

Makop Ransomware

Prepared by: Date: LIFARS, LLC 8/13/2021

EXECUTIVE SUMMARY

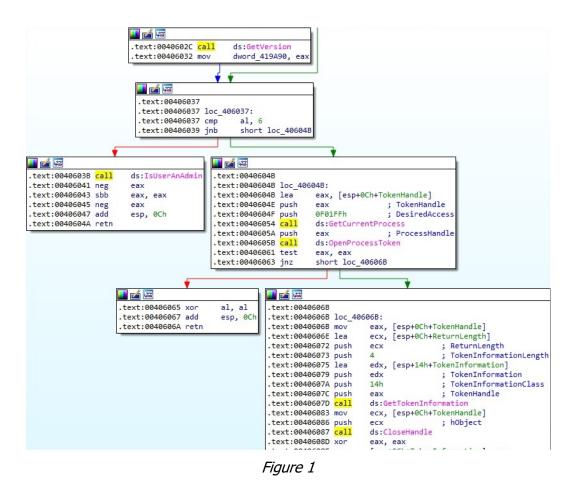
Makop ransomware encrypts user's files using the AES256 algorithm and advises the victims to contact the attackers via Tox (P2P instant-messaging protocol). The ransomware imports an AES256 key that is used to decrypt a lot of strings, including an RSA public key. There is a mutex called "m23071644" created by the process to ensure that only one instance is running at a single time and a new process spawned by the malware that encrypts network shares. The Windows Product ID is extracted from the registry and is used to generate a personal ID that will also be present in the ransom note. The ransomware deletes all volume shadow copies and kills specific processes that could lock different targeted file types. The malware operators are aware of other ransomware families because they don't encrypt possibly encrypted files by ransomware such as Shootlock, RAGA and origami. Two new AES256 keys are generated by the ransomware, which will be used interchangeably to encrypt the content of the files. A new initialization vector (IV) that consists of 16 bytes is generated and stored in the encrypted file, and the AES key used for encryption is encrypted using the RSA public key. There is no possibility to decrypt the files without knowing the RSA private key that corresponds to the hard-coded public one. Even if the operators pretend that they exfiltrate data from the network, we didn't observe any network communications.



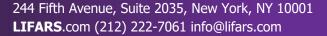
ANALYSIS AND FINDINGS

SHA256:9D90919B4434B9CAC736945384857209103FDF1A749671F190C947FDA8CC1681

The malware uses the GetVersion function to retrieve the major and minor version numbers of the OS along with other information:



The Get TokenInformation API is used to determine the elevation level of the token (0x14 = **TokenElevationType**):





00406072 51	push ecx	
00406073 GA 04 00406075 8D 54 24 0C 00406079 52 00406079 GA 14	push 4 lea edx,dword ptr ss:[esp+C] push edx push 14	x875tatusWord 0000 x875W_B 0 x875W_C3 0 x875W_C2 0 x875W_C1 0 x875W_C0 0 x875W_E5 0 x875W_SF 0 x875W_P 0 x875W_U 0
● 0040507C 50 EIE → 0040507C FF 15 18 70 40 00 < <	push eax call dword ptr ds:[c&GetTokenInformation>]	Default (stdcall)
<pre>dword ptr [00407018 <makop.&gettokeninformation>]=<adv .text:0040607D makop.exe:\$607D #547D</adv </makop.&gettokeninformation></pre>	pi32.GetTokenInformation>	1: [esp] 0000022C 2: [esp+4] 0000014 3: [esp+5] 0019FF78 4: [esp+C] 0019FF7C 5: [esp+10] 0019FF7C
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5	Watch 1 x= Locals 2 Struct 0019FF64	0000022C
Address Hex 76FE1000 1C 00 1E 00 00 DE FE 76 28 00 24 00 D4 DD FE 76FE1000 1C 00 1E 00 00 DE FE 76 1E 00 24 00 D4 DD FE	ASCII 0019FF6C	0019FF78 00000004 0019FF7C

Figure 2

The ransomware retrieves the command-line string for the current process and compares the number of arguments with 2:

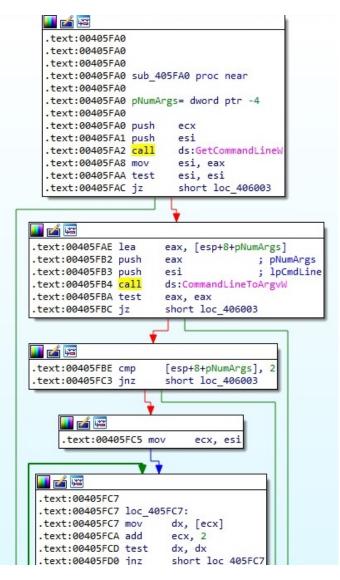


Figure 3



The CryptAcquireContextW routine is utilized to acquire a handle to a key container within a cryptographic service provider ($0x18 = PROV_RSA_AES$):



Figure 4

The following 32 bytes represent an AES256 key that will be used to decrypt a lot of strings at runtime:

		, , , , ,	
.text:004029E8	mov	[esp+4Ch+var_4C],	52h ; 'R'
.text:004029EC	mov	[esp+4Ch+var_4B],	1Fh
.text:004029F1	mov	[esp+4Ch+var_4A],	0CBh ; 'Ë
.text:004029F6	mov	[esp+4Ch+var_49],	24h ; '\$'
text:004029FB	mov	[esp+4Ch+var_48],	54h ; 'T'
.text:00402A00	mov	[esp+4Ch+var_47],	al
.text:00402A04	mov	[esp+4Ch+var_46],	1Eh
.text:00402A09	mov	[esp+4Ch+var_45],	7Fh
.text:00402A0E	mov	[esp+4Ch+var 44],	0A9h ; '©
.text:00402A13	mov	[esp+4Ch+var 43],	0D7h ; 'x
.text:00402A18	mov	[esp+4Ch+var_42],	6Bh ; 'k'
.text:00402A1D	mov	[esp+4Ch+var_41],	0DAh ; 'Ú
.text:00402A22	mov	[esp+4Ch+var_40],	0D5h ; 'Õ
.text:00402A27	mov	[esp+4Ch+var 3F],	0A6h ; '
.text:00402A2C	mov	[esp+4Ch+var_3E],	7Eh ; '~'
.text:00402A31	mov	[esp+4Ch+var_3D],	
.text:00402A36	mov	[esp+4Ch+var_3C],	91h ; '''
.text:00402A3B	mov	[esp+4Ch+var_3B],	0CAh ; 'Ê
.text:00402A40	mov	[esp+4Ch+var_3A],	0B4h ; '
.text:00402A45	mov	[esp+4Ch+var_39],	1Ch
.text:00402A4A	mov	[esp+4Ch+var 38],	6
.text:00402A4F	mov	[esp+4Ch+var_37],	al
.text:00402A53	mov	[esp+4Ch+var_36],	87h ; '‡'
.text:00402A58	mov	[esp+4Ch+var_35],	0BEh ; '%
.text:00402A5D	mov	[esp+4Ch+var_34],	44h ; 'D'
.text:00402A62	mov	[esp+4Ch+var_33],	20h ; ' '
.text:00402A67	mov	[esp+4Ch+var_32],	0BFh ; '¿
.text:00402A6C	mov	[esp+4Ch+var_31],	0Ch
.text:00402A71	mov	[esp+4Ch+var_30],	0B5h ; 'µ
.text:00402A76	mov	[esp+4Ch+var_2F],	48h ; 'H'
.text:00402A7B	mov	[esp+4Ch+var_2E],	59h ; 'Y'
.text:00402A80	mov	[esp+4Ch+var 2D],	6Dh ; 'm'



The AES key constructed earlier is imported by calling the CryptImportKey API, as shown in figure 6:



 00402ADF 00402AE0 00402AE2 00402AE4 00402AE6 00402AE6 00402AEA 00402AEA 00402AEA 	50 6A 00 6A 20 8D 4C 24 30 51 52	push eax push 0 push 0 push 2C lea ecx,dword ptr ss: push ecx push edx			x875W_SF 0 >	0000 875W_C3 0 x875W_C2 0 875W_C0 0 x875W_E5 0
EIP 00402AEC	FF 15 38 70 40 00	call dword ptr ds:[<&C	ryptimportkey>j		V Default (stdcall)	▼ 5 🜩 🗆 Unlock
• <				>		CD88 <&CPAcquireContext>
dword ptr [00407038 <makop.exe:\$2 <="" td=""><td></td><td>.CryptImportKey></td><td></td><td></td><td>2: [esp+4] 00 3: [esp+8] 00 4: [esp+C] 00 5: [esp+10] 0</td><td>19FEF0 00002C 000000</td></makop.exe:\$2>		.CryptImportKey>			2: [esp+4] 00 3: [esp+8] 00 4: [esp+C] 00 5: [esp+10] 0	19FEF0 00002C 000000
Dump 1 Dump 2	ump 3 📖 Dump 4 💷 Dump 3	🗑 Watch 1 🛛 🕅 🕅 🕅 🕅 🕅	Struct	0019FEB8 026 0019FEBC 001		
Address Hex		ASCII		0019FEC0 000		
0019FEC0 2C 00 00 00 00 00	00 00 00 00 00 00 74 AC 64	02t¬d.		0019FEC4 000 0019FEC8 000		
0019FED0 52 1F CB 24 54 86 0019FED0 91 CA 84 1C 06 86 0019FEF0 08 02 00 00 10 66 0019FF00 54 86 1E 7F A9 D7	LE 7F A9 D7 6B DA D5 A6 7E	C6 R.E\$T1.@xkU0;~4		0019FECC 026		
0019FEE0 91 CA 84 1C 06 86	57 BE 44 20 BF 0C B5 48 59	6D .E 1. 20 2. µHYm		0019FED0 24C		
0019FF00 54 B6 1E 7F A9 D7	58 DA D5 A6 7E C6 91 CA B4	1C T1. @xkUO'~4.Ê'.		0019FED4 7F1		
0019FF10 06 B6 87 BE 44 20 1	BF OC B5 48 59 6D DB 28 40	00 .¶ .#D ¿.μΗΥmÛ+@.		0019FED8 DA6		

Figure 6

The parameters of the blob are explained below:

- 08 PLAINTEXTKEYBLOB the key is a session key
- 02 CUR_BLOB_VERSION
- 0x6610 CALG_AES_256
- 0x20 key size

Using the AES key, the binary decrypts data by calling the CryptDecrypt function:

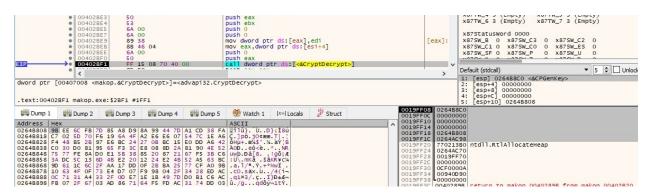


Figure 7

The result of the decryption is an RSA public key:

Address	He	ĸ															ASCII
																	¤RSA1
																	!Kvíë(ÄÚ`ÔP¦
																	<pre>peofwa.0i)A.w</pre>
																	.]e x1r&'?.'q*^ù
0264B848	AO	45	11	5D	CC	6B	3A	1E	BF	54	7A	50	6D	D3	F8	3B	E.]lk:.¿TzPmÓø;
0264B858	AB	2A	86	A5	89	65	99	9F	2B	74	44	8D	80	9D	87	67	«*.¥.e+tDg
0264B868	2E	5E	30	6B	1D	BF	D8	18	61	B 9	37	A5	OE	69	46	F7	.^0k.¿Ø.a'7¥.iF÷
0264B878	OF	0C	64	F7	49	D3	E2	B1	5D	44	D7	D9	59	B2	87	93	d÷IÓâ±]D×ÙY⁼
0264B888	85	3C	AC	B6	DB	7E	2B	FD	42	4E	31	32	DO	A9	A9	81	.<¬¶Û~+ýBN12D@@.
02648898	D9	EA	06	B7	00	00	00	00	00	00	00	00	00	00	00	00	Ùê

Figure 8

The parameters of the blob are detailed below:

- 06 PUBLICKEYBLOB the key is a public key
- 02 CUR_BLOB_VERSION
- 0xa400 CALG_RSA_KEYX
- 0x0400 key size
- 0x010001 public key exponent

Other strings are decrypted by the malicious process using the same hard-coded AES key:



Address	Нех						1000									ASCII
0264E038	61 00	64	00	6D	00	69	00	6E	00	00	00	00	00	00	00	a.d.m.i.n
Address	Нех															ASCII
0264E070	6E 00	6F	00	74	00	20	00	61	00	64	00	6D	00	69	00	n.o.ta.d.m.i.
0264E080	6F 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	n
Address	Нех															ASCII
0264E0A8	31 00	2E	00	20	00	49	00	44	00	ЗA	00	20	00	25	00	1I.D.:%.
0264E0B8	30 00	38	00	58	00	2D	00	57	00	0D	00	0A	00	32	00	0.8.XW2.
Address	Нех															ASCII
0264E100	31 00	2E	00	20	00	49	00	44	00	3A	00	20	00	25	00	1
0264E110	30 00	38	00	58	00	2D	00	44	00	0D	00	0A	00	32	00	0.8.XD2.
0264E120	2E 00	20	00	25	00	73	00	OD	00	0A	00	00	00	00	00	%.5
Address	Нех															ASCII
0264E158	4B 65	72	6E	65	6C	33	32	2E	64	6C	6C	3B	57	6F	77	Kernel32.dll;Wow
0264E168	36 34	44	69	73	61	62	6C	65	57	6F	77	36	34	46	73	64DisableWow64Fs
0264E178	52 65	64	69	72	65	63	74	69	6F	6E	3B	57	6F	77	36	Redirection; Wow6
0264E188	34 52	65	76	65	72	74	57	6F	77	36	34	46	73	52	65	4RevertWow64FsRe
0264E198	64 69	72	65	63	74	69	6F	6E	3B	41	64	76	61	70	69	direction; Advapi
0264E1A8	33 32	2E	64	6C	6C	3B	43	72	65	61	74	65	50	72	6F	32.dll;CreatePro
0004545450	CD CE	73	73	57	69	74	69	54	6F	6B	65	6E	57	3B	00	cessWithTokenW:.
0264E1B8	63 65	15	15	37	03	1 -	00	34	OF	00	05	OF	31	20	00	cesswitchiokenw,.

