ptr ds:[<&StrCmpIW>]		Default (x64 fastcall)	▼ S C Unlocke
word ptr [0000010180009380 <4StrCm	pIW>]= <shlwapi.strcmp< td=""><td>IW></td><td></td><td>1: rcx 00000000487FA00 L"txt" 2: rdx 000000002546D40 L"exe"</td><td>White Schline Schedenb</td></shlwapi.strcmp<>	IW>		1: rcx 00000000487FA00 L"txt" 2: rdx 000000002546D40 L"exe"	White Schline Schedenb

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.

• 00000101800034D1	48 8B 4B 20	mov rcx, qword ptr ds: [rbx+20] call gword ptr ds: [<&SetEvent>]			
	FF 15 1D 5D 00 00	call gword ptr ds:[<&SetEvent>]	· · ·	Default (x64 fastcall)	▼ 5 C Unlocke
gword ptr [00000101800091F8 <&SetEv	ent>]= <kernel32.seteve< td=""><td>ent></td><td></td><td>1: rcx 000000000000274</td><td></td></kernel32.seteve<>	ent>		1: rcx 000000000000274	

Figure 71

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

00000101800 00000101800 00000101800 00000101800 00000101800	5588 BA 00 00 00 40 mov edx,40000000 5580 45 33 C9 xor r9d,r9d	x87r7 000000000000000000 ST7 Empty 0.00000000000000000000000000000000000
RTP 00000101800	ESED FF 15 02 3C 00 00 call gword ptr ds: [<&CreateFile	v Default (x64 fastcall) v 5 ♀ □ Union
qword ptr [0000010180009198	<&CreateFilew>]= <kernel32.createfilew></kernel32.createfilew>	1: rcx.0000000002565F8L.////DESKTOP- 2: rdx.00000004000000 3: r8.00000004000000 4: r8.00000000000000

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:

0000010180006889 0000010180006889 0000010180006880 0000010180006820	48 88 49 18 45 33 CO 41 88 D1	mov rcx,qword ptr ds:[rcx+18] xor r8d,r8d mov edx,r9d		x87Tagword FFFF	
81P 0000010180006BC3	FF 15 37 26 00 00	<pre>call qword ptr ds:[<dreleasesemaphore>]</dreleasesemaphore></pre>	```	Default (x64 fastcall)	▼ 5 ♥ Unlocke
qword ptr [0000010180009200 <&Rele	easeSemaphore>]= <kerne< td=""><td>132.ReleaseSemaphore></td><td>,</td><td>1: rcx 0000000000002A0 2: rdx 00000000000001 3: r8 00000000000000</td><td></td></kerne<>	132.ReleaseSemaphore>	,	1: rcx 0000000000002A0 2: rdx 00000000000001 3: r8 00000000000000	

Figure 74

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

000010180006404 000010180006409 000010180006409 000010180006405 000010180006450 0000010180006450 0000010180006450 0000010180006450 0000010180006450 0000010180006450 0000010180006450	48 83 60 E8 00 and qword ptr ds: [rax-18],0 88 80 E0 00 and dword ptr ds: [rax-20],0 88 68 E9 mov rds, rcx 48 88 E8 F9 mov rds, rcx mov rds, rcx F4 54 50 00 00 64 53 C6 10 74 50 03 00 00 75 10 50 03 00 00 74 53 C0 00	X8772 00000000000000000000000000000000000
RIF → 00000101800064F8	FF 15 9A 2C 00 00 call gword ptr ds:[<&CreateFilew>]	V Default (x64 fastcall) V 5 0 Unloc
qword ptr [0000010180009198 <&Cre	iteFilew>]= <kernel32.createfilew></kernel32.createfilew>	1: rcx 00000000025C620 L"\\\DESKTOP- 2: rdx 00000000000000 3: r8 00000000000000 4: r9 00000000000000

The size of the file is obtained using GetFileSizeEx:

	> 000001018000652F 0000010180006533	48 8D 57 08 48 88 C8	lea rdx,qword ptr ds:[rdi+6] mov rcx,rax		x87TagWord FFFF	40
RIP	> 0000010180006536 <	FF 15 1C 2D 00 00	call qword ptr ds:[<&GetFileSizeEx>]	~	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
qword p	tr [0000010180009258 <&GetF	ileSizeEx>]= <kernel32< td=""><td>.GetFileSizeEx></td><td></td><td>1: rcx 0000000000002F0 2: rdx 000000004C81048</td><td></td></kernel32<>	.GetFileSizeEx>		1: rcx 0000000000002F0 2: rdx 000000004C81048	

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:

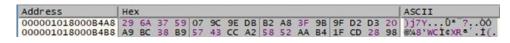


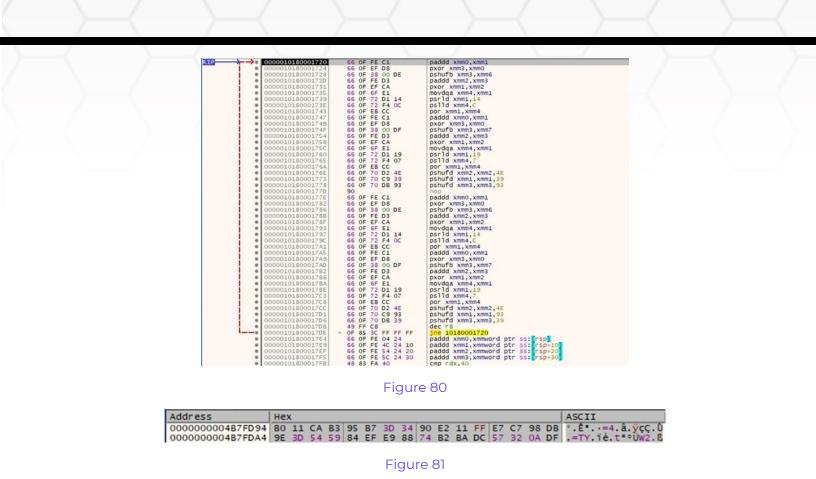
Figure 78

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

Address	He	x															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	OB	39	A6	8F	ED	D7	10.1. «òA.¤.9!.ix
0000000004B7FD00	B3	AG	16	D9	5E	E6	FA	37	6F	3A	E4	B4	AE	2B	15	45	'. UAæú70: ä´@+. E
0000000004B7FD10	CD	D3	85	CD	84	AB	F2	C1	01	A4	OB	39	AG	8F	ED	D7	10.1.«òA.¤.9'.i>

Figure 79

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



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

 00000101\$0003AE\$ 0000010180003AE\$ 0000010180003AF1 0000010180003AF4 	48 88 08 41 89 02 00 00 00 45 33 C0 33 D2	mov rcx,qword ptr ds:[rbx] mov r9d,2 xor r8d,r8d xor edx,edx		XMM9 00000000000000000000000000000000000	00000
RIP 0000010180003AF6	FF 15 7C 57 00 00	<pre>call qword ptr ds:[<&SetFilePointerEx>]</pre>	Ň	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
qword ptr [0000010180009278 <&Set	FilePointerEx>]= <kerne< td=""><td>132.SetFilePointerEx></td><td></td><td>1: rcx 000000000000000000 2: rdx 00000000000000 3: r8 000000000000000 4: r9 000000000000000</td><td></td></kerne<>	132.SetFilePointerEx>		1: rcx 000000000000000000 2: rdx 00000000000000 3: r8 000000000000000 4: r9 000000000000000	

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:

<pre></pre>	X8/F6 000000000000000000000000000000000000
212 0000010180003B1C FF 15 7E 56 00 00 call gword ptr ds:[<&writeFile>]	Default (x64 fastcall) 🔹 5 🗘 🗌 U
<pre>qword ptr [0000001800091A0 <dwritefile>]=<kernel32.writefile> 0000010180003B1C</kernel32.writefile></dwritefile></pre>	1: rcx 0000000000000000 2: rdx 00000000487F080 3: r8 00000000000139 4: r9 000000000487FED0
🗱 Dump 1 🗱 Dump 2 🗱 Dump 3 🗱 Dump 4 🗱 Dump 5 🥮 Watch 1 🛛 🕸 Locals 🍃 Struct	00000000487F050 000000004C81040 00000000487F058 000000000000000
Address Hex Asclii 000000000437FD30 00	 00000000487F065 000000000000000000000000000000000000

