A step-by-step analysis of a new version of Darkside Ransomware (v. 2.1.2.3)

cybergeeks.tech/a-step-by-step-analysis-of-a-new-version-of-darkside-ransomware/

Summary

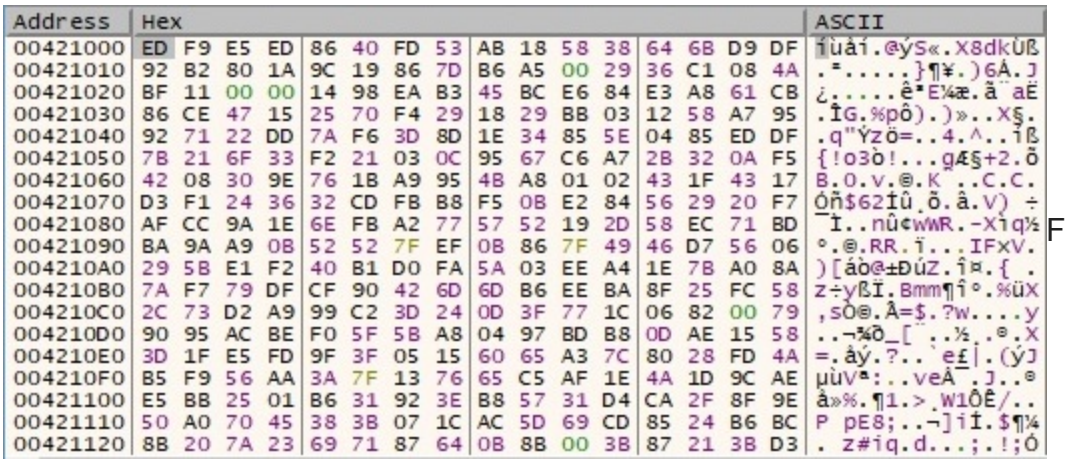
Darkside ransomware is the malware family responsible for the Colonial Pipeline attack on May 7 2021 as described at https://www.zdnet.com/article/darkside-the-ransomware-group-responsible-for-colonial-pipeline-cyberattack-explained/. The binary contains an encrypted configuration that will be decrypted using a custom algorithm, which reveals a 22-byte buffer that describes different actions performed by the malware. These actions include: checking the system language and avoiding to encrypt Russian language machines, deleting Shadow copies, wiping Recycle Bin, ignore specific files, directories and file extensions, killing specific processes, deleting specific services, etc. The ransomware can perform privilege escalation using the CMSTPLUA COM interface and achieves persistence by installing itself as a service. The files are encrypted using the custom Salsa20 implementation, with the Salsa20 matrix being encrypted by the public RSA key hard-coded in the binary. Darkside uses multithreading with I/O completion ports to communicate between the main thread and the worker threads responsible for file encryptions. It's important to mention that the process generates a random Salsa20 matrix using the RDRAND and RDSEED instructions, as opposed to earlier versions that use the RtlRandomEx function.

Analyst: @GeeksCyber

Technical analysis

SHA256:
0A0C225F0E5EE941A79F2B7701F1285E4975A2859EB4D025D96D9E366E81ABB9

The malware comes with an encrypted configuration that is decrypted using a custom algorithm:



Address	Hex																ASCII
00421000	ED	F9	E5	ED	86	40	FD	53	AB	18	58	38	64	6B	D9	DF	íùåí.@ýS«.X8dkÙß
00421010	92	B2	80	1A	9C	19	86	7D	B6	A5	00	29	36	C1	08	4A	.ª.....}¶¥.)6Á.J
00421020	BF	11	00	00	14	98	EA	B3	45	BC	E6	84	E3	A8	61	CB	¿.....ê³E¼æ.ã¨aË
00421030	86	CE	47	15	25	70	F4	29	18	29	BB	03	12	58	A7	95	.ÎG.%pô).)»..X§.
00421040	92	71	22	DD	7A	F6	3D	8D	1E	34	85	5E	04	85	ED	DF	.q"Ýzö=..4.^..íß
00421050	7B	21	6F	33	F2	21	03	0C	95	67	C6	A7	2B	32	0A	F5	{!o3ò!...gÆ§+2.õ
00421060	42	08	30	9E	76	1B	A9	95	4B	A8	01	02	43	1F	43	17	B.0.v.©.K¨..C.C.
00421070	D3	F1	24	36	32	CD	FB	B8	F5	0B	E2	84	56	29	20	F7	Óñ$62Íû¸õ.â.V) ÷
00421080	AF	CC	9A	1E	6E	FB	A2	77	57	52	19	2D	58	EC	71	BD	¯Ì..nû¢wWR.-Xìq½
00421090	BA	9A	A9	0B	52	52	7F	EF	0B	86	7F	49	46	D7	56	06	º.©.RR.ï...IF×V.
004210A0	29	5B	E1	F2	40	B1	D0	FA	5A	03	EE	A4	1E	7B	A0	8A	)[áò@±ÐúZ.î¤.{ .
004210B0	7A	F7	79	DF	CF	90	42	6D	6D	B6	EE	BA	8F	25	FC	58	z÷yßÏ.Bmm¶îº.%üX
004210C0	2C	73	D2	A9	99	C2	3D	24	0D	3F	77	1C	06	82	00	79	,sÒ©.Â=$.?w....y
004210D0	90	95	AC	BE	F0	5F	5B	A8	04	97	BD	B8	0D	AE	15	58	..¬¾ð_[¨..½¸.®.X
004210E0	3D	1F	E5	FD	9F	3F	05	15	60	65	A3	7C	80	28	FD	4A	=.åý.?..`e£|.(ýJ
004210F0	B5	F9	56	AA	3A	7F	13	76	65	C5	AF	1E	4A	1D	9C	AE	µùVª:..veÅ¯.J..®
00421100	E5	BB	25	01	B6	31	92	3E	B8	57	31	D4	CA	2F	8F	9E	å»%.¶1.>¸W1ÔÊ/..
00421110	50	A0	70	45	38	3B	07	1C	AC	5D	69	CD	85	24	B6	BC	P pE8;..¬]iÍ.$¶¼
00421120	8B	20	7A	23	69	71	87	64	0B	8B	00	3B	87	21	3B	D3	. z#iq.d...;.!;Ó

Figure 1

The custom decryption algorithm consists of 4 subtraction operations by 0x10101010 each time and then some addition operations, as shown below:

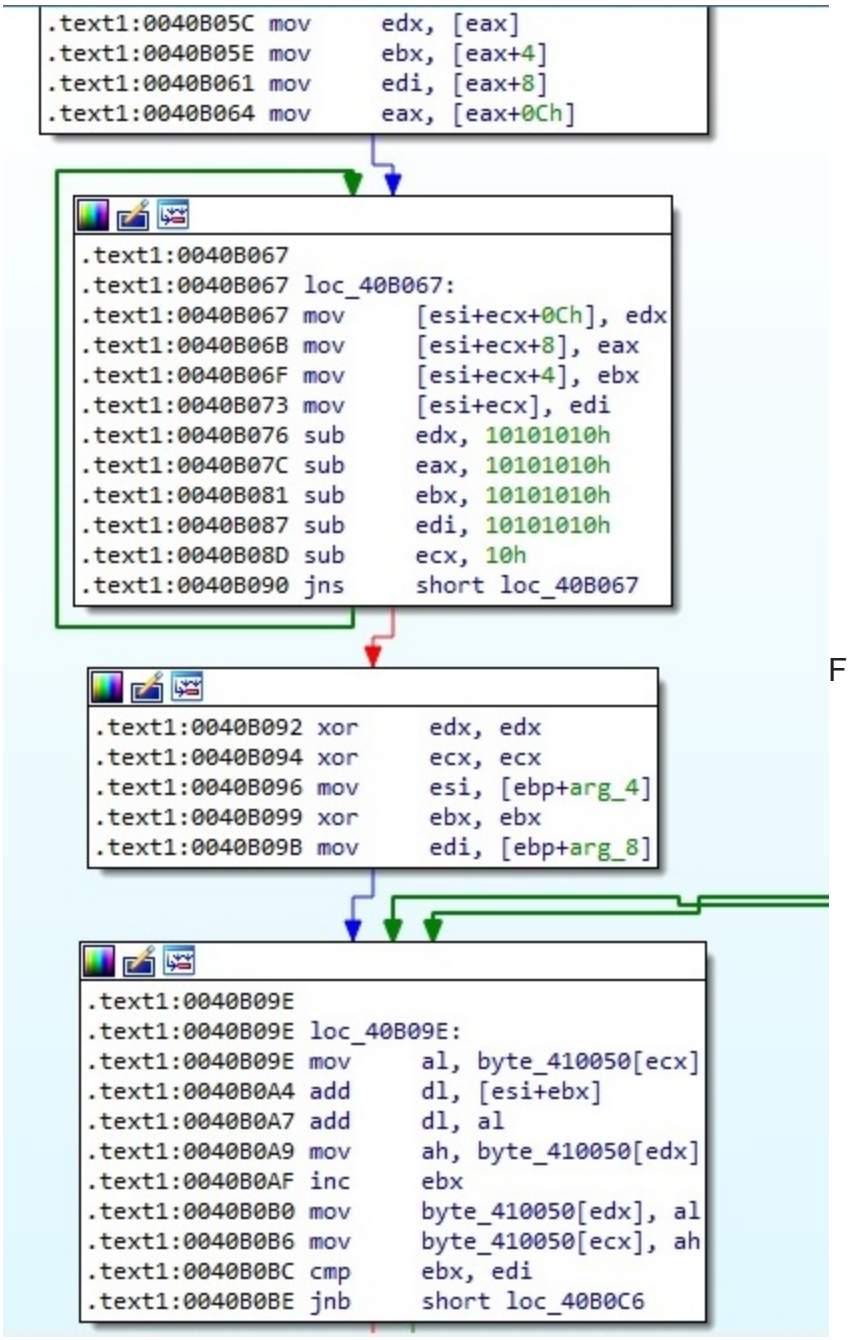


Figure 2

For each DLL to be loaded, there is a hash function that is applied to the DLL name, and the 4-byte result is compared to hardcoded values:

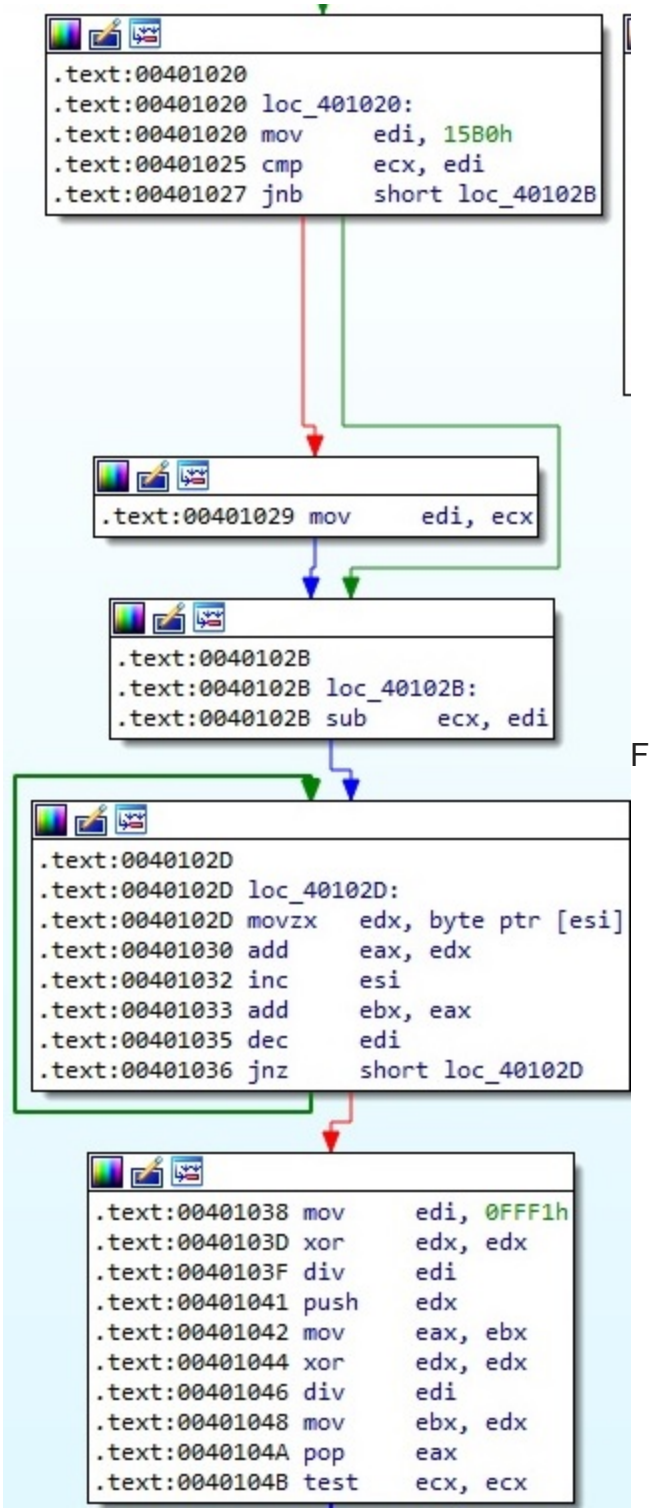


Figure 3

For example, the following value corresponds to kernel32.dll:

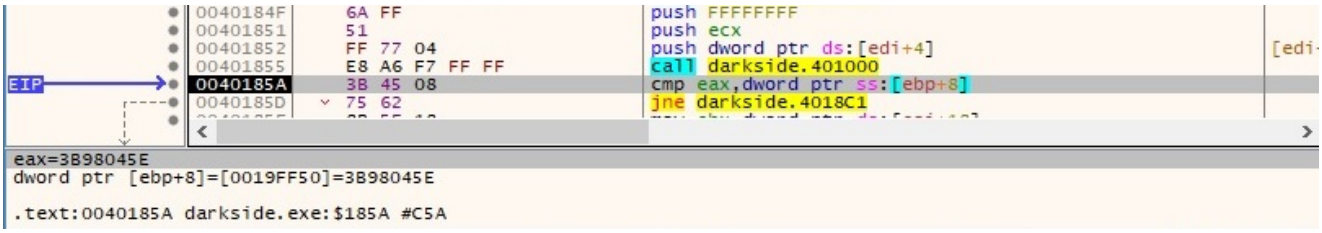


Figure 4

The following DLLs are expected to be loaded: ntdll, kernel32, advapi32, user32, gdi32, ole32, oleaut32, shell32, shlwapi, wininet, netapi32, wtsapi32, activeds, userenv, mpr, rstrtmgr. The process retrieves the address of multiple export functions based on similar hash values computed using the same algorithm:

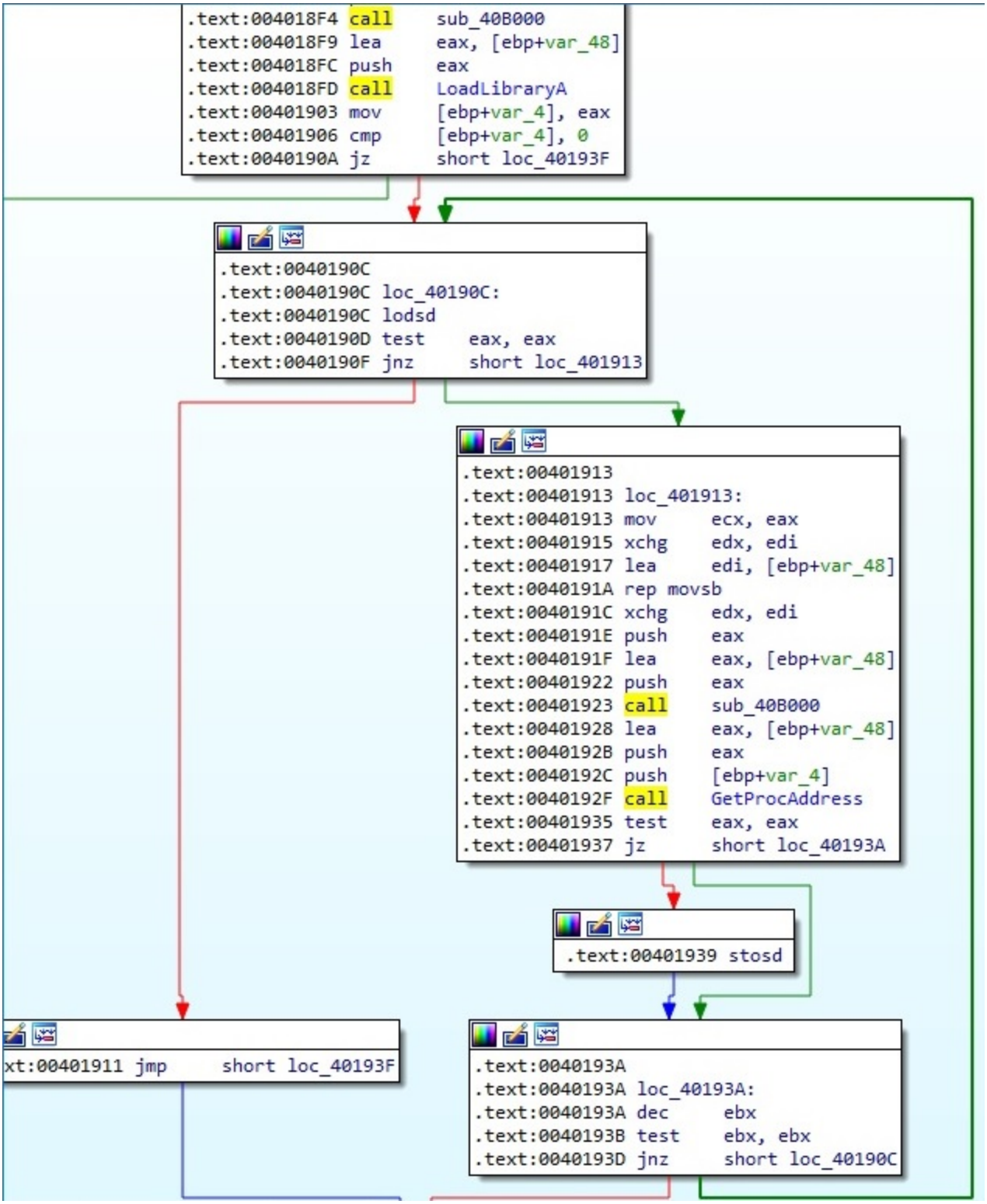