Figure 9

The malware retrieves the address of the following export functions by calling the GetProcAddress routine: Wow64DisableWow64FsRedirection, Wow64RevertWow64FsRedirection and CreateProcessWithTokenW. GetLocaleInfoW is used to retrieve the **LOCALE_FONTSIGNATURE** value for the default locale of the OS (0x800 =**LOCALE_SYSTEM_DEFAULT** and 0x58 =**LOCALE_FONTSIGNATURE**):

	 004055D1 004055D3 004055D7 004055D8 004055DA 	6A 20 8D 4C 24 0C 51 6A 58 68 00 08 00	00	Dush 20 lea ecx,dword Dush ecx Dush 58 Dush 800						87StatusWord 87SW_B 0 x 87SW_C1 0 x 87SW_SF 0 x	87SW_C3 87SW_C0		
EIP	→ 004055DF	FF 15 D0 70	40 00	all dword p	tr ds:[<mark><&G</mark>	etLocaleInfoW>]		>		fault (stdcall)		•	5 💠 🗌 Unlock
	004070D0 <mako< td=""><td>55DF #49DF</td><td>V>]=≺kernel32.Ge</td><td>etLocaleInfo</td><td>N></td><td></td><td></td><td></td><td>234</td><td>[esp] 00000 [esp+4] 000 [esp+8] 000 [esp+C] 000 [esp+C] 000 [esp+10] 70</td><td>000058 9FF50 000020</td><td>erne132.Get</td><td>ProcessHeap></td></mako<>	55DF #49DF	V>]=≺kernel32.Ge	etLocaleInfo	N>				234	[esp] 00000 [esp+4] 000 [esp+8] 000 [esp+C] 000 [esp+C] 000 [esp+10] 70	000058 9FF50 000020	erne132.Get	ProcessHeap>
Ump 1	Ump 2	Dump 3 🛛 🕮 Dump	0 4 💷 Dump 5	🛞 Watch 1	[x=] Locals	Struct	0019	FF38 0000 FF3C 0000	0058				
Address He				ASCII	hibib			FF40 0019 FF44 0000					

Figure 10

The ransomware decrypts even more strings, and their purpose will be explained later on:



Address	He	x															ASCII
0264E158		00	6F	00	6F	00	74	00	2E	00	69	00	6E	00	69	00	b.o.o.ti.n.i.
0264E168		00	62	00	6F	00	6F	00	74	00	66	00	6F	00	GE	00	;.b.o.o.t.f.o.n.
0264E178	74	00	2E	00	62	00	69	00	GE	00	3B	00	6E	00	74	00	tb.i.n.;.n.t.
0264E188		00	64	00	72		3B	00	6E	00	74	00	64	00	65	00	1.d.r.;.n.t.d.e.
0264E198	74	00	65	00	63	00	74	00	2E	00	63	00	6F	00	GD	00	t.e.c.tc.o.m.
0264E198	38	00	69	00	6F	00	2E	00	73	00	79	00	73	00	3B	00	
0264E188	72	00	65	00	61	00	64	00	GD	00	65	00	2D	00	77	00	;.i.os.y.s.;. r.e.a.d.m.ew.
0264E1C8	61	00	72	00	GE	00	69	00	6E	00	67	00	2D	00	74	00	
0264E1D8	78	00	74	00	3B	00	64	00	65	00	73	00	6B	00	74	00	a.r.n.i.n.gt.
0264E1D8	6F	00	70	00	2E	00	69	00		00	69	00		00	00	00	x.t.;.d.e.s.k.t. o.pi.n.i.;
Address	He		70	00	20	00	65	00	0E	00	0.5	00	120	00	00	00	ASCII
0264E178	63		68	00	72	00	6F	00	6D	00	65	00	3B	00	6D	00	c.h.r.o.m.e.;.m.
0264E188	6F	00	7A	00	69	00	6C	00	GC	00	61	00	20	00	66	00	o.z.i.l.l.af.
0264E198	69	00	72	00	65	00	66	00	6F	00	78	00	3B	00		00	i.r.e.f.o.x.:.i.
0264E198	GE	00	74	00	65	00	72	00	6E	00	65	00	74	00	20	00	n.t.e.r.n.e.t
0264E188	65		78	00	70	00	60	00	6F	00	72	00	65		72	00	e.x.p.1.o.r.e.r.
0264E1C8	3B	00		00	00	00	00	00	00	00	00	00	00	00	00	00	
Address	He		00	00	00	00	00	00	00	00	00	00	00	00	00	00	ASCII
0264E198	_		72	00	GE	00	72	00	72	00	5C	00	50	00	75	00	And the second descent of the second descent des
0264E198 0264E1A8	62	00	6C	00	65 69	00	72	00	73 3B	00	00	00	00	00	00	00	U.s.e.r.s.\.P.u. b.l.i.c.;
Address	He	_	0C	00	05	00	05	00	50	00	00	00	00	00	00	00	ASCII
0264E568		00	61	00	6B	00	6F	00	70	00	3B	00	43	00	41	00	m.a.k.o.p.;.C.A.
0264E578	52	00	40	00	4F	00	53	00	38	00	73	00	68	00	6F	00	R.L.O.S.;.s.h.o.
0264E588	GF	00	74	00	6C	00	6F	00	63	00	6B	00	38	00	73	00	o.t.1.o.c.k.;.s.
0264E598	68	00	GF	00	GF	00	74	00	6C	00	6F	00	63	00	GB	00	h.o.o.t.l.o.c.k.
0264E5A8	32	00	38	00	31	00	72	00	65	00	63	00	GF	00		00	2.;.1.r.e.c.o.e.
0264E5B8	73	00	75	00	66	00	56	00	38	00	53	00	76	00		00	s.u.f.V.8.S.v.6.
0264E5C8	67	00	38	00		00	72	00	65	00	63	00	GF	00	63	00	q.;.1.r.e.c.o.c.
0264E5D8	72	00	38	00	4D	00	34	00	59	00	4A	00	73	00	68	00	r.8.M.4.Y.J.s.k.
0264E5E8	44	00	37	00	3B	00	62	00	74	00	63	00	38	00	4B	00	J.7.;.b.t.c.;.K.
0264E5F8	44		48	00	73	00	6C	00	67	00	6A	00	6B	00		00	J.H.s.l.q.j.k.j.
0264E608	64	00	66	00	67	00	3B	00	6F	00	72	00	69	00		00	d.f.g.;.o.r.i.g.
0264E618	61	00	GD	00	69	00	3B	00	74	00	GF	00	GD	00	61	00	a.m.i.;.t.o.m.a.
0264E628	73	00	3B	00	52	00	41	00	47	00	41	00	38	00	7A	00	s.;.R.A.G.A.;.Z.
0264E638	62	00	77	00	3B	00	66	00	69	00	72	00	65	00	65	00	b.w.;.f.i.r.e.e.
0264E648	65	00	38	00	58	00	58	00	58	00	38	00	65	00	6C	00	e.;.x.x.x.;.e.l.
0264E658	65	00	6D	00	65	00	6E	00	74	00	3B	00	48	00		00	e.m.e.n.t.;.H.E.
0264E668	4C	00	50	00	3B	00	7A	00	65	00	73	00	3B	00		00	
0264E668	6F	00	63	00	6B	00	62	00	69	00	74	00	3B	00		00	L.P.;.z.e.s.;.1. o.c.k.b.i.t.;.c.
0264E688	61	00	70	00	74	00	63	00	68	00	61	00	38	00	67	00	a.p.t.c.h.a.;.g.
0264E698	75	00	6E	00	67	00	61	00	3B	00	66	00	61	00	69	00	u.n.q.a.;.f.a.i.
0264E698	72	00	3B	00	53	00	4F	00	53	00	3B	00	42	00		00	r.;.S.O.S.;.B.O.
0264E6B8	73	00		00	3B	00	GD	00	6F	00	6C	00	6F	00		00	s.s.:.m.o.1.o.c.
0264E6C8	68	00	3B	00	42	00	4B	00	47	00	48	00	4A	00	3B	00	h.;.B.K.G.H.J.;.
0264E6D8	57	00	4B	00	53	00	47	00	4A	00	3B	00	74	00	65	00	W.K.S.G.J.;.t.e.
0264E6E8	72	00	6D	00	69	00	74	00	3B	00	42	00	42	00	43	00	r.m.i.t.;.B.B.C.
0264E6E8	3B	00	64	00	61	00	72	00	68	00	3B	00	69	00	64	00	;.d.a.r.k.;.i.d.
0264E708	32	00	30	00	32	00	30	00	3B	00	61	00	72	00	63	00	2.0.2.0.;.a.r.c.
0264E708	68	00	3B	00	52	00	61	00	66	00	38	00	72	00		00	h.;.R.a.f.;.r.y.
0264E718	61	00	6E	00	3B		7A	00	78	00	7A	00	38	00		00	a.n.;.z.x.z.;.X.
0264E738	58	00	40	00	3B	00	78	00	61	00	68	00	65	00		00	X.L.;.x.a.k.e.p.
0264E748	7A	00	38	00	65	00	78	00	65	00	3B	00	64	00	6C	00	z.:.e.x.e.:.d.l.
0264E758	6C	00	3B	00	73	00	70	00	68	00	65	00	72	00	61	00	1.:.s.p.h.e.r.a.
0264E768	3B	00	40	00	6F	00	6F	00	6B	00	66	00	GF	00	72	00	;.L.o.o.k.f.o.r.
0264E778	GE	00	65	00	77	00	69	00	74	00	67	00	75	00		00	n.e.w.i.t.g.u.y.
0264E788	3B	00	58	00	48	00	41	00	4D	00	53	00	54	00	45	00	;.X.H.A.M.S.T.E.
0264E788	52	00	3B	00	78	00	64	00	71	00	64	00	3B	00	42	00	
0264E738	54	00	43	00	48	00	4F	00	52	00	53	00	45	00	42	00	R.;.x.d.q.d.;.B. T.C.H.O.R.S.E.B.
0264E7B8	4F	00	52	00		00	53	00	3B	00		00	6F		64	00	0.R.I.S.;.c.o.d.
0264E7C8				00													e.;
02042/08	05	00	20	00	00	00	00	00	00	00	00	00	00	00	00	00	

Figure 11

Address	He	x															ASCII
0264E5C8	77	00	69	00	6E	00	64	00	6F	00	77	00	73	00	3B	00	w.i.n.d.o.w.s.;.
0264E5D8	77	00	69	00	6E	00	6E	00	74	00	3B	00	5C	00	73	00	w.i.n.n.t.;.\.s.
0264E5E8	79	00	73	00	74	00	65	00	GD	00	33	00	32	00	3B	00	y.s.t.e.m.3.2.;.
0264E5F8		00		00			_	00	65		64	00	69	00	74	00	\.r.e.g.e.d.i.t.
0264E608	2E	00	65	00	78	00	65	00	3B	00	00	00	00	00	00	00	e.x.e.;
Address	He	c															ASCII
02650118	2E	00	5 B	00	25	00	30	00	38	00	58	00	2D	00	57	00	[.%.0.8.XW.
02650128	5D	00	2E	00	5 B	00	25	00	73	00	5D	00	2E	00	25	00][.%.5.]%.
02650138	73	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	5
Address	He	c														1	ASCII
02650170	6D	00	61	00	6B	00	6F	00	70	00	00	00	00	00	00	00	m.a.k.o.p
Address	He	(ASCII
026501A8	5C	00	5C	00	3F	00	5C	00	00	00	00	00	00	00	00	00	N.\.?.\

Figure 12



The binary retrieves a handle to the Shell's desktop window using the GetShellWindow API, as shown in the next figure:

→● 00405A52 89 2D 60 9A 41		ds:[419A60],ebp	
EIP 00405A58 FF 15 58 71 40	00 call dword pt	r ds:[<&GetShellWindow>]	

Figure 13

GetWindowThreadProcessId is utilized to extract the identifier of the thread and of the process that created the window from above:

00405A62 00405A62 00405A63 00405A63 <t< th=""><th></th><th>push makop.419A60 push eax call dword ptr ds:[<&GetWindowThreadProces b>]=<user32.getwindowthreadprocessid></user32.getwindowthreadprocessid></th><th>sId>] v</th><th>x87ControlWord 027F x87CW_IC 0 x87CW_ZM 1 x87CW_PM 1 → efault(stdcall) v 5 € Uniod 1: [esp1 000100A6 2: [esp+3] 00419A60 3: [esp+4] 00419A60 3: [esp+4] 06459K0 <kernel32.getprocessheap> 4: [esp+C] 0253CB8 5: [esp+10] 0019FF74</kernel32.getprocessheap></th></t<>		push makop.419A60 push eax call dword ptr ds:[<&GetWindowThreadProces b>]= <user32.getwindowthreadprocessid></user32.getwindowthreadprocessid>	sId>] v	x87ControlWord 027F x87CW_IC 0 x87CW_ZM 1 x87CW_PM 1 → efault(stdcall) v 5 € Uniod 1: [esp1 000100A6 2: [esp+3] 00419A60 3: [esp+4] 00419A60 3: [esp+4] 06459K0 <kernel32.getprocessheap> 4: [esp+C] 0253CB8 5: [esp+10] 0019FF74</kernel32.getprocessheap>
Dump 1 Dump 2 Dump 2	imp 3 💭 Dump 4 💭 Du	mp 5 👹 Watch 1 🛛 🕼 🖉 Struct	0019FF20 000100A 0019FF24 00419A6	

Figure 14

The malware obtains the join status information for the local computer by calling the NetGetJoinInformation function:

Dump 1		Dump 3 🔛 Dump 4	Dump 5 🛞 Watch 1	[x=] Locals 🖉 Struct	0019FF1C 000 0019FF20 000 0019FF24 000	419A24	makop.00419A24
	etJoinInformati A83 makop.exe:\$					1: 2: 3: 4: 5:	: [esp1] 00000000 : [esp+4] 00419A24 makop.00419A24 : [esp+6] 0019FF40 : [esp+C] 76A750F0 <kernel32.getprocessheap> : [esp+10] 0263CB88</kernel32.getprocessheap>
EIP	→ 00405A82 00405A83	55 E8 58 13 00 00	push ebp call <makop< th=""><th>.NetGetJoinInformation></th><th></th><th>××</th><th>87CW_IC 0 x87CW_ZM 1 x87CW_PM 1</th></makop<>	.NetGetJoinInformation>		××	87CW_IC 0 x87CW_ZM 1 x87CW_PM 1
	 00405A7C 00405A7D 	50 68 24 9A 41 00	push eax push makop.	419A24		Ĵ	87ControlWord 027F