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

000001018000715E 48 88 0E 0000010180007161 4C 80 4C 2 0000010180007166 48 36 42 000001018000716C 45 88 C6 000001018000716C 49 88 D7	4 20 00 and gword ptr ss:[rsp+20],0 mov rSd,r14d mov rdx,r15		x87F5 000000000000000000000000000000000000	mpty 0.000000000000000000000
SIP 0000010180007172 FF 15 E8 2	0 00 00 call gword ptr ds:[<&ReadFile>]	, ×	Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
qword ptr [0000010180009260 <&ReadFile>]= <kern< td=""><td>132.ReadFile></td><td></td><td>1: rcx 00000000000002F0 2: rdx 00000004C81078 3: r8 00000000000001E 4: r9 000000000487FED8</td><td></td></kern<>	132.ReadFile>		1: rcx 00000000000002F0 2: rdx 00000004C81078 3: r8 00000000000001E 4: r9 000000000487FED8	

Figure 84

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

Address	He	x															ASCII
0000000004B7FE00	65	78	70	61	6E	64	20	33	32	2D	62	79	74	65	20	68	expand 32-byte k
0000000004B7FE10	29	6A	37	59	07	90	9E	DB	82	A8	3F	9B	9F	D2	D3	20) j7Y0= 700
0000000004B7FE20	A9	BC	38	B9	57	43	CC	A2	58	52	AA	B4	1F	CD	28	98	@%8'WCI¢XRª .1(.
0000000004B7FE30	29	6A	37	59	07	90	9E	DB	82	AS	3F	9B	9F	D2	D3	20) 17Y 0= 7 00

Figure 85

0000010180007183 0000010180007183 0000010180007185 0000010180007185 00000101800071c3	4C 8D 4C 24 68 48 88 0E 49 88 D7 48 83 64 24 20 00		x8/r5_000000000000000000000000000000000000		
RIP 00000101800071C9	FF 15 D1 1F 00 00	and gword ptr ss: [rsp+20],0 call gword ptr ds: [<&writeFile>]	×	Default (x64 fastcall) • 5 🗘 🗌 Unlock	
qword ptr [00000101800091A0 <≀	iteFile>]= <kernel32.wri< th=""><th>teFile></th><th></th><th>1: rcx 000000000000002F0 2: rdx 000000004581078 3: r8 000000000001E 4: r9 000000000487FED8</th></kernel32.wri<>	teFile>		1: rcx 000000000000002F0 2: rdx 000000004581078 3: r8 000000000001E 4: r9 000000000487FED8	
1 Dump 1 1 Dump 2 Dump 3	Dump 4 👹 Dump 5	👹 Watch 1 🛛 🕼 🖉 Struct		00000000487FE70 0000000000000000000000000000000000	
Address Hex 0000000004C81078 GC AA 47 AB 61 0000000004C81088 43 AE 1D 84 43	8A C5 8B 02 28 85 78 00 78 27 85 62 D8 9E 87 9E	ASCII 09 1A FC [*G«a.Å(.{ü 08 00 00 C*Cx'.b0	^	00000000487FE80 DA2009EFEC0EB00E 000000000487FE80 00000000000001E 000000000487FE80 00000000000000	

Figure 86

We continue with the analysis of the main thread.

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

 0000010150007C8 0000010150007C95 	48 80 95 80 00 00 00 lea rdx,qword ptr ss:[rbp+80] B9 04 01 00 00 mov ecx,104	x87Tagword FFFF
31P 0000010180007C94	FF 15 98 14 00 00 call qword ptr ds: [<&GetTempPathws	Default (x64 fastcall)
gword ptr [0000010180009138 <&G	etTempPathw>]= <kernel32.gettemppathw></kernel32.gettemppathw>	1: rcx 00000000000000004 2: rdx 00000000014F980