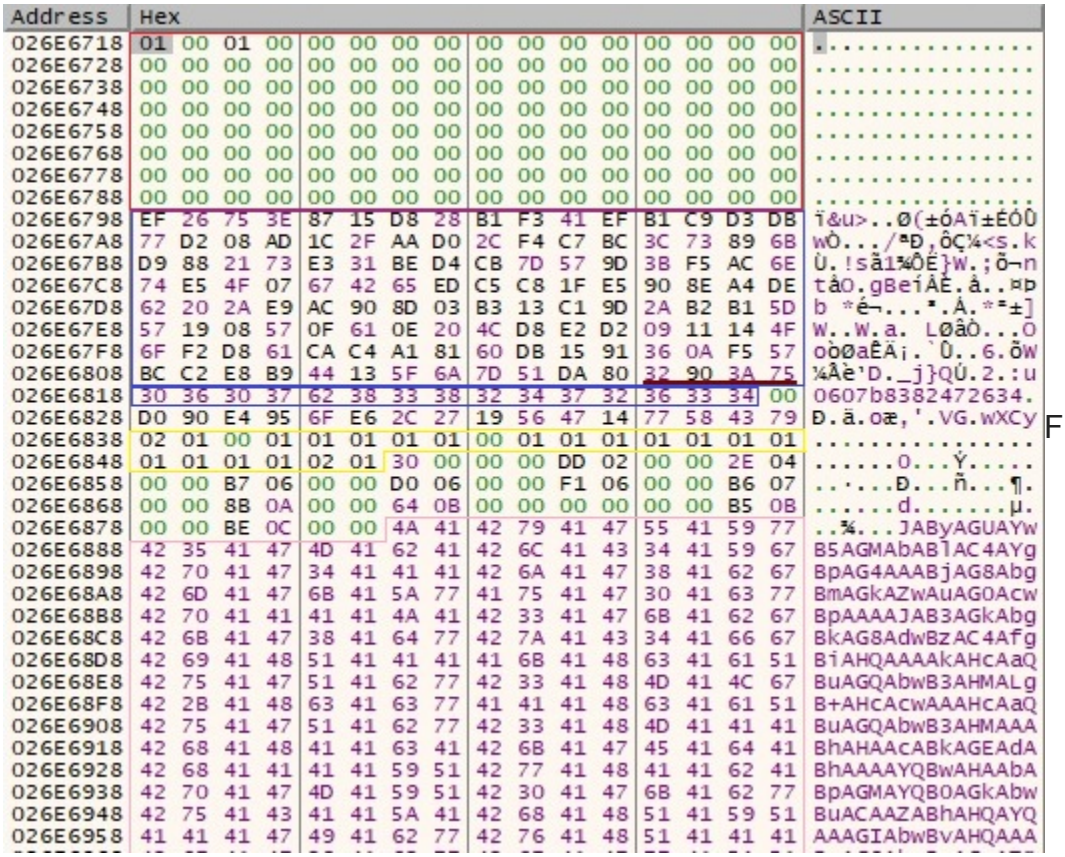