Figure 15

Some directories names and a mutex name are decrypted by the executable, as shown in figure 16:

Address	Нех																ASCII
026501E0	53	00	79	00	73	00	74	00	65	00	6D	00	44	00	72	00	S.y.s.t.e.m.D.r.
026501F0	69	00	76	00	65	00	00	00	00	00	00	00	00	00	00	00	i.v.e
Address	Hex																ASCII
026514F8	58	00	ЗA	00	5C	00	50	00	72	00	6F	00	67	00	72	00	X.:.\.P.r.o.g.r.
02651508	61	00	6D	00	44	00	61	00	74	00	61	00	5C	00	6D	00	a.m.D.a.t.a.\.m.
02651518	69	00	63	00	72	00	6F	00	73	00	6F	00	66	00	74	00	i.c.r.o.s.o.f.t.
02651528	5C	00	77	00	69	00	6E	00	64	00	6F	00	77	00	73	00	\.w.i.n.d.o.w.s.
02651538	5C	00	63	00	61	00	63	00	68	00	65	00	73	00	00	00	\.c.a.c.h.e.s
Address	Hex																ASCII
02651570	58	00	3A	00	5C	00	55	00	73	00	65	00	72	00	73	00	X.:.\.U.s.e.r.s.
02651580	5C	00	41	00	6C	00	6C	00	20	00	55	00	73	00	65	00	\.A.1.1U.s.e.
02651590	72	00	73	00	5C	00	4D	00	69	00	63	00	72	00	6F	00	r.s.\.M.i.c.r.o.
026515A0	73	00	6F	00	66	00	74	00	5C	00	57	00	69	00	6E	00	s.o.f.t.\.W.i.n.
026515B0	64	00	6F	00	77	00	73	00	5C	00	43	00	61	00	63	00	d.o.w.s.\.C.a.c.
026515C0	68	00	65	00	73	00	00	00	00	00	00	00	00	00	00	00	h.e.s
Address	Нех				1000												ASCII
02651618	6D 3	32	33	30	37	31	36	34	34	00	00	00	00	00	00	00	m23071644

Figure 16

The value of the "SystemDrive" environment variable is retrieved using the GetEnvironmentVariableW API:



						0 0 11 0 0 1 0 0 0 0 0 0
	0040582A 0040582A 0040582F 00406833 00406834 00406834 00406834 € 00406835 € € 00407004 «makop. 835 makop.exe:\$6		<pre>push 104 lea eax,dword ptr ss:[[esp+1C]] push eax push easi call dword ptr ds:[c&GetEnvironmentVariablew>] =- <kernel32.getenvironmentvariablew></kernel32.getenvironmentvariablew></pre>	esi:L"	x87Sw_I 0 x87Sw_TOP 0 (ST0=x87r0) x87ControlWord 027F x87CW_ZC 0 x87CW_ZM 1 x87CW_IC 0 x87CW_ZM 1 x87CW_PM 1 Default(stdcal) 5 ① Unlod 1: [esp+4] 00197D18 5 ① Unlod 2: [esp+4] 0000104 5: 6: 3: [esp+6] 00000104 x1AllocateHeap> 5: 6:	
Address Hee		Dump 3 💭 Dump 4 💭 Dump	5 🛞 Watch 1 [x=] Locals 🖉 Struct	0019FCF4 02650 0019FCF8 0019F 0019FCFC 00000	01E0 L"SystemDrive" 1918 0104	

Figure 17

The ransomware creates a mutex called "m23071644" to ensure that only one instance of the executable is running at a single time:

> 00406AE2 00406AE3 00406AE5 00406AE7 00406AE7	56 6A 01 6A 00 FF 15 C8 70 40 00	push esi push 1 push 0 call dword ptr ds:[<&Cr	eateMutexA>]	esi:"m	x87Contro1Word 027F x87Cw_LC0 x87Cw_PM 1
dword ptr [004070C8 <makop.&< td=""><td></td><td>CreateMutexA></td><td></td><td>1</td><td>Default (stacall) ▼ 5 € Unlod 1: [esp] 00000000 2: [esp+4] 00000001 2: [esp+6] 002651618 "m23071644" 4: [esp+C] 02651688 5: [esp+C] 0038A000</td></makop.&<>		CreateMutexA>		1	Default (stacall) ▼ 5 € Unlod 1: [esp] 00000000 2: [esp+4] 00000001 2: [esp+6] 002651618 "m23071644" 4: [esp+C] 02651688 5: [esp+C] 0038A000
Dump 1 Dump 2	mp 3 💭 Dump 4 💭 Dump	D 5 🛞 Watch 1 🛛 🕅 🖉	Struct	0019FF2C 0000000 0019FF30 0000000 0019FF34 0265161	1

Figure 18

The process opens the "SOFTWARE\Microsoft\Windows NT\CurrentVersion" registry key using the RegOpenKeyExA routine:

BI 2	00406884 00406885 0040688A 0040688A 0040688C 0040688C 0040688C 004068C2 004068C2	50 68 19 01 02 00 6A 00 51 68 02 00 00 80 FF 15 30 70 40	push eax push 20119 push 0 push ecx push 800000 call dword	ptr ds:[<&R	eg0penKeyExA>]		eax:&"	x875tatusWord 0000 x875W_B 0 x875W_C3 0 x875W_C2 0 x875W_C1 0 x875W_C0 0 x875W_E5 0 x875W_SF 0 x875W_P 0 x875W_U 0 x875W_0 0 x875W_Z 0 x875W_D 0	
	<pre></pre> <pre>0407030 <makop. <pre="">c2 makop.exe:\$6</makop.></pre>		advapi32.RegOpenKeyExA	,			>	Default (stdcal) 1: [esp] 8000002 2: [esp+4] 02651618 "SOFTWARE\\Microsof 3: [esp+4] 0000010 4: [esp+c] 00020119 5: [esp+10] 0019FAF4 & "SOFTWARE\\Micros	
Dump 1 Address Hex 02651670 50		Dump 3 Dump 4	Dump 5 Watch 1 ASCII ProductId.	[x=] Locals	Struct	0019F 0019F 0019F	AE0 80000002 AE4 02651618 AE8 00000000 AEC 00020119 AF0 0019FAF4	"SOFTWARE\\Microsoft\\Windows NT\\Curren &"SOFTWARE\\Microsoft\\Windows NT\\Curre	

Figure 19

The Windows product ID is extracted from the registry and it will be used to compute a victim ID:

004068D9 004068D9 004068D4 88 54 24 08 004068D4 50 004068D5 6A 00 004068E1 52 004068E1 52 004068E3 004068E3 52 004068E3 52 004068E3 52 004068E3 52 004068E3 004068 004068E3 004068E3 004068E3 004068E3 004068E3 004068E3 004068E3 004068	push edx mov edx,dword ptr ss:[esp+8] push eax push o push o push edx push edx call dword ptr ds:[<&RegQueryValueExA>]	ecx:"P	x8/1m_b s (EmpLy) x8/1m_/ s (EmpLy) x875W_E0 0000 x875W_E0 0 x875W_C2 x875W_E0 0 x875W_E2 x875W_E0 0 x875W_E5 x875W_E0 0 x875W_E0 x875W_E0 0 x875W_E0 x875W_E0 0 x875W_E0 x875W_E0 0 x875W_E0 Default (stdcall) ▼ 5 € Unded
dword ptr [0040702C <makop.®queryvalueexa>]=<adva< td=""><td>pi32.RegQueryValueExA></td><td></td><td>Li [esp] 00000288 2: [esp+4] 02651670 "ProductId" 3: [esp+5] 00000000 4: [esp+C] 00000000 4: [esp+C] 00000000 5: [esp+L] 0000100000</td></adva<></makop.®queryvalueexa>	pi32.RegQueryValueExA>		Li [esp] 00000288 2: [esp+4] 02651670 "ProductId" 3: [esp+5] 00000000 4: [esp+C] 00000000 4: [esp+C] 00000000 5: [esp+L] 0000100000
Hownp 1 Dump 2 Dump 3 Dump 4 Dump 4 Address Hex F6 64 75 63 74 49 64 00 00 00 <	p 5 @ Watch 1 [k=]Locals	019FA05 0000288 0019FADC 02651670 019FAE0 0000000 019FAE4 0000000 019FAE5 0019F820 0019FAE5 0019F828	"ProductId"



The ransomware extracts the volume serial number of the C drive by calling the GetVolumeInformationW API:



	0,1 ,0100000 10
D0406354 O0406354 O0406356 GA 00 Dush 0 Dush 0 Dush 0 Dush 0 Ou406358 GA 00 Ou406358 GA 00 Ou406358 GA 00 Ou406358 GA 00 Ou40635 S5 S4 Ou40636 S5 Ou40636 S5 Ou406261 GA 00 Ou406263 GA 00 Ou406263 S0 Ou40626 S0 Ou4062 S0 Ou4062 Ou40626 S0 Ou4062 Ou40626 Ou40626 Ou40626 Ou4062 Ou40 Ou40626 Ou40 Ou40626 Ou4062 Ou40 Ou4062 Ou40 Ou40 Ou40626 Ou40 Ou40 Ou40 Ou40626 Ou40 Ou40 Ou40 Ou40626 Ou40 Ou40	Default (stdcall)
dword ptr [004070FC <makop.&getvolumeinformationw>]=<kernel32.getvolumeinformationw> .text:0040626A makop.exe:\$626A #566A</kernel32.getvolumeinformationw></makop.&getvolumeinformationw>	I: [esp1]00.3F8F8 L"C:\\" 2: [esp+4]0000000 3: [esp+8]0000000 4: [esp+2]005F8F0 5: [esp+10]0000000
Image: Second state	001958260 00195876 "C':\\" 0019580c 0000000 0019580b 0000000 0019580b 0000000 0019580b 0000000 0019580c 0000000 0019580c 0000000 00195826 0000000 00195824 0000000

Figure 21

The malware uses a custom "hash" function to compute a 4-byte value that corresponds to the Product ID. A snippet of the implementation is shown below:

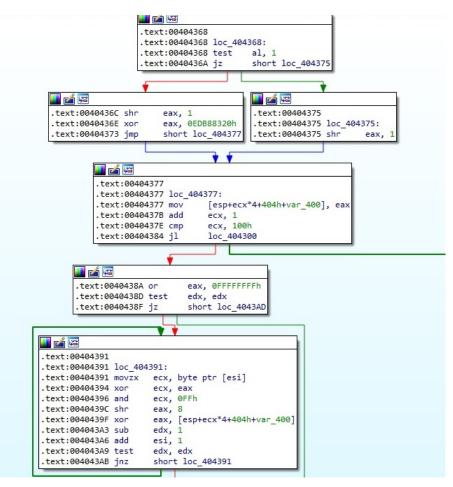


Figure 22

Two more strings are decrypted by the ransomware using the AES key imported before:



Address	He	(ASCII
0264FF80	25	00	73	00	20	00	28	00	25	00	30	00	38	00	58	00	%.s(.%.0.8.X.
0264FF90	29	00	25	00	63	00	20	00	25	00	49	00	36	00	34	00).%.c%.I.6.4.
0264FFA0	64	00	2E	00	25	00	30	00	32	00	49	00	36	00	34	00	d%.0.2.I.6.4.
0264FFB0	64	00	20	00	67	00	62	00	20	00	28	00	25	00	75	00	dg.b(.%.u.
0264FFC0	29	00	2F	00	25	00	49	00	36	00	34	00	64	00	2E	00)./.%.I.6.4.d
0264FFD0	25	00	30	00	32	00	49	00	36	00	34	00	64	00	20	00	%.0.2.I.6.4.d
0264FFE0	67	00	62	00	20	00	28	00	25	00	75	00	29	00	2F	00	g.b(.%.u.)./.
0264FFF0	25	00	75	00	25	00	25	00	0D	00	0A	00	00	00	00	00	96. U. 96. 96
Address	Нех	c															ASCII
02650550	33	00	2E	00	20	00	54	00	6F	00	74	00	61	00	6C	00	BT.o.t.a.l.
02650560	3A	00	20	00	25	00	49	00	36	00	34	00	64	00	2E	00	:%.I.6.4.d
02650570	25	00	30	00	32	00	49	00	36	00	34	00	64	00	20	00	%.0.2.I.6.4.d
02650580	67	00	62	00	20	00	28	00	25	00	75	00	29	00	2F	00	g.b(.%.u.)./.
02650590	25	00	49	00	36	00	34	00	64	00	2E	00	25	00	30	00	%.I.6.4.d%.O.
026505A0	32	00	49	00	36	00	34	00	64	00	20	00	67	00	62	00	2.I.6.4.dg.b.
026505B0	20	00	28	00	25	00	75	00	29	00	2F	00	25	00	75	00	.(.%.u.)./.%.u.
026505C0	25	00	25	00	OD	00	0A	00	00	00	00	00	00	00	00	00	96.96

Figure 23

The binary disables the file system redirection for the calling thread using Wow64DisableWow64FsRedirection:

EIP 00405F80 00405F81 00405F81 00405F81	50 FF 15 44 9A 41 00 E0 04 ED EE EE	push eax call dword ptr ds:[<&Wow64DisableWow64FsRedirection>]		x875W_0 0 x875W_Z 0 x875W_D 0
dword ptr [00419A44 <makop.&< td=""><td>Wow64DisableWow64FsRedi</td><td>rection>]=<kernel32.wow64disablewow64fsredirection></kernel32.wow64disablewow64fsredirection></td><td></td><td>1: [esp] 0019FF20</td></makop.&<>	Wow64DisableWow64FsRedi	rection>]= <kernel32.wow64disablewow64fsredirection></kernel32.wow64disablewow64fsredirection>		1: [esp] 0019FF20
.text:00405F81 makop.exe:\$5F	81 #5381			2: [esp+4] 0038A000 3: [esp+8] 0264E568 4: [esp+C] FFFFFFF 5: [esp+10] 00404571 makop.00404571
Dimo 1 Bill Dimo 2 Bill Di		0019FF14	0019FF2	20