Figure 87

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



<pre> 000001018000556C 48 83 60 E8 00 mov rdi,r8 mov rdi,r9 m</pre>	Xarra 04040000000000000000000000000000000
3FG →• OCCONTREMENDESSO FF 15 02 3C 00 00 call qword ptr ds:[<screatefilew>] qword ptr (0000010180005190 <6CreateFilew>]=<kernel32.createfilew> 0000010180005190</kernel32.createfilew></screatefilew>	Default (x64 fastcal) Default (x64 fastcal) I1 rcx 00000000014F980 L"C:\\Users\\\\\AppData\\Local\\ 2: rdx 000000000000000 3: r8 0000000000000 3: r9 0000000000000
🗱 Dump 1 🗱 Dump 2 🗱 Dump 3 🗱 Dump 4 🗱 Dump 5 👹 Watch 1 💷 Locals 🖉 Struct	00000000014F750 50000000000000000000000000000000000
Address Hex ASCII 0000000000014F980 43 00 34 00 55 00 73 00 65 00 73 00 5 00 73 00 5 00 73 00 5 00 73 00 5 00 73 00 5 00 73 00 5 00 73 00 5 00 73 00 5 00 74 00 5 00 73 00 5 00 74 00 5 00 74 00 5 00 74 00 5 00 74 00 5 00 74 00 5 00 74 00 5 00 74 00 5 00 74 00 5 00 74 00 5 00 74 00 5 00 74 00 5 00 74 00 7	00000000014F7C0 00000000014F300 00000000014F30 00000000014F70 000000000000000 00000000014F70 000000000000000 00000000014F7E 0000000000000000 00000000014F7E 000000000000000000 00000000014F7E 0000000000000000000 00000000014F7E 000000000000000000000 00000000

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

000001180005554 0000010180005545 0000010180005545 0000010180005544 0000010180005580 0000010180005580 0000010180005580 0000010180005580 0000010180005580	48 83 64 24 20 00 4C 8D 4C 24 68 44 8B C7 48 8B D3 48 8B C8 FF 15 E7 38 00 00	and qword ptr ss:[rsp+20],0 lea r9.qword ptr ss:[rsp+60] mov r8d.edi mov rdx.rbx mov rcx.rbx call qword ptr ds:[<dwritefile>]</dwritefile>	, v	X8/F6 000000000000000000000000000000000000	Empty 0.00000000000000000000000000000000000
qword ptr [00000101800091A0 <&wr11 0000010180005583	teFile>]= <kernel32.wri< th=""><th></th><th colspan="2">1: rcx 00000000000248 2: rdx 0000000014F920 "attrib -s -r -h %1\r 3: r8 00000000000014 4: r9 00000000014F818</th></kernel32.wri<>		1: rcx 00000000000248 2: rdx 0000000014F920 "attrib -s -r -h %1\r 3: r8 00000000000014 4: r9 00000000014F818		
Ump 1 Ump 2 Ump 3	Dump 4 Ump 5	🛞 Watch 1 🛛 🕸 I Locals 🛛 🖉 Struct		000000000014F780 =00000000000000000000000000000000000	00
Address Hex 000000000014F920 61 74 72 69 6 00000000014F930 25 31 00 0A 3A 60 000000000014F930 12 25 31 00 0A 3A 60 000000000014F930 31 20 67 6F 74 6	OD 0A 64 65 6C 20 2F A 69 66 20 65 78 69 73	46 20 2F %11del /F / 74 20 25 Q %11f exist %	^	00000000014F7C0 0000000014F9 00000000014F7C8 00007F8AF2491 00000000014F7D8 00000000000 00000000014F7D8 000000000000 00000000014F7E0 000000000000	7C return to user32.00007FF

Figure 89

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

36	000010180007040 000010180007050 0000010180007050 0000010180007050 0000010180007063 0000010180007063 0000010180007064 0000010180007064 0000010180007074 0000010180007074 0000010180007074	48 89 44 24 48 mov qword ptr ss:[rsp+48],rax 48 80 95 90 02 00 00 48 80 45 90 12 a rdx,qword ptr ss:[rbp+290]		x87r1 0000000000000000000 5T1 Empty 0.00000000000000000000000000000000000
qword ptr		eateProcessW>]= <kernel32.createprocessw></kernel32.createprocessw>		1: rcx 00000000000000 2: rdx 00000000014F890 L"\"C:\\Users\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
Dump 1	Telles of the second second	Ump 4 👹 Dump 5 🧑 Watch 1 💷 Locals 🎾 Struct		00000000014F800 =000001018000EECE 00000000014F808 0000000014F850 L"\"%\$\" \"%\$\""
Address	Hex 014F980 43 00 3A 00 5C	ASCII	^	000000000014F810 0000000014F980 L"C:\\Users\\ \ AppOat
000000000 000000000 000000000 00000000	014F990 5C 00 014F9A0 44 00 61 00 74 014F980 61 00 6C 00 5C 014F9C0 5C 00 30 00 34	00 51 00 55 00 41 00 70 00 70 00 70 00,,,		0000000000014F820 00000000000000 00000000014F828 000000000000000 00000000014F830 00000000000000 00000000014F830 00000000000000 00000000014F830 00000000000000000000000000000000000

Figure 90

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

Quantum ransomware creates a new console using the AllocConsole routine:

212	 Default (x64 fastcall)	▼ 5 🗘 🗌 Unlocke
gword ptr [00000101800090D8 <&AllocConsole>]= <kernel32.allocconsole></kernel32.allocconsole>	1: rcx 36D985A399F20000 2: rdx 00000000000000000	

Figure 91

The executable retrieves a handle to the standard output device using GetStdHandle (0xFFFFFF5 = **STD_OUTPUT_HANDLE**):



0000010180005603 B9 F5 FF FF mov ecx,FFFFFF5			
0000010180005608 FF 15 22 3A 00 00 call gword ptr ds:[<&GetStdHandle>]	~	Default (x64 fastcall)	▼ 5 ¢ Unlocke

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

<pre></pre>		X87F5 00000000000000000000 515 Empty 0.00000000000000000000000000000000000
216 0000010180005552 FF 15 C0 3E 00 00 call qword ptr ds:[<&CreateThread>]	>	Default (x64 fastcall)
qword ptr [0000010180009218 <&CreateThread>]= <kernel32.createthread></kernel32.createthread>		1: rCx 00000000000000000000000000000000000

Figure 93

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

<pre>>>> 0000010180007E15 0000010180007E1C 0000010180007E1E</pre>	4C 8D 85 80 01 00 33 D2 33 C9	00 lea r8,qword ptr ss:[rbp+180] xor edx,edx xor ecx.ecx		x87Tagword FFFF	
31P 0000010180007E20	E8 B6 AA FF FF	call <netgetdcname></netgetdcname>	>	Default (x64 fastcall)	▼ 5 🗘 🗋 Unlocke
<netgetdcname></netgetdcname>				1: rcx 0000000000000000 2: rdx 000000000000000 3: r8 00000000014FCB0	

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:

	 0000010180005CD0 0000010180005CD3 0000010180005CD8 0000010180005CE1 0000010180005CE1 	4C 8B C2 48 8D 4C 24 20 41 B9 04 00 00 00 48 8B D0 0F 11 44 24 20 0F 11 44 24 40	mov r8,rdx 16a r6x,rdword ptr ss:[rsp+20] mov r8d,4 mov r9d,4 mov r9d,5 mov r8d,rdx movups xmmoord ptr ss: [rsp+20],xmm0 movups xmmoord ptr ss: [rsp+20],xmm0		X8/FS 000000000000000000000000000000000000	ty 0.00000000000000000000000000000000000
RIP	0000010180005CEB	ES A3 CB FF FF	call <www.etaddconnectionzw></www.etaddconnectionzw>	×	Default (x64 fastcall)	🔹 💈 🗖 Unlocke
<wnetaddconn< td=""><td>ection2w></td><td></td><td></td><td></td><td>1: rcx 000000004EBFDC0 2: rdx 000000002586A8E L"pass" 3: r8 000000002586A70 L"user" 4: r9 0000000000000004</td><td></td></wnetaddconn<>	ection2w>				1: rcx 000000004EBFDC0 2: rdx 000000002586A8E L"pass" 3: r8 000000002586A70 L"user" 4: r9 0000000000000004	

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:

00000101800077C4 00000101800077C9 48 80 54 24 20 100000101800077C9 48 88 CF mov rcx,rd1 mov rcx,rd1 m	V Default (x64 fastcall)
<pre>gword ptr [0000010180009378 <&StrStrIw>]=<shlwap1.strstriw> 00000101800077CC</shlwap1.strstriw></pre>	1: rcx 000000004EBFD80 2: rdx 000000004EBFC90 3: r8 00000000000000 4: r9 0000000000000
🗱 Dump 1 🗱 Dump 2 🗱 Dump 3 🗱 Dump 4 📾 Dump 5 👹 Watch 1 💷 Locals 🖉 Struct	0000000004EBFC70 C000001018000DEA6 00000000004EBFC78 C000000004EBFC90 0000000004EBFC58 C00000000000000
Address Hex ASCII 0000000004E8FD80 5C 00 14 00 45 00 53 00 48 00 54 00 47 00 54 00 47 00 54 00 54 00 54 00 54 00 50 00 14 00 52 00 54 00 54 00 54 00 50 00 14 00 52 00 54 00 54 00 50 00 14 00 52 00 54 00 54 00 50 00 70 00 0 70 00 70 00 70 00 70 00 70 00 70 00 70 00 70 00 70 00 70 00 70 00 70 00 70 00 70 00 70 00 70 70 70	C0000000048FC80 0000000000000 C0000000048FC90 0040040041005C C0000000048FC90 00400440041005C C0000000048FC98 00000240048049 C0000000048FC98 000002407872934 C00000000048FC40 00001A031772934

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):

OD0001013000775F 48 8D 00 AA 37 00 00 [lea rck.quord ptr ds: [bis0000030] 0000010130007855 45 8B 13 mov rdk.quord ptr ds: [rbs] 0000010130007855 45 33 C0 xor rdd.rdd call.quord ptr ds: [cscopyFilew] c	, ·	x877agword_FFFF Default (x64 fastcall) ▼ 5 ♀ □ Unlock
<pre>qword ptr [0000010180009108 <&CopyFilew>]=<kernel32.copyfilew> 000001018000788C</kernel32.copyfilew></pre>		1 rdx 000000000258C6E0 L"\\\\DESKTOP- 2 rdx 00000000000000 L"\\\DESKTOP- 3 rdx 000000000000000 L"\\DESKTOP- 4 r 00000000000000000 L"\DESKTOP- 000000000000000000000000000000000000
ump 1 ump 2 ump 3 ump 4 ump 5 🛞 Watch 1 💷 Locals 🎾 Struct		000000004EBFC70 000000004EBFD70 000000004EBFC80 000000004294346
Address Hex AscII 000000000258C660 SC 00 44 00 45 00 54 00 46 00 54 00 46 00 54 00 46 00 54 00 54 00 54 00 54 00 46 00 54 00 46 00 54 00 46 00 54 00 46 00 74 00 60 00 74 00 50 00 72 00 66 00 1.4 00 54 00 54 00 54 00 54 00 74 00 60 00 1.4 7.4 10 <t< td=""><td>^</td><td>00000000048Fc80 00000000294346 00000000448Fc80 0000001180010658 00000000448Fc90 0004000500750072 00000000448Fc90 000500060066 00000000448Fc80 000500280020006 0000000048Fc80 00050028 0000000048Fc80 0002800250025005</td></t<>	^	00000000048Fc80 00000000294346 00000000448Fc80 0000001180010658 00000000448Fc90 0004000500750072 00000000448Fc90 000500060066 00000000448Fc80 000500280020006 0000000048Fc80 00050028 0000000048Fc80 0002800250025005

Figure 99

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

	000001180004604 000013180004604 000013180004604 000013180004604 00001318000460 000013180004616 0000013180004616 0000013180004618 0000013180004628 0000013180004628 000001318004628 00000138004628 00000138004628 00000138004628 00000138004628 00000138004628 00000138004628 00000138004628 00000138004628 000000138004628 0000000004628 0000000004628 0000000004628 0000000004628 000000004628 0000000004628 000000000000000000000000000000000	83 65 7F 00 48 80 45 E7 46 83 65 EF 0 4C 80 45 EF 48 89 44 24 04 5 EF 48 80 45 EF 8A 01 00 00 00 48 89 44 24 20 48 80 45 7F 48 80 45 7F	and dword ptr ss:["bp+76] 0 lea rax, gword ptr ss:["bp-19] and qword ptr ss:["bp-11] mov qword ptr ss:["sp+30], rax or rdd, FFFFFFF lea rax, qword ptr ss:["sp+30], rax mov qword ptr ss:["sp+26], rax mov qword ptr ss:["bp+76] lea rax, qword ptr ss:[K87r0 00000000000000000000 TI Empty 0.00000000000000000000000000000000000
RIP	0000010180004634	ES AE E2 FF FF	call <netshareenum></netshareenum>	>	efault (x64 fastcall) 🔹 🗧 🗌 Unlocke
<net shar<="" td=""><td>reEnum></td><td></td><td></td><td>2</td><td>: rcx 00000000258AC80 L"DESKTOP- : rdx 0000000000000000 : r8 000000004EBFCA8 : r9 00000000FFFFFFF</td></net>	reEnum>			2	: rcx 00000000258AC80 L"DESKTOP- : rdx 0000000000000000 : r8 000000004EBFCA8 : r9 00000000FFFFFFF