Figure 24

An open handle to the current process is obtained by calling the GetCurrentProcessId and OpenProcess APIs:

	00407098 <makop.< th=""><th></th><th>nel32.0penProcess></th><th></th><th></th><th>></th><th>2: [esp 3: [esp</th><th>00000400 +4] 00000000 +8] 00000874</th><th>▼ 5 ÷ Unlock</th></makop.<>		nel32.0penProcess>			>	2: [esp 3: [esp	00000400 +4] 00000000 +8] 00000874	▼ 5 ÷ Unlock
Dump 1		Dump 3 🛄 Dump 4	💭 Dump 5 🛛 👹 Watch 1	[x=] Locals	Struct 00	019F67C 00000 019F680 00000 019F684 00000	5: [esp 0400 0000	+10] 0264E568	

Figure 25

The malicious executable opens the access token associated with the current process:

EIP	• 00405D 00405D • 00405D • 00405D	01 68 0 06 56 07 FF 1	0 00 00 02 5 20 70 40 0	00	push eax push 2000000 push esi call dword p		penProcessToken>		~ ×	875W_SF 0 x875W_P 875W_O 0 x875W_Z	0 x875W_D 0
dword ptr [00407020 <m< td=""><td>akop.@OpenPr</td><td>ocessToken></td><td>1=<advapi32< td=""><td>.OpenProcess</td><td>Token></td><td></td><td>,</td><td></td><td>fault (stdcall)</td><td>▼ 5 🚖 🗆 Unlock</td></advapi32<></td></m<>	akop.@OpenPr	ocessToken>	1= <advapi32< td=""><td>.OpenProcess</td><td>Token></td><td></td><td>,</td><td></td><td>fault (stdcall)</td><td>▼ 5 🚖 🗆 Unlock</td></advapi32<>	.OpenProcess	Token>		,		fault (stdcall)	▼ 5 🚖 🗆 Unlock
	DD7 makop.ex								2:3:4:5:	[esp+4] 02000000 [esp+8] 0019F69C [esp+C] 76A750F0	<kernel32.getprocessheap></kernel32.getprocessheap>
Dump 1	Dump 2	💭 Dump 3	🚛 Dump 4	Ump 5	Watch 1	[x=] Locals	Struct	0019F67C 00000 0019F680 02000 0019F684 0019	0000		

Figure 26

DuplicateTokenEx is utilized to create a new access token that duplicates the existing token:



		01.010.0000
00405E0A 50 push eax 00405E0B 68 00 00 00 2 push 2000 00405E10 51 push eax 00405E11 89 7C 24 38 poy dword 00405E15 FF 15 28 70 40 00 call dword	ptr ss:[esp+38],edi ptr ds:[<&OuplicateTokenEx>]	x877W_6 3 (EmpLy) x87TW_7 3 (EmpLy) x87TW_6 3 (EmpLy) x87TW_7 3 (EmpLy) x87SW_8 0 x87SW_C3 0 x87SW_C2 0 x87SW_2 0 x87SW_C3 0 x87SW_2 0 x87SW_2 0 x87SW_2 0 x87SW_2 0 x87SW_8 0 x87SW_2 0 x87SW_0 0 x87SW_8 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0 x87SW_1 0
<pre>dword ptr [00407028 <makop.&duplicatetokenex>]=<advapi32.duplicate .text:00405E15 makop.exe:\$5E15 #5215</advapi32.duplicate </makop.&duplicatetokenex></pre>	okenEx>	1: [esp] 0000264 2: [esp+4] 0200000 3: [esp+8] 0019F6A8 4: [esp+c] 0000002 5: [esp+10] 0000001
Image: Second	IX= Locals Struct 0019F674 0019F674 0019F674 0019F674 0019F674 0019F674 0019F674	20 0000264 74 0200000 78 0019F6A8 70 00000002 80 0000001 84 0019F638

Figure 27

The ransomware creates an anonymous pipe by calling the CreatePipe function:

	55 80 44 24 24 50 80 42 20 51 80 54 24 20 89 7C 24 30 89 7C 24 30 67 44 24 38 01 00 00 00 67 44 24 38 01 00 00 00	push ebp lea eax,dword ptr ss:[esp+24] push eax lea ecx,dword ptr ss:[esp+20] push ecx push ecx mov dword ptr ss:[esp+30],ed1 mov dword ptr ss:[esp+30],i call dword ptr ss:[esp+30],i ss:[esp+3	X 8 X 8 X 8 X 8 X 8 X 8 X 8 X 8 X 8 X 8	3757W_4 3 (Empty) x877W_5 3 (Empty) 377W_6 3 (Empty) x877W_7 3 (Empty) 375tatusWord 0000 x875W_C 3 0 x875W_C 2 0 375W_5 0 x875W_C 3 0 x875W_E 2 0 x875W_5 0 x875W_E 0 375W_5 0 x875W_C 0 x875W_U 0 x875W_U 0 375W_5 0 x875W_C 0 x875W_U 0 x875W_U 0 375W_5 0 x875W_C 0 x875W_U 0 x875W_U 0
dword ptr [004070B4 <makop.8< th=""><th></th><th></th><th>1: 2:</th><th>ault (stdcall) ▼ 5 🐳 🗌 Unlod [esp1 0019F6A4 [esp+4] 0019F6A0</th></makop.8<>			1: 2:	ault (stdcall) ▼ 5 🐳 🗌 Unlod [esp1 0019F6A4 [esp+4] 0019F6A0
.text:00405E4E makop.exe:\$5E	4E #524E			[esp+8] 0019F6A8 [esp+C] 00000000 [esp+10] 76A750F0 <kernel32.getprocessheap></kernel32.getprocessheap>
Dump 1 Dump 2	ump 3 💭 Dump 4 💭 Dump	5 🛞 Watch 1 🛛 🖛 Locals 🖉 Struct	0019F678 0019F6A4 0019F67C 0019F6A0	

Figure 28

There is a new process spawned by the malware with the "-n<Process ID>" parameter. The new process handles the encryption of the network shares, as will be detailed in the upcoming paragraphs:

312	O 0405 E&1 O 405 E&2 O 405 E&2 O 405 E&2 O 405 E&3 O 405 E&3 O 405 E>3 O 405 E>3	52 88 54 24 14 89 84 24 84 00 80 84 24 80 0 50 55 55 55 55 55 55 55 55 55 55 55 55	0 00 00 0 00 00 4 00 00 00 0 00 00 00 00	push edx mov dx, dword ptr s: mov dword ptr ss: lea eax, dword ptr ss push eax push ebp push ebp	sp+84],eax sp+80],eax s:[esp+44] s:[esp+9C]	okenw>]	ecx:L" 44: 'D'	x87r5 000000000000000000000000000000000000	. 000C . 000C
	[00419A38 <makop. SEBA makop.exe:\$</makop. 		ithTokenW>]=<	advapi32.CreateProce:	sswithTokenw>			1: [esp] 00000290 2: [esp+4] 0000000 3: [esp+8] 0000000	Des
	5EBA makop.exe:\$					0019F664 0019F668 0019F662	00000000	1: [esp1 00000290 2: [esp+4] 0000000 4: [esp+4] 00000000 4: [esp+c] 0019F710 L"\"C:\\Users\"\\ 5: [esp+10] 00000000 0	

Figure 29

The process restores the file system redirection for the current thread:

00405F90 00405F91 00405F93	51 8A D8 FF 15 50 9A 41 00	<pre>push ecx mov bl,al call dword ptr ds:[<&wow64RevertWow64FsRedirection>] mov al bl</pre>	~	x875W_SF 0 x875W_P 0 x875W_0 0 x875W_Z 0
<			>	Default (stdcall)
dword ptr [00419A50 <mako .text:00405F93 makop.exe:</mako 		ection>]= <kernel32.wow64revertwow64fsredirection></kernel32.wow64revertwow64fsredirection>		1: [esp] 00000000 2: [esp+4] 0038A000 3: [esp+8] 0264E568 4: [esp+C] 0000000 5: [esp+10] 00404571 mak
Dump 1 Dump 2		0019FF14	00000000	

Figure 30



The file enforces the system to send critical errors to the calling process using the SetErrorMode API (0x1 = **SEM_FAILCRITICALERRORS**). It obtains the currently available disk drives using GetLogicalDrives:

● 004012CE ● 004012D0 ● 004012D4 ● 004012D4	6A 01 89 5C 24 18 FF 15 50 70 40 00	<pre>push 1 mov dword ptr ss:[esp+18],ebx call dword ptr ds:[<&SetErrorMode>] call dword ptr ds:[<&SetErrorMode>]</pre>		x875W_SF 0 x875W_P x875W_0 0 x875W_Z	0 x875W_D 0
· · ·			>	Default (stdcall)	🔻 5 🖨 🗌 Unloci
dword ptr [00407050 <makop.< td=""><td>&SetErrorMode>]=<kerne13< td=""><td>2.SetErrorMode></td><td></td><td>1: [esp] 00000001</td><td><kernel32.getprocessheap></kernel32.getprocessheap></td></kerne13<></td></makop.<>	&SetErrorMode>]= <kerne13< td=""><td>2.SetErrorMode></td><td></td><td>1: [esp] 00000001</td><td><kernel32.getprocessheap></kernel32.getprocessheap></td></kerne13<>	2.SetErrorMode>		1: [esp] 00000001	<kernel32.getprocessheap></kernel32.getprocessheap>
.text:004012D4 makop.exe:\$1	2D4 #6D4			2: [esp+4] 76A750F0 3: [esp+8] 00000000 4: [esp+C] 0038A000 5: [esp+10] 0264E568	
	FF 15 F0 70 40 00	call dword ptr ds: [<&GetLogicalDrives>]	0019FEA4 00000001		

Figure 31

The GetDriveTypeW API determines if a disk drive is a removable, fixed, CD-ROM, RAM or network drive. Makop doesn't target CD-ROM drives and RAM disks:

EIP 00401336 00401337 00401337	51 FF 15 48 70 40 00	push ecx call dword ptr ds:[<&GetDriveTypew>]	ecx:L" ~	x87SW_0 0 x87SW_Z	0 x875₩_D 0
dword ptr [00407048 <makop.< th=""><td>&GetDriveTypeW>]=<kernel< td=""><td>32.GetDriveTypeW></td><td></td><td>1: [esp] 0019FED8 L"C 2: [esp+4] 76A750F0 < 3: [esp+8] 00000000</td><td>kernel32.GetProcessHeap></td></kernel<></td></makop.<>	&GetDriveTypeW>]= <kernel< td=""><td>32.GetDriveTypeW></td><td></td><td>1: [esp] 0019FED8 L"C 2: [esp+4] 76A750F0 < 3: [esp+8] 00000000</td><td>kernel32.GetProcessHeap></td></kernel<>	32.GetDriveTypeW>		1: [esp] 0019FED8 L"C 2: [esp+4] 76A750F0 < 3: [esp+8] 00000000	kernel32.GetProcessHeap>
.text:00401337 makop.exe:\$1	337 #737			4: [esp+C] 0038A000 5: [esp+10] 00000043	
Dump 1 Dump 2	Dump 2 III Dump 4	mp E 🙀 Watch 1 Iz=l acala 🗐 Struct	0019FEA4 0019FED8	L"C:\\"	



The malware opens the "<u>C:</u>\" drive using the CreateFileW routine (0x80000000 = **GENERIC_READ**, 0x3 = **FILE_SHARE_READ**|**FILE_SHARE_WRITE**, 0x3 = **OPEN_EXISTING** and 0x80 = **FILE_ATTRIBUTE_NORMAL**):

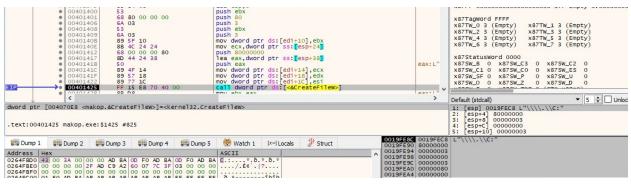


Figure 33

DeviceIoControl is utilized to retrieve the physical location of the specified volume (0x560000 = **IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS**):





				0100000 13	
L> 0040144B 00401452 00401452 00401452 00401452 00401452 00401452 00401452 00401452 00401453 00401453 00401453 00401453	80 54 24 18 52 6A 20 80 44 24 4C 50 6A 00 6A 00 68 00 00 56 00 53 FF 15 44 70 40 00	<pre>push 0 lea edx,dword ptr ss:[esp+18] push edx push edx push edx push edx push eax push 0 push soon push soon push soon push ebx call dword ptr ds:[<&DeviceIoControl>]</pre>		x87TW_2 3 (Empty) xx x87TW_4 3 (Empty) xX x87TW_6 3 (Empty) xX x87TW_6 3 (Empty) xX x87StatusWord 0000 x87SW_B 0 x87SW_C3 x87SW_C1 0 x87SW_C3 x87SW_F5 0 x87SW_P	77W_3 3 (Empty) 77TW_3 3 (Empty) 77TW_7 3 (Empty) 77TW_7 3 (Empty) 0 x875W_C2 0 0 x875W_E5 0 0 x875W_U 0 0 x875W_D 0 0 x875W_D 0
<	0E CO	Itart asv asv	>	Default (stdcall)	▼ 5
dword ptr [00407044 <make .text:00401463 makop.exe</make 	op.&DeviceIoControl>]= <kerne \$1463 #863</kerne 	l32.DeviceIoControl>		1: [esp] 00000268 2: [esp+4] 00560000 3: [esp+8] 0000000 4: [esp+C] 00000000 5: [esp+10] 0019FEE8	
Dump 1 Dump 2	🗒 Dump 3 🛛 💭 Dump 4 💭 Du	mp 5 👹 Watch 1 🛛 🛛 🖉 Struct	0019FE88 00000268 0019FE8C 00560000		
Address Hex 0264FBD0 43 00 3A 00 00 0264FBD0 00 00 00 02 2F 0264FBD0 00 00 00 00 00 00 0264FBD0 00 00 00 00 00 00 00 0264FBD0 00 00 00 00 00 00 00 0264FD0 01 F0 AD BA AB AB	00 AD BA 0D F0 AD BA 0D F0 AD C9 AZ 60 07 7C 3F 03 00 0 00 00 00 00 00 00 00 00 00 00 0 AB AB AB AB AB AB AB AB EE FE 1 00 00 00 00 23 F 70 41 EF AD	ASCII D BA (, °, ∂, °, ∂, ° 00 00, / £€ , [? EE FE , ∂, °≪≪≪≪≪≪101	▲ 0019FE90 0000000 0019FE94 0000000 0019FE98 0019FE88 0019FE9C 0000000 0019FEA0 0019FEBC 0019FEA4 0000000		

Figure 34

Makop generates 2 buffers of 32 random bytes using the CryptGenRandom API (let's call the first one **AESKey1** and the second one **AESKey2**):

00401692 00401693 00401693 00401693 00401698 00401694 00401684 00401684 00401684 00401687 00401687	51 66 C7 06 08 02 66 C7 46 04 10 66 C6 46 08 20 88 17 6A 20 52 FF D5 57 75	push ecx mov word ptr ds: [esi],208 mov word ptr ds: [esi]+4],6610 mov byte ptr ds: [esi]+8],20 mov edx,dword ptr ds: [edi] push 20 push edx call ebp	20: ' '	x87StatusWord 0000 x87SW_B 0 x87SW_C3 x87SW_C1 0 x87SW_C0 x87SW_SF 0 x87SW_P x87SW_0 0 x87SW_Z	0 x875W_U 0 0 x875W_D 0
ebp= <advapi32.crvptgenrandom< td=""><td>(*******</td><td></td><td>></td><td>Default (stdcall)</td><td>▼ 5 ≑ 🗌 Unloc</td></advapi32.crvptgenrandom<>	(*******		>	Default (stdcall)	▼ 5 ≑ 🗌 Unloc
.text:004016A7 makop.exe:\$16/				1: [esp] 0264CD88 <&C 2: [esp+4] 0000020 3: [esp+8] 02651414 4: [esp+C] 76A750F0 < 5: [esp+10] 00000000	kernel32.GetProcessHeap>
Dump 1 Dump 2 Dump 2	imp 3 🗰 Dump 4 💷 Dump	5 🛞 Watch 1 🛛 🖉 Struct	0019FEF8 0264CD8 0019FEFC 0000002	0	
Address Hex		ASCII	0019FF00 0265141	.4	
Address Hex 02651414 0E 7E D2 44 70 AF 7E 02651424 C9 46 11 A2 DA 9F B0	3 0A C4 56 E0 04 8A 28 B9 0 80 C0 C1 46 D5 E2 BB 96	ASCII CB .~ODp {.ÄVà('Ĕ 7D ÉF.¢Ù.'.ÀAFÔå».}			
		Figure 35			

E16-	00401701 00401702 00401707 00401707 00401711 00401713 00401715 00401716	52 66 C7 06 08 02 66 C7 46 04 10 6 C6 46 08 20 88 07 6A 20 50 FF DS E7 DS	push edx mov word ptr ds mov word ptr ds mov byte ptr ds mov eax, dword p push 20 push eax call ebp	[esi+4],6610 [esi+8],20		20:'''	ASTRLES (Empusy) ASTRL'S (Empusy) x875talusWord 0000 x875WLS 0 x875WLC3 0 x875WLC2 0 x875WLS10 x875WLC3 0 x875WLE5 0 x875WLS10 x875WL9 0 x875WL0 x875WL9 0 x875WL9 x875WL9 0 x875WL9 x875WL9 0 x875WL9 x875WL9 0 x875WL9 x875W
	<					>	Default (stdcall) 🔻 5 🖨 🗌 Unlock
	2.CryptGenRando						1: [esp] 0264CD88 <&CPAcquireContext> 2: [esp+4] 0000020 3: [esp+8] 0265293C 4: [esp+C] 76A750F0 <kernel32.getprocessheap> 5: [esp+10] 00000000</kernel32.getprocessheap>
Dump 1	Dump 2	0ump 3 🔛 Dump 4	📖 Dump 5 🛛 🛞 Watch 1 🛛 🕼	=l Locals 🛛 🐉 Struct	0019FE	FS 0264CD88 FC 00000020	
Address Hex			ASCII		, 0019FF	00 0265293C	
0265293C FF	OC DF E2 3F CF	81 5A 24 OC 1E CF C8 94 F4 D5 FA 8F	A9 61 00 7B ¥'ù'ùwÈ.ÔÔú.@a	æs - {			

Figure 36

4 bytes that will be used as a marker in the encrypted files are decrypted by the binary: "AD AD 6B A1". The RSA public key is imported using the CryptImportKey function:



■ 0040235A 50 0040235B 88 05 0040235F 6A 00 0040235F 6A 00 0040235F 51 00402361 51 00402365 52 00402365 50 00402368 FF 15 38 70 40 00 00402368 FF 15 38 70 40 00	push eax mov eax,dword ptr ds:[esi] push 0 push ecx push ecx push ecx push ecx int dword ptr ds:[<&CryptImportKey>] tert ex ex		x871m_0 5 (EmpLy) x871m_1 5 (EmpLy) x875tatusword 0000 x875w_C3 0 x875w_C2 0 x875w_C1 0 x875w_C0 0 x875w_E5 0 x875w_SF 0 x875w_P 0 x875w_U 0 x875w_C2 0 x875w_D 0 x875w_C2 0 x875w_D 0
dword ptr [00407038 «makop.&CryptImportKey>]= <advap13 #1d64<="" .text:00402964="" makop.exe:\$2964="" td=""><td>2.CryptImportKey></td><td>></td><td>Default(stdcal)</td></advap13>	2.CryptImportKey>	>	Default(stdcal)
Image: Second state Image: Second state	5 000 March 1 March 2003 2 Struct 0013 001	FEFC 02655DDC	return to makop.0040184D from makop.00402920 kerne132.GetProcessHeap

Figure 37

These 2 buffers are encrypted using the RSA public key and contain the following information: 4-byte marker, 4-byte "hash" value of Product ID generated earlier, 4-byte the volume serial number, 4-byte value obtained from the result of the GetLocaleInfoW call, **AESKey1** (or **AESKey2**) and 4-byte "hash" value of this buffer, computed using the same function used for Product ID:

 0040298A 8D 0040298E 51 0040298E 52 0040298F 52 00402993C 6A 00402994 6A 00402995 6A 00402995 60 00402995 50 00402997 C7 	4C 24 0C 1ea 00 pus pus 00 pus pus 00 pus pus 44 24 24 75 00 00 15 34 70 40 00 cal	sh 80 a cex,dword ptr ss:[esp+C] sh ecx sh edx sh dox sh 0 sh 0	75:'u' >	Aor(m_L 2 (Empty) Aor(m_D 2 (Empty) X87Tw_L 3 (Empty) X87Tw_D 3 (Empty) X87Tw_G 3 (Empty) X87Tw_D 7 3 (Empty) X87Stu_G 3 (Empty) X87Tw_D 7 3 (Empty) X87Stw_G 0 0000 X87Sw_G 0 0000 X87Sw_G 0 0 X87Sw_C 0 0 X87Sw_D 0 X87Sw_G 0 0 X87Sw_D 0 0 X87Sw_G 0 0 X87Sw_D 0 X87Sw_G 0 0 X87Sw_D 0 Defaul(stdcal) Defaul(stdcal) 1: [esp14] 00000000 3: [esp+4] 00000000
.text:0040299F makop.exe:\$299F #1		ao	9FED0 02655F00	5: [csp+C] 00000000 5: [csp+10] 02655DD0
Address Hex Outpoint Top in the point of the poi	ZF AD C9 A2 B3 C6 C0 A3 C4 56 E0 04 8A 28 B9 CB - C0 C1 45 D5 E2 B8 96 C0 - <td>Watch 1 Ix=llcals Struct 001 SCII 001 001 k:Im 17/.54 001 001 S00 7 1.4%, ('E' 001 001 Stu 001 001 001 Stu 001 001 001 001 Stu 001</td> <td>19FED4 0000000 19FED8 0000000 19FEDC 00000000 19FEEC 02655DD 19FEEE 00000080 19FEEC 02655DD 19FEEC 02655F00 19FEEF 00000080</td> <td>return to makop.0040184D from makop.00402920</td>	Watch 1 Ix=llcals Struct 001 SCII 001 001 k:Im 17/.54 001 001 S00 7 1.4%, ('E' 001 001 Stu 001 001 001 Stu 001 001 001 001 Stu 001	19FED4 0000000 19FED8 0000000 19FEDC 00000000 19FEEC 02655DD 19FEEE 00000080 19FEEC 02655DD 19FEEC 02655F00 19FEEF 00000080	return to makop.0040184D from makop.00402920

Figure 38



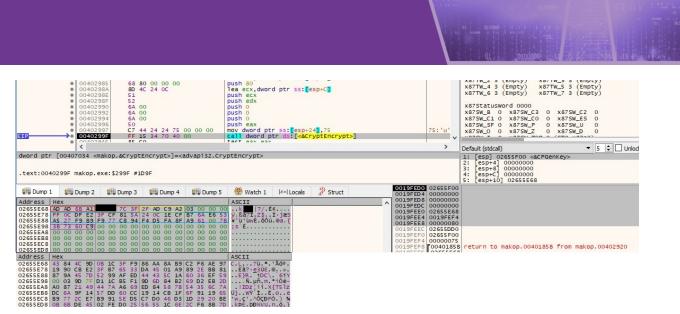


Figure 39

The ransomware decrypts the following strings and uses the GetProcAddress API to obtain the addresses of the export functions:

Address	He	(ASCII
02651060	6E	74	64	6C	6C	2E	64	6C	6C	3B	4E	74	51	75	65	72	ntdll.dll;NtQuer
02651070	79	4F	62	6A	65	63	74	3B	4E	74	51	75	65	72	79	53	yObject; NtQueryS
																	ystemInformation
02651090	3B	52	74	6C	47	65	74	56	65	72	73	69	6F	6E	3B	4B	;RtlGetVersion;K
026510A0	65	72	6E	65	6C	33	32	2E	64	6C	6C	3B	47	65	74	46	ernel32.dll;GetF
																	inalPathNameByHa
																	ndleW;QueryFullP
																	rocessImageNameW
026510E0	3B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;



NtQueryObject is used to retrieve information about the system and the current process (0x3 = ObjectAllTypesInformation):



Figure 41

The executable gets information about the OS using the RtlGetVersion API. It compares the major version number (0xa for Windows 10) of the OS with 0x6:



00404603 50 00404603 C7 44 24 04 1C 01 00 00 00101633 FF 15 80 9A 41 00 C C C C C C C	push eax mov dword ptr ss:[esp+4],11C call dword ptr ds:[<artigetversion>]</artigetversion>	· · · ·	x875w_SF 0 x875w_P x875w_0 0 x875w_Z	0 x875W_U 0 0 x875W_D 0	04.03
dword ptr [00419A80 <makop.&rtlgetversion>]=<ntdll.rtl .text:004046E2 makop.exe:\$46E2 #3AE2</ntdll.rtl </makop.&rtlgetversion>	GetVersion>		1: [esp] 0019FDD0 2: [esp+4] 0000011C 3: [esp+8] 0000001 4: [esp+C] 00000000 5: [esp+10] 00000000		

Figure 42

There is a call to GetTokenInformation that obtains information about whether virtualization is enabled for the token (0x18 = TokenVirtualizationEnabled):

	004043EE 004043EF 004043F1 004043F1 004043F1 004043F6 004043F6 004043F6 004043F6 ● 004043F6 004043F6 004043F6 004043F1 004043F6 004043F6 004043F6 004043F6 004043F6 004043F6 004043F6 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3 00403F3	51 6A 04 8D 54 24 0C 52 6A 18 50 FF 15 18 70 4(56 CO &GetTokenInforma 43F9 #37F9	00 call	edx,dword pt edx 18 eax dword ptr d	ls:[<&Ge	tTokenInformation	>]		× 2 2 3 4	875W_C1 x875W_C0 1875W_SF x875W_C 1875W_O x875W_Z 1875W_0 x875W_Z 1875W_0 x875W_Z 1875W_1 000002A 1 [esp+4] 00000018 [esp+4] 1 [esp+6] 0000004	0 0 0
Dump 1	x	Dump 3 🔛 Dump 4	ASC	II	l Locals	Struct	^ 0	019FEF4 000002 019FEF8 000000 019FEFC 0019FF 019FF00 000000	2A0 018 FOC	: [esp+10] 0019FF10	

Figure 43

The malware decrypts even more data using the CryptDecrypt routine:

Address	He	< .															ASCII
02651458	76	73	73	61	64	6D	69	6E	20	64	65	6C	65	74	65	20	vssadmin delete
																	shadows /all /qu
02651478	69	65	74	0A	77	62	61	64	6D	69	6E	20	64	65	6C	65	iet.wbadmin dele
02651488	74	65	20	63	61	74	61	6C	6F	67	20	2D	71	75	69	65	te catalog -quie
02651498	74	0A	77	6D	69	63	20	73	68	61	64	6F	77	63	6F	70	t.wmic shadowcop
026514A8	79	20	64	65	6C	65	74	65	0A	65	78	69	74	0A	00	00	y delete.exit
Address	Нех	c															ASCII
0264E5C8	43	00	6F	00	GD	00	53	00	70	00	65	00	63	00	00	00	C.o.m.S.p.e.c



The ComSpec environment variable points to the command line interpreter and its content is extracted using the GetEnvironmentVariableW API:

dword ptr [004062Di 004062Ei 004062Ei	8D 8 51 50 FF 1			Fart asv asv	tr ds:[<mark><&G</mark>	etEnvironment	/ariablew>	1	eax:L"	264E5C8 L 0019F8F0	0 0 x 0 x 2 0 x	87 SW_ES 87 SW_U 87 SW_D	0] Unloci
.text:00406	2E9 makop.exe	:\$62E9 #5	5E9								00000104 76A750F0 0265145	<kerne< th=""><th>132.GetF dmin del</th><th>rocessHe ete shade</th><th>ap> ows /a</th></kerne<>	132.GetF dmin del	rocessHe ete shade	ap> ows /a
Dump 1	Dump 2	Dump 3	Ump 4	Ump 5	Watch 1	[x=] Locals	Struct		0019F844 0019F848 0019F84C	0019F8F0					