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	OA	OB	oc	OD	0E	OF	
00004540	73	00	2E	00	20	00	40	00	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	ÚæÜMq
00004560	8F	3C	8C	55	1E	40	5D	9C	49	34	95	FD	CE	F8	AF	ED	.<ŒU.@]œI4•ýÎø ⁻ í
00004570	71	F4	F4	43	33	AB	AO	04	E9	E7	A0	99	9B	E8	B 3	EE	qôôC3≪ .éç ™>è³î
00004580	FA	80	4E	CA	09	53	Fl	BC	A7	3B	CA	9E	CD	DA	CD	80	ú€NÊ.Sñ4\$;ÊžÍÚÍŒ
00004590	F9	9B	B 6	C4	D8	Fl	AD	F5	67	59	94	78	DE	3B	F3	AB	ù>¶ÄØñ.õgY″xÞ;ó≪
000045A0	58	FA	DA	C5	7 A	02	44	88	BE	FA	15	9C	F4	70	DC	C7	XúÚÁz.D^¾ú.œôpÜÇ
000045B0	10	11	12	8D	22	42	D7	0B	OF	B 9	D5	AB	06	4F	BD	B 3	"B×¹Õ«.0⁵≤³
000045C0	43	7D	Α3	F4	02	7B	59	55	2E	0E	94	D4	CA	B9	07	8F	C}£ô.{YU"Ôʹ
000045D0	79	E3	09	4B	A1	A 2	B2	87	50	D4	F7	46	C7	EF	56	4C	yã.K;¢°‡PÔ÷FÇïVL
000045E0	F6	56	1F	75	FD	77	10	02	D4	C0	FD	95	4C	C2	D3	DC	öV.uýwÔÀý∗LÂÓÜ
000045F0	82	AD	46	7D	8B	55	99	FO	1E	D6	D9	C0	BC	F9	18	DF	,.F} <u™ð.öùà∔¢ù.b< td=""></u™ð.öùà∔¢ù.b<>
00004600	A2	48	2B	EE	45	EC	36	F1	C0	05	F7	1D	EA	D6	F5	AE	¢H+îEì6ñÀ.÷.êÖõ⊗
00004610	FF	6D	28	79	96	B8	3D	BE	70	3C	3A	EA	05	15	3B	03	ÿm(y-,=¾p<:ê;.
00004620	EC A2	BB	El	92 67	F2	6B	8F	C7	90	69	26	A2	A2	6B	E9	28	ì≫á′òk.Ç.i&≎¢ké(
00004630	A2	AD 14	4A C6	DB	40 C5	72 E5	35 AC	D1 6B	D3 16	30 05	43 D8	39 CB	C2 BF	9E 04	EA 02	52 6B	∘.Jg@r5ÑÓ0C9žêR É.ÆÛÅå¬k.ØË¿k
00004650	2D	43	22	1D	08	20	AC 59	D3	58	05 F3	93	8C	A2	A3	02 E9	88	E.ÆUAa⊣køE¿k −C",YÓX󔌢£é^
00004650	E3	99	FD	22	91	A8	07	5E	24	35	93	83	A2 AE	38	12	2B	-C",IOAO @C£e ó™ý"`'.^\$5.f⊗8.+
00004670	23 76	76	DC	EO	45	05	D4	CB	70		32	9E	F4	EA	29	20	vvÜàE.ÔËpJ2žôê)
00001070	10	10	200	20	-10	00		00		in	02	015	-	Lin			110021020007

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