The CreatePipe routine is utilized to create an anonymous pipe that is used as an inter-process communication mechanism:



EIP	 0040630 0040631 0040631 0040633 0040633 0040633 0040633 0040633 0040633 0040633 	SD SD 1 50 2 8D 6 51 7 8D 8 52 C FF			ea eax, dword ush eax ea ecx, dword ush ecx ea edx, dword ush edx all esi	ptr ss:[e	sp+20]			esi:Cr v	x87 x87 x87 x87 x87	7SW_B (7SW_C1 (7SW_C1 (7SW_SF (7SW_SF (7SW_0 ((Empty) vord 0000 0 x875W_C 0 x875W_C 0 x875W_Z 0 x875W_Z	0 0 x879 0 x879 0 x879	SW_C2 SW_ES SW_U SW_D	0	
esi= <kernel< td=""><td>132.CreatePip</td><td>e> (76A744</td><td>90)</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>[esp+4]</td><td>019F870 0019F868</td><td></td><td></td><td>5 - 11</td><td>JNIOCH</td></kernel<>	132.CreatePip	e> (76A744	90)									[esp+4]	019F870 0019F868			5 - 11	JNIOCH
.text:00406	53EC makop.ex	e:\$63EC #5	7EC								4:	[esp+C]	00000000 76A750F0	<pre>kernel3</pre>	2.GetP	rocessHea	.p>
Dump 1	Dump 2	Dump 3	Dump 4	💷 Dump 5	🛞 Watch 1	[x=] Locals	2 Struct		0019F844	0019F870 0019F868							
Address He	ex		!		ASCII			,		0019F898							

Figure 46

The pipe created above will be inherited by child processes:

Jump 1		Ump 3	Dump 4	Ump 5	🛞 Watch 1	[x=] Locals	Struct	0019F844 000002A4 0019F848 00000001 0019F84C 00000000		
	el32.SetHandl 0641E makop.e))					1: [esp] 000002A4 2: [esp+4] 00000001 3: [esp+8] 0000000 4: [esp+C] 76A750F0 < 5: [esp+10] 02651458	kernel32.GetProcessHeap> "vssadmin delete shadows ;
1	<							>	Default (stdcall)	▼ 5 ≑ 🗌 Unla
11	00406				tart asv asv			CST.SE V		
TR	004064 004064		c		call esi			esi:Se u	x875W_0 0 x875W_Z	0 x875W_D 0
	• 004064		1		push ebp push 1				x87SW_C1 0 x87SW_C0 x87SW_SF 0 x87SW_P	0 x875W_U 0

Figure 47

There is a new cmd.exe process created by the ransomware:

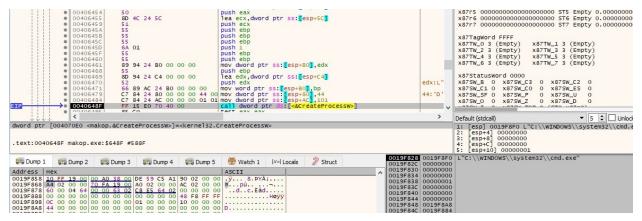
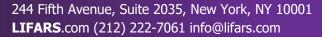


Figure 48

All volume shadow copies are deleted by the cmd.exe process using the following commands:

- vssadmin delete shadows /all /quiet
- wbadmin delete catalog -quiet
- wmic shadowcopy delete



	55 8D 44 24 48 51 51 52 FF 15 5C 70 40 00 59 47 24 44 &writeFile>]= <kernel32.writ< th=""><th>push ebp laa eax doord ptr ss:[esp+48] push ecx push ecx push edi push edi call dword ptr ds:[<dwritefile>] may ery dword ptr ss:Ferne448 eFile></dwritefile></th><th>edi:"v</th><th>x875tatusword 0000 x875w_B 0 x875w_C2 0 x875w_C2 0 x875w_C1 0 x875w_C2 0 x875w_E5 0 x875w_5F 0 x875w_P 0 x875w_U 0 x875w_D 0 x875w_Z 0 x875w_D 0 Defaul (stdcall) ▼ 5 ↓ Unlod 1: [esp 00000244 2: [esp+4] 00261458 3: [esp+4] 000005F 4: [esp+1] 0009F894 5: [esp+1] 0009F894</th></kernel32.writ<>	push ebp laa eax doord ptr ss:[esp+48] push ecx push ecx push edi push edi call dword ptr ds:[<dwritefile>] may ery dword ptr ss:Ferne448 eFile></dwritefile>	edi:"v	x875tatusword 0000 x875w_B 0 x875w_C2 0 x875w_C2 0 x875w_C1 0 x875w_C2 0 x875w_E5 0 x875w_5F 0 x875w_P 0 x875w_U 0 x875w_D 0 x875w_Z 0 x875w_D 0 Defaul (stdcall) ▼ 5 ↓ Unlod 1: [esp 00000244 2: [esp+4] 00261458 3: [esp+4] 000005F 4: [esp+1] 0009F894 5: [esp+1] 0009F894
Address Hex 02651458 76 73 73 61 64 60 02651458 76 73 73 61 64 67 02651468 73 68 61 64 67 72 02651478 69 65 74 0A 77 62 02651488 74 65 20 63 61 64 69 63 02 02 65 14 04 77 62 04 77 60 63 63 64 65 20 63 61 64 65 04 07 76 69 63 63 64 65 63 63 64 77 60 63 63 64 65 74 0A 77 60 63 63 74 0A 77 60 63 63 74 0A 77 60 63 63 74 0A 77 0A	Dump 3 Dump 4 Dump 5 69 62 20 64 65 6C 65 74 65 73 20 27 16 64 65 62 26 73 20 27 61 64 65 6C 65 74 65 6C 65 74 65 6C 65 74 65 74 65 74 65 6C 65 74 65 74 65 74 65 74 65 74 65 74 75 69 72 20 73 68 61 64 67 77 73 67 74	ASCII 20 Vssadmin delete ZS shadows /all /qu 65 iet.wbadmin dele 65 te catalog -quie 70 t.wmic shadowcop	0019582 0000024 0019584 0265143 ▲ 0019584 0000055 ● 0019584 0019584 ● 0019584 0019584 ● 0019584 0195957 ● 0019585 7647507 ● 0019585 0238400 ● 0019585 0338400 ● 0019585 0415500	"vssadmin delete shadows /all /quiet\nwbadmin di kernel32.GetProcessHeap "vssadmin delete shadows /all /quiet\nwbadmin di

Figure 49

The output of the above operations is transmitted to our initial process via pipes:

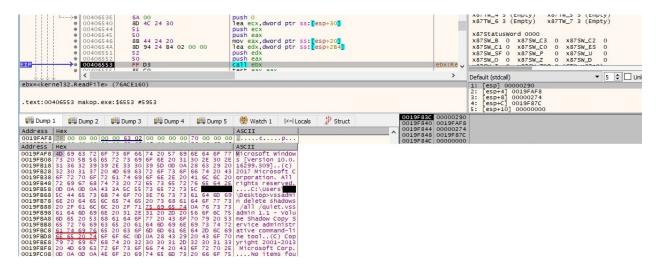
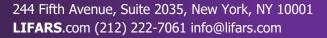


Figure 50

A small part of the processes that will be killed by the malware is presented in figure 51 (the entire list can be found in the appendix):

Address	He	ĸ															ASCII
026505E8	6D	00	73	00	66	00	74	00	65	00	73	00	71	00	6C	00	m.s.f.t.e.s.q.l.
026505F8	2E	00	65	00	78	00	65	00	3B	00	73	00	71	00	6C	00	e.x.e.;.s.q.1.
02650608	61	00	67	00	65	00	6E	00	74	00	2E	00	65	00	78	00	a.g.e.n.te.x.
02650618	65	00	3B	00	73	00	71	00	6C	00	62	00	72	00	6F	00	e.;.s.q.l.b.r.o.
02650628	77	00	73	00	65	00	72	00	2E	00	65	00	78	00	65	00	w.s.e.re.x.e.
02650638																	;.s.q.l.s.e.r.v.
																	re.x.e.;.s.q.
02650658	6C	00	77	00	72	00	69	00	74	00	65	00	72	00	2E	00	1.w.r.i.t.e.r
02650668										00							
																	1.ee.x.e.;.o.
02650688	63	00	73	00	73	00	64	00	2E	00	65	00	78	00	65	00	c.s.s.de.x.e.
02650698	3B	00	64	00	62	00	73	00	6E	00	6D	00	70	00	2E	00	;.d.b.s.n.m.p
026506A8	65	00	78	00	65	00	3B	00	73	00	79	00	6E	00	63	00	e.x.e.;.s.y.n.c.
026506B8	74	00	69	00	6D	00	65	00	2E	00	65	00	78	00	65	00	t.i.m.ee.x.e.
026506C8	3B	00	61	00	67	00	6E	00		00			72	00	76	00	;.a.g.n.t.s.r.v.
026506D8	63	00	2E	00	65	00	78	00	65	00	3B	00	6D	00	79	00	ce.x.e.;.m.y.
026506E8	64	00	65	00	73	00	6B	00	74	00	6F	00	70	00	71	00	d.e.s.k.t.o.p.q.
026506F8	6F	00	73	00	2E	00	65	00	78	00	65	00	3B	00	69	00	o.se.x.e.;.i.





CreateToolhelp32Snapshot is utilized to take a snapshot of the processes ($0x2 = TH32CS_SNAPPROCESS$):

EIP	004066 004066 004066	CC 6A 0	2 1 07 00 00		oush 0 oush 2 all kmakop.	CreateToolh	e1p32Snapshot>	×	•	x875W_SF 0 x875W_P 0 x875W_U 0 x875W_O 0 x875W_Z 0 x875W_D 0	
ļ	<								>	Default (stdcall) 🔻 5 🜩	Unl
<makop.crea< td=""><td>teToolhelp32</td><td>2Snapshot></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1: [esp] 00000002</td><td></td></makop.crea<>	teToolhelp32	2Snapshot>								1: [esp] 00000002	
.text:00406	GCE makop.ex	xe:\$66CE #5A	КСЕ							2: [esp+4] 0000000 3: [esp+8] 76A750F0 <kernel32.getprocessh 4: [esp+C] 0000000 5: [esp+10] 0019FF74</kernel32.getprocessh 	Heap>
Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	🛞 Watch 1	[x=] Locals	3 Struct	0019FCC0 0019FCC4			

Figure 52

The processes are enumerated using the Process32FirstW and Process32NextW functions:

EIP	 004066E8 004066E9 004066EA 004066E7 004066E7 	55 C7 44 24 24 2C 0	2 00 00 push ecx push ebp mov dword ptr ss:[esp call <makop.process321< th=""><th></th><th></th><th>x875w_C1 0 x875w_C0 0 x875w_E5 0 x875w_SF 0 x875w_P 0 x875w_U 0 x875w_O 0 x875w_Z 0 x875w_D 0</th></makop.process321<>			x875w_C1 0 x875w_C0 0 x875w_E5 0 x875w_SF 0 x875w_P 0 x875w_U 0 x875w_O 0 x875w_Z 0 x875w_D 0
1	<				>	Default (stdcall) 🔻 5 🖨 🗌 Unk
	cess32FirstW> 66F2 makop.exe	\$66F2 #5AF2				1: [esp] 000002AC 2: [esp+4] 0019FCE4 3: [esp+6] 76A750F0 <kernel32.getprocessheap> 4: [esp+C] 00000000 5: [esp+10] 0019FF74</kernel32.getprocessheap>
Dump 1	Ump 2	Dump 3 🛛 Dump 4	💷 Dump 5 👹 Watch 1 [x=] Locals	2 Struct 00	019FCC0 000002AC	
EIP	● 00406794 00406795 ● 00406795 00406795	55 E8 33 06 00 00	push edx push ebp call smakop.process32t	NextW>		x875W_SF 0 x875W_P 0 x875W_U 0 x875W_O 0 x875W_Z 0 x875W_D 0
malian Dran	ess32NextW>					Default (stdcall) 🔻 5 🖶 🗌 Unloc
	5796 makop.exe	\$6796 #5B96				1: [esp] 000002AC 2: [esp+4] 0019FCE4 3: [esp+8] 76A750F0 <kerne]32.getprocessheap> 4: [esp+C] 00000000 5: [esp+t0] 0019FF74</kerne]32.getprocessheap>
Ump 1	Dump 2	Dump 3 💭 Dump 4	🕮 Dump 5 🛛 🛞 Watch 1 🛛 💷 Locals		19FCC0 000002AC 19FCC4 0019FCE4	

Figure 53

The following function is used to compare processes' names with the targeted list:



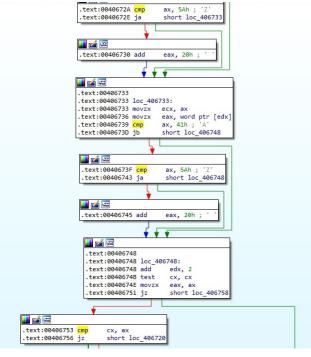


Figure 54

Any targeted process found is killed using the TerminateProcess routine:

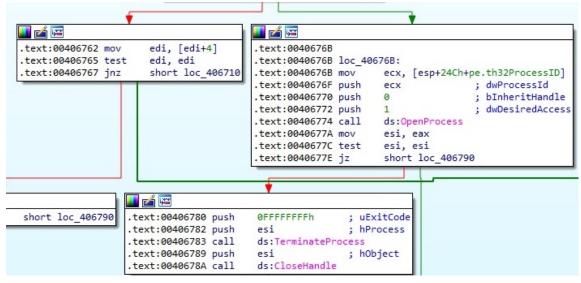


Figure 55



The following extension will be appended to the encrypted files (note the "hash" of the Product ID):

Address	Hey	(ASCII
02665FE2			00	2D	00	57	00	5D	00	2E	00	5 B	00	5D	00	[.3.F.7.C.

Figure 56

A new thread is created to encrypt the current directory. Similar threads will be created to encrypt the "<u>C:\</u>", "<u>C:\ProgramData</u>" and "<u>C:\Users</u>" directories:

C>● 00401AD8 O401ADC O401ADC O401ADC O0401ADE O0401ADE O0401AE3 O0401AE3	51 56 68 F0 27 40 00 51 51 89 4E 20 89 4E 24 FF 15 F4 70 40 00 FF 15 F4 70 40 00	push ecx push ecx push ecx push ecx push ecx push ecx mov dword ptr ds:[es1+20].ecx mov dword ptr ds:[es1+24].ecx call dword ptr ds:[<&createThread>]	esi:åL	x87Tw_6 3 (Empty) x87Tw_7 3 (Empty) x87Tw_6 3 (Empty) x87StatusWord 0000 x87SW_8 0 x87SW_C0 0 x87SW_C2 0 x87SW_C1 0 x87SW_C0 0 x87SW_E5 0 x87SW_5F 0 x87SW_F 0 x87SW_D 0 x87SW_0 0 x87SW_2 0 x87SW_D 0 x87SW_0 0 x87SW_2 0 x87SW_D 0 pefault (tdcall) ▼ 5 ♀ uhod
dword ptr [004070F4 <makop.4< td=""><td></td><td>CreateThread></td><td></td><td>1: [esp] 00000000 2: [esp+4] 0000000 3: [esp+8] 004027F0 makop.004027F0 4: [esp+C] 0265FA0 &L"C:" 5: [esp+10] 00000000</td></makop.4<>		CreateThread>		1: [esp] 00000000 2: [esp+4] 0000000 3: [esp+8] 004027F0 makop.004027F0 4: [esp+C] 0265FA0 &L"C:" 5: [esp+10] 00000000
Image: Dump 1 Image: Dump 2 Image: D		ASCII 04 ĐûdAe. Àr.	0119FEC3 0000000 0019FEC0 0000000 0019FED0 004027F0 0019FED4 02655FA0 0019FED8 0000000 0019FEDC 0000000	makop.004027F0 &L"C:"

Figure 57

It's important to mention that the following folders will not be encrypted: "<u>C:\WINDOWS</u>", "C: <u>\ProgramData\microsoft\windows\caches</u>", "C:\Users\All Users\Microsoft\Windows\Caches" and "C: \Users\Public". Also, the process doesn't target directories that contain "windows" or "winnt" in their names.

THREAD ACTIVITY – START ADDRESS FUNCTION

The files are enumerated using the FindFirstFileW and FindNextFileW APIs:

● 00402481 50 00402482 56 EIT → 00402482 FF 15 40 70 40 00 (MAD1050 FC	push eax push esi call dword ptr ds:[<&FindFirstFilew>]	esi:L"	x875w_SF 0 x875w_P 0 x875w_U 0 x875w_O 0 x875w_Z 0 x875w_D 0 Default (stdcall)
dword ptr [00407040 <makop.&findfirstfilew>]=<kernel32< td=""><td>.FindFirstFileW></td><td></td><td>1: [esp] 02651458 L"C:\\Users\\ \\Desktop*.*</td></kernel32<></makop.&findfirstfilew>	.FindFirstFileW>		1: [esp] 02651458 L"C:\\Users\\ \\Desktop*.*
.text:004024B3 makop.exe:\$24B3 #18B3			2: [esp+4] 0492FD18 3: [esp+8] 004027F0 makop.004027F0 4: [esp+C] 02655FA0 &L"C:" 5: [esp+10] 02655FA0 &L"C:"
💭 Dump 1 💭 Dump 2 💭 Dump 3 💭 Dump 4 💭 Dump 9	5 🛞 Watch 1 🛛 🛛 🖉 Struct	0492FCF0 02651458 0492FCF4 0492FD18	L"C:\\Users\\\\Desktop*.*"
00402785 51 00402785 52 00402785 52 00402785 55 00402785 55	<pre>push ecx push edx call dword ptr ds:[<&FindNextFileW>] test eax eax</pre>		x875W_SF 0 x875W_P 0 x875W_U 0 x875W_0 0 x875W_Z 0 x875W_D 0
<	Ther any any	>	Default (stdcall) 🔻 5 ≑ 🗌 Unloc
dword ptr [00407054 <makop.&findnextfilew>]=<kernel32.< td=""><td>FindNextFileW></td><td></td><td>1: [esp] 02650EC0</td></kernel32.<></makop.&findnextfilew>	FindNextFileW>		1: [esp] 02650EC0
.text:00402786 makop.exe:\$2786 #1886			2: [esp+4] 0492FD18 3: [esp+8] 004027F0 makop.004027F0 4: [esp+C] 02655FA0 &L"C:" 5: [esp+10] 02655FA0 &L"C:"
💭 Dump 1 💭 Dump 2 💭 Dump 3 💭 Dump 4 💭 Dump	5 🛞 Watch 1 🛛 🕼 🖉 Struct	0492FCF0 02650EC0 0492FCF4 0492FD18	

Figure 58

The files that have the following extensions will be skipped: "makop", "CARLOS", "shootlock", "shootlock2", "1recoesufV8Sv6g", "1recoer8M4YJskJ7", "btc", "KJHslgjkjdfg", "origami", "tomas", "RAGA", "zbw", "fireee", "XXX", "element", "HELP", "zes", "lockbit", "captcha", "gunga", "fair", "SOS", "Boss", "moloch", "BKGHJ", "WKSGJ", "termit", "BBC", "dark", "id2020", "arch", "Raf", "ryan", "zxz", "XXL", "xakepz", "exe", "dll", "sphera", "Lookfornewitguy", "XHAMSTER", "xdqd", "BTCHORSEBORIS", "code". Some of these extensions like shootlock, origami, raga and others are the result of other ransomware infections (Shootlock, Origami and Raga ransomware). The following files are not encrypted by Makop: "boot.ini",



"bootfont.bin", "ntldr", "ntdetect.com", "io.sys", "readme-warning.txt", "desktop.ini". The ransomware opens a file for encryption using the CreateFileW API:

	6A 00 68 80 00 00 00 6A 03 6A 00 6A 04 68 00 00 00 C0 53 C6 44 24 38 00 FF D6 ep ce	push 0 push 80 push 3 push 4 push 4 push ebx mov byte ptr ss:[esp+38],0 call esi	ebx:L" esiicr	
esi= <kernel32.createfilew> .text:004034FC makop.exe:\$3</kernel32.createfilew>				Default(stdcall)
Address Hex 02652540 43 00 3A 00 5C 00 02652550 5C 00	55 00 73 00 65 00 72 0 00 5C 00 44 00 65 0		0492FCBC 0265 0492FCC0 0000 0492FCC8 0000 0492FCC8 0000 0492FCC 0000 0492FCC0 0000 0492FC0 0000 0492FC0 0000	00004 00000 00003 00080

Figure 59

CryptGenRandom is utilized to generate 16 random bytes:



Figure 60

The process imports AESKey1 using the CryptImportKey routine:







The initialization vector (IV) is set to the 16-byte buffer generated above:

	00402CE5 6A 00402CE7 52 00402CE8 6A 00402CEA 50 00402CEB FF	00 01 15 00 70 40 00	push 0 push edx push 1 push eax call dword ptr	ds:[<&CryptSetKey	Param>]		×8750 ×8750 ×8750		0 0 x87SW_ES 0 x87SW_U 0 x87SW_D	0
dword ptr [00407			api32.CryptSetKeyPar	am>		>	2: [e 3: [e 4: [e	(stdcall) sp] 02650F10 <& sp+4] 00000001 sp+8] 02655D98 sp+C] 00000000 sp+10] 00000004	CPGenKey>	Vnloci
Address Hex	ump 2 💭 Dump 3	5D BE F9 11 A0 1A	ASCII	=l Locals 2 Struct		0492FC58 02 0492FC5C 00 0492FC60 02 0492FC64 00	000001 655D98			

Figure 62

The filename is encrypted using **AESKey1**:

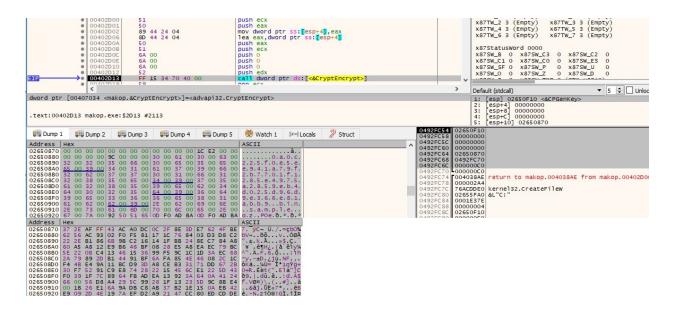


Figure 63

The encrypted filename is written to the file:





O04038CB 6A 00 4C 24 14 O04038CD 51 O04038CD 51 O04038D2 52 O04038D4 55 O04038D4 O0400 O0000 O00000 O0000000 O00000000	push 0 lea ecx,dword ptr ss:[esp+14] push ecx push edx push edx push edi push edp fact asy asy >	x87TW_6 3 (Empty) x87TW_7 3 (Empty) x87StatusWord 0000 x87SW_B 0 x87SW_C3 0 x87SW_C2 0 x87SW_C1 0 x87SW_C0 0 x87SW_E5 0 x87SW_55 0 x87SW_P 0 x87SW_U 0 x87SW_5 0 x87SW_Z 0 x87SW_U 0 x87SW_5 0 x87SW_2 0 x87SW_U 0 x87SW_5 0 x87SW_5 0 x87SW_5 0 x87S
esi= <kernel32.writefile> (76ACE2SO) .text:004038DD makop.exe:\$38DD #2CDD</kernel32.writefile>		1: [esp] 00000244 2: [esp+4] 02650870 3: [esp+4] 000000C0 4: [esp+C] 0492FC88 5: [esp+10] 00000000
Image Dump 1 Image Dump 2 Image Dump 3 Image Dump 4 Image Dump 5 Address Hex 1 1 3 2 2 4 1 0 0 1	ASCII 0432rCGC <	650870 0000C0 92FC88 000000 ACDDED kernel32.CreateFilew 655FA0.du"C:" 000004 000000 000000 000000 000000 000000

Figure 64

A 4-byte value that represents the encrypted filename size is also written to the file:

Odd38F5 Odd38F7 Odd38F7 Odd38F7 Odd38F8 Odd38F8 Odd38F8 Odd38F Odd38F	push 0 lea ecx,dword ptr ss:[esp+14] push ecx push 4 lea edx,dword ptr ss:[esp+24] push edx call esi tert esy esy >	x87 m_0 3 (cmpty) x87 m_1 3 (cmpty) x87 StatusWord 0000 x87 SW_6 0 X87 SW_C 3 0 X87 SW_C 2 0 x87 SW_C 1 0 X87 SW_C 0 0 X87 SW_C 2 0 x87 SW_C 5 0 X87 SW_D 0 X87 SW_D 0 x87 SW_C 5 0 X87 SW_D 0 X87 SW_D 0 x87 SW_C 0 0 X87 SW_Z 0 X87 SW_D 0 Default (stdcall) ▼ 5 🗇 Unloc
esi= <kernel32.writefile> (76ACE250) .text:00403904 makop.exe:\$3904 #2D04</kernel32.writefile>		1: [esp] 000002A4 2: [esp+4] 0492FC90 3: [esp+6] 0000004 4: [esp+C] 0492FC88 5: [esp+L0] 00000000
Jimp 1 Jimp Jump 2 Jimp Jump 3 Jimp Jump 4 Jimp Jump 5 Address Hex 0492FC90 C0 00	Image: Watch 1 Image: Image: Watch 1 Image: Image: Watch 1 Image: Image: Watch 1 Image: Watch 1 <thimage: 1<="" th="" watch=""> <thimage: 1<="" th="" watch=""></thimage:></thimage:>	0002A4 92FC90 000004 92FC88

Figure 65

The initialization vector that was generated earlier is added to the encrypted file:

	0040391D 0040391F 00403923 00403924 00403926 00403927 00403928	6A 00 8D 44 24 14 50 6A 10 57 55 FF D6 95 CO		oush 0 ea eax,dword oush eax oush 10 oush edi oush ebp all esi	d ptr ss: [esp+14		esi:Wr	×8 ×8 ×8	7SW_C1 0 x87SW_C0	- (CTO	0
esi= <kernel32< td=""><td colspan="9">sti=<kernel32.writefile> (76ACE250)</kernel32.writefile></td><td>ault (stdcall) [esp] 000002A4</td><td></td><td>🕶 5 ≑ 🗌 Unloci</td></kernel32<>	sti= <kernel32.writefile> (76ACE250)</kernel32.writefile>									ault (stdcall) [esp] 000002A4		🕶 5 ≑ 🗌 Unloci
.text:0040392	text:00403928 makop.exe:\$3928 #2D28									[esp+4] 02655D98 [esp+8] 00000010 [esp+C] 0492FC88 [esp+10] 00000000		
🗰 Dump 1 🚦	Dump 2	Dump 3 🛛 💷 Dump 4	Ump 5	🛞 Watch 1	[x=] Locals	Struct		0492FC64 00 0492FC68 02	655D98	3		
Address Hex 02655D98 FD 6	D OE AD 20 AG	74 68 5D BE F9 1		ASCII ým ¦tk]‰ù	AZ		^	0492FC6C 00 0492FC70 04 0492FC74 00	92FC 88	3		

Figure 66

The encrypted buffer that also contains **AESKey1** is written to the file:



	O0403950 O403950 O403952 S0 4C 24 14 Jea ecx, dword ptr ss:[[e5p+14]] O0403957 S1 O0403957 O8 80 00 00 00 push 80 O0403950 S5 push eax O0403950 S5 O0403950 S O0403950 S						×8 ×8 ×8 ×8) x87SW_ES) x87SW_U) x87SW_D	
	00402060	95 60	tert asy asy				*			
	<						> Defa	ault (stdcall)	-	5 🗘 🗌 Unlock
	32.WriteFile> (7 95E makop.exe:\$3						2: 3: 4: 5:	[esp+C] 0492FC88 [esp+10] 00000000		
Dump 1	💭 Dump 2	Dump 3 🔛 Dump 4	📖 Dump 5 🛛 👹 Watch 1	[x=] Locals 🛛 🖉 Strue	t	0492FC64 0492FC68	02655DD0			
Address He:	x		ASCII			0492FC6C				
02655DD0 DB	EE 88 56 98 E2	16 5F FD 3C 0C FA C	DO F4 0D D1.V.âý<.ú	ÇĐÔ.		0492FC70 0492FC74				
02655DE0 71	9C C1 5E A5 47	D1 FA A6 F1 5C A3 B1	7D 00 EC q.A^¥GÑú¦ñ\£	±}.1		0492FC78				
02655DF0 15	4F 7E D4 B4 FA	45 63 89 87 50 D3 EC	0B 5A C4 .0~0 úEC .]0 09 30 25 .KæX.k; .0%	1.ZA				kernel32.CreateFile	W	
			3 10 B4 D3 .FµgrQQ.			0492FC80 0492FC84				
02655E20 4B	13 D6 5C 3C 9B	5A B9 21 5F D8 33 03	C6 94 38 K.O\<.Z'!_Ø3	.Æ.:			000012372			
			1E 29 DC .0~3UJ5.&			0492FC8C				
02055E40 1B	F5 5E 23 4A F2	4C AB SC D4 SF 81 18	65 F5 68 .0^#JoL«.0	. COK		04925090	000000000			

Figure 67

Makop adds 8 NULL bytes after the encrypted content from above:

	6A 00 8D 44 24 14 50 6A 08 8D 4C 24 34 51 55 FF D6 95 C0	Jush 0 lea eax, dword ptr ss:[esp+14] push 8 lea ecx, dword ptr ss:[esp+34] push ex push ex call esi tert esv esv	esitwr v	xxx/iw_6 s (Empty) xxx/iw_/ s (Empty) x875tatusword 0000 x875w_E 0 x875w_C 2 0 x875w_C 1 0 x875w_C 3 0 x875w_C 2 0 x875w_C 1 0 x875w_C 0 0 x875w_E 5 0 x875w_F 0 x875w_U 0 x875w_F 0 x875w_U 0 x875w_0 0 x875w_Z 0 x875w_U 0 x875w_F 0 x875w_U 0 x875w_F 0 x875w_U 0 x875w_F 0 x875w_U 0 x875w_U
esi= <kernel32.writefile> (76 .text:0040399F makop.exe:\$35</kernel32.writefile>			2 3 4	: [esp] 000002A4
Image: Dump 1 Image: Dump 2 Image: Dump 2 Address Hex 0492FCA0 00 <td< td=""><td>0 00 1C E2 00 00 00 00 00 00</td><td>Watch 1 x= Locals 2 Struct</td><td>0492FC64 000002/ 0492FC68 0492FC 0492FC6C 000000 0492FC70 0492FC7 0492FC74 000000</td><td>A0 08 88</td></td<>	0 00 1C E2 00 00 00 00 00 00	Watch 1 x= Locals 2 Struct	0492FC64 000002/ 0492FC68 0492FC 0492FC6C 000000 0492FC70 0492FC7 0492FC74 000000	A0 08 88

Figure 68

The ransomware reads the content of the file by calling the ReadFile function:

	push 0 lea eax, push eax push eax push ecx mov ecx, dword ptr ds:[ebx+C] push ebp push ebp call dword ptr ds:[<&ReadFile>]	, ×	x875tatusword 0000 x875w_B 0 x875w_C3 0 x875w_C1 0 x875w_C2 0 x875w_C1 0 x875w_P 0 x875w_O 0 x875w_P 0 x875w_O 0 x875w_P 0 Edaut (stdcall) 1: [csp1 000002A4 2: [csp+4] 0472C020 3: [csp+6] 0000021c	
.text:00403A4B makop.exe:\$3A4B #2E4B			4: [esp+C] 0492FC94 5: [esp+10] 00000000	
Jump 1 Jump 2 Jump 3 Jump 4 Jump 5 Address Hex 04725020 000000000000000000000000000000000000	Watch 1 Ix=I Locals 2 Struct 0492 ASCII 0492 0492 0492	2FC64 00000 2FC68 04720 2FC6C 00008 2FC70 0492F 2FC74 00000	C020 E21C FC94	

Figure 69

The file content is encrypted using **AESKey1:**



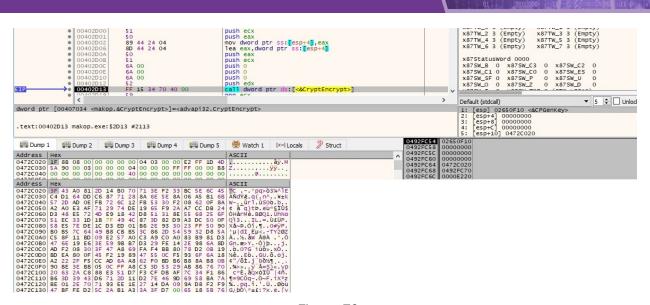


Figure 70

The encrypted content is written to the file:

CONSTRAINTS CONSTRAIN	push 0 lea edx, dword ptr ss:[esp+14] push edx add eax,esi push eax push ecx push ecx call edi tert eax eax	edi:Wr v	X8/1W_0 3 (EmpLy) X8/1W_1 3 (EmpLy) X875LatusWord 0000 X875W_E0 0 X875W_E0 0 X875W_C0 0 X875W_E2 0 X875W_E5 0 X875W_51 0 X875W_F0 0 X875W_F0 0 X875W_51 0 X875W_51 0 X875W_F0 0 X875W_F0 0 X875W_51 0 X875W_51 0 X875W_F0 0 X875W_F0 0 X875W_51 0 Default (stdcall)
edi= <kernel32.writefile> (76ACE250) .text:00403AD9 makop.exe:\$3AD9 #2ED9</kernel32.writefile>			1: [ssp] 000002A4 2: [ssp+4] 0472C020 3: [ssp+4] 0000E220 4: [ssp+C] 0492FC88 5: [ssp+10] 0000000
Image: Second	Watch 1 Ix= Locals Struct ASCII A 5 PC * pg>03¼^1E	0492FC64 00000 0492FC68 0472C 0492FC6C 0000E 0492FC70 0492F 0492FC74 00000	020 220 CS8

Figure 71

The file extension is changed to show that the file has been encrypted:

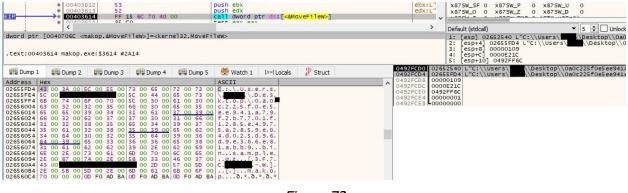


Figure 72

The ransom note name and content are also decrypted at runtime:



Address	Hex												1	ASCII							
And and a state of the local division of the	-		65	00	64	00	6.4	00	60	00	65	00	20	00		00	And the second se				
02650F10				00										00			r.e.a.d.m.ew.				
02650F20	61											00		00			a.r.n.i.n.gt.				
	1		74	00	00	00	00	00	00	00	00	00	00	00	00	00	x.t				
Address	He	<	_						_					_	_		ASCII				
02651458	59	00	4F	00	55	00	52	00	5F	00	46	00	49	00	4C	00	Y.O.U.RF.I.L.				
02651468	45	00	53	00	5F	00	41	00	52	00	45	00	5F	00	45	00	E.SA.R.EE.				
02651478	4F.	00	43	00	52	00	59	00	50	00	54	00	45	00	44	00	N.C.R.Y.P.T.F.D.				
Address	He	ĸ															ASCII				
02653830	41	6C	6C	20	6F	66	20	79	6F	75	72	20	66	69	6C	65	All of your file				
02653840	73	20	68	61	76	65	20	62	65	65	6E	20	65	6E	63	72	s have been encr				
02653850	79	70	74	65	64	2E	20	59	6F	75	72	20	62	61	63	6B	vpted. Your back				
02653860	75	70	20	66	69	6C	65	73	20	61	73	20	77	65	6C	6C	up files as well				
02653870	2E	20	57	65	20	68	61	76	65	20	65	78	66	69	6C	74	. We have exfilt				
02653880	72	61	74	65	64	20	74	6F	6E	73	20	6F	66	20	79	6F	rated tons of yo				
02653890	75	72	20	70	72	69	76	61	74	65	20	64	61	74	61	20	ur private data				
026538A0	74	6F	20	6F	75	72	20	73	65	72	76	65	72	73	20	69	to our servers i				
026538B0	6E	63	6C	75	64	69	6E	67	20	64	61	74	61	20	6F	66	ncluding data of				
026538C0	20	79	6F	75	72	20	63	6C	69	65	6E	74	73	2C	20	64	vour clients. d				
026538D0	6F	6E	74	20	62	65	GC	69	65	76	65	20	75	73	20	3F	ont believe us ?				
026538E0	20	52	65	61	64	20	6F	6E	2E	20	49	6E	20	6F	72	64	Read on. In ord				
026538F0	65	72	20	74	6F	20	72	65	73	74	6F	72	65	20	79	6F	er to restore vo				
02653900		72	20	6F	70	65	72	61	74	69	6F	6E	73	2C	20	61	ur operations, a				
02653910	76	6F	69	64	20	6C	65	61		69				73		6C	void leaking/sel				
02653920	6C	69	6E	67	20	79				20				61		20	ling your data,				
02653930	61	6E	64	20	6B	65										75	and keep your bu				
02653940							20										siness reputatio				
																21					

Figure 73

The ransom note that contains the personal ID is shown below:

i igure y i

It's important to mention that **AESKey1** and **AESKey2** are successively used to encrypt files.

RUNNING WITH "-n" PARAMETER

Makop verifies that the Process ID that comes with the "-n" parameter is composed of digits only:



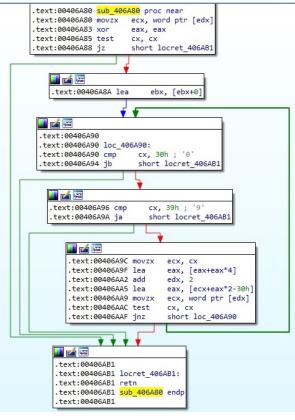


Figure 75

The binary creates a new thread that will enumerate the network resources:

EIP 00405703 GA 00 00405705 GA 00 00405707 FF 15 F4 70	• 004057C8 GA 00 push 0 • 004057C0 52 push edx • 004057C1 52 push edx • 004057C3 6A 00 push edx • 004057C5 6A 00 push 0 • 004057C5 6A 00 push 0 • 004057C5 6A 00 push 0 • 00405705 6A 00 push 0 • 00405705 FE 15 F4 70 40 00 call dword ptr ds: [<&CreateThread>] • 00405705 FE 15 F4 70 40 00 call dword ptr ds: [<&CreateThread>] • 00405705 FE 15 F4 70 40 00 call dword ptr ds: [<&CreateThread>]							
Jump 1 Jump 2 Jump 3 Jump 3 Address Hex	ASCII	t 0019FF20 00000000 0019FF24 0000000 0019FF25 00405080 0019FF2C 02635EE0	0 makop.004050B0					
02646D88 25 00 73 00 20 00 28 00 25 00 30 02646D98 29 00 25 00 63 00 20 00 25 00 49 02646D98 64 00 25 00 65 00 20 00 25 00 49		0019FF2C 02635EE0 0019FF30 00000000 0019FF34 00000000	D					



The WNetOpenEnumW and WNetEnumResourceW APIs are used to enumerate the network resources. The malicious executable is looking for network shares that will also be encrypted:



EIR	 00404F50 00404F51 00404F52 00404F54 00404F56 00404F56 00404F57 	50 51 6A 00 6A 00 57 E8 5A 1E 00 00	push eax push ecx push 0 push di call <makop.wwetop< th=""><th>enEnumW></th><th></th><th>XE</th><th>37StatusWord 0000 37SW_B 0 x87SW_C3 37SW_C1 0 x87SW_C0 37SW_SF 0 x87SW_P 57SW_SF 0 x87SW_P 57SW_SF 0 x87SW_P</th><th>0</th><th>x875W_U</th><th>0 0 0 5 🗘 Unlock</th></makop.wwetop<>	enEnumW>		XE	37StatusWord 0000 37SW_B 0 x87SW_C3 37SW_C1 0 x87SW_C0 37SW_SF 0 x87SW_P 57SW_SF 0 x87SW_P 57SW_SF 0 x87SW_P	0	x875W_U	0 0 0 5 🗘 Unlock
	NetOpenEnumW>	F57 #4357					[esp] 00000001 [esp+4] 00000000 [esp+8] 00000000 [esp+C] 00000000 [esp+10] 046DFF44			
Address 02646D88			ASCII	ss:[esp+18]	0460FF20 00000 0460FF20 00000 0460FF24 00000 0460FF28 00000 0460FF28 0460F	0000 0000 F 44	875tatusword 0000 875w_B 0 x875w_C0 875w_C1 0 x875w_C0 875w_SF 0 x875w_P 55w_SF 0 x875w_P	0	x875W_U	0 0 0 5 🔄 🗌 Unlod
	NetEnumResourceW>	4FA2 #43A2 Dump 3	🚛 Dump 5 🛞 Watch 1 💷 Li	ocals 🖉 Struct	0460FF20 02645	2: 3: 4: 5:	[esp] 0264E4D0 "CO [esp+4] 046DFF40 [esp+8] 02648140 [esp+C] 046DFF48 [esp+10] 02635EE0 "CONN")nn"		
Address			ASCII		046DFF24 046DF 046DFF28 02648 046DFF2C 046DF	3140				

Figure 77



APPENDIX

List of processes to be stopped

msftesql.exe sqlagent.exe sqlbrowser.exe sqlservr.exe sqlwriter.exe oracle.exe ocssd.exe dbsnmp.exe synctime.exe

agntsrvc.exe

mydesktopqos.exe

isqlplussvc.exe

xfssvccon.exe

mydesktopservice.exe

ocautoupds.exe

encsvc.exe

firefoxconfig.exe

tbirdconfig.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

steam.exe

thebat.exe

thebat64.exe

thunderbird.exe

visio.exe

winword.exe

wordpad.exe