Figure 5

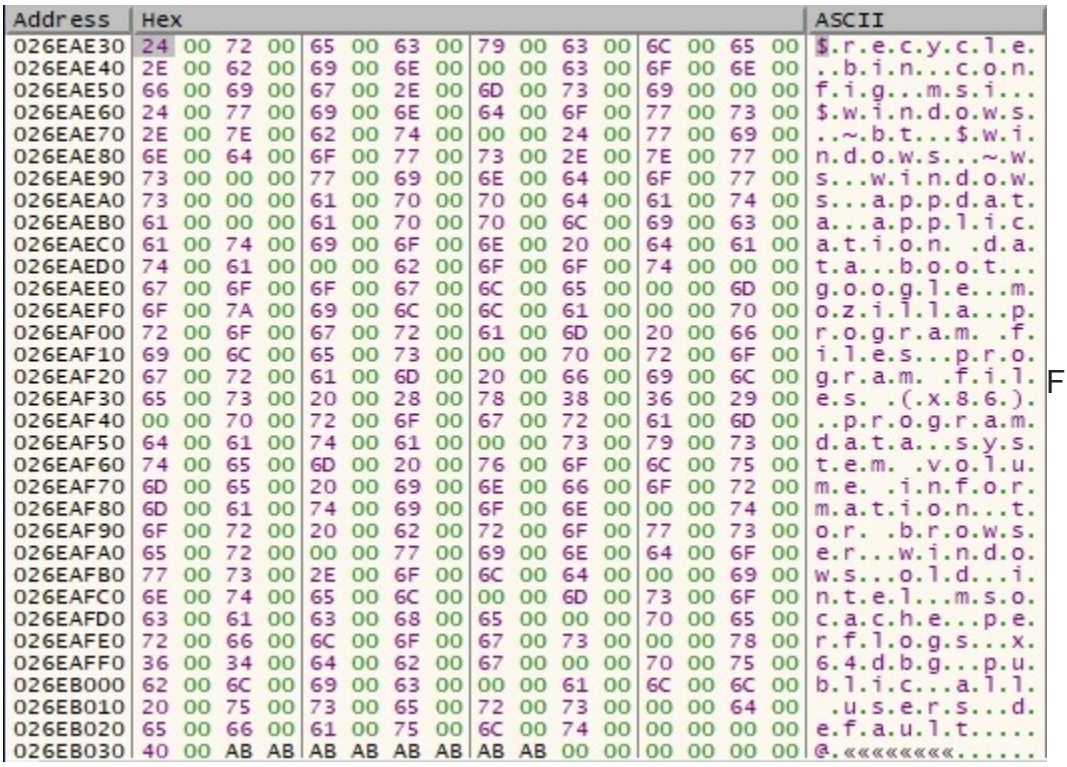
The decrypted configuration is presented below and is composed of the RSA-1024 exponent (0x010001 = 65537), 0x80-byte RSA-1024 modulus, victim UID, 22 configurations bytes (will be detailed further on) and the aPLib-compressed configuration:



Address	Hex	ASCII
026E6718	01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
026E6728	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
026E6738	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
026E6748	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
026E6758	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
026E6768	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
026E6778	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
026E6788	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
026E6798	EF 26 75 3E 87 15 D8 28 B1 F3 41 EF B1 C9 D3 DB	ï&u>..Ø(±óAï±ÉÓÛ
026E67A8	77 D2 08 AD 1C 2F AA D0 2C F4 C7 BC 3C 73 89 6B	wÒ.../ªÐ,ôÇ¼<s.k
026E67B8	D9 88 21 73 E3 31 BE D4 CB 7D 57 9D 3B F5 AC 6E	Ù.!sã1¾ÔË}W.;õ¬n
026E67C8	74 E5 4F 07 67 42 65 ED C5 C8 1F E5 90 8E A4 DE	tåO.gBeíÅÈ.å..¤Þ
026E67D8	62 20 2A E9 AC 90 8D 03 B3 13 C1 9D 2A B2 B1 5D	b *é¬...³.Á.*²±]
026E67E8	57 19 08 57 0F 61 0E 20 4C D8 E2 D2 09 11 14 4F	W..W.a. LØâÒ...O
026E67F8	6F F2 D8 61 CA C4 A1 81 60 DB 15 91 36 0A F5 57	oòØaÊÄ¡.`Û..6.õW
026E6808	BC C2 E8 B9 44 13 5F 6A 7D 51 DA 80 32 90 3A 75	¼Âè¹D._j}QÚ.2.:u
026E6818	30 36 30 37 62 38 33 38 32 34 37 32 36 33 34 00	0607b8382472634.
026E6828	D0 90 E4 95 6F E6 2C 27 19 56 47 14 77 58 43 79	Ð.ä.oæ,'.VG.wXCy
026E6838	02 01 00 01 01 01 01 01 00 01 01 01 01 01 01 01	.....
026E6848	01 01 01 01 02 01 30 00 00 00 DD 02 00 00 2E 04	......0...Ý.....
026E6858	00 00 B7 06 00 00 D0 06 00 00 F1 06 00 00 B6 07	..·...Ð...ñ...¶.
026E6868	00 00 8B 0A 00 00 64 0B 00 00 00 00 00 00 B5 0B	......d.......µ.
026E6878	00 00 BE 0C 00 00 4A 41 42 79 41 47 55 41 59 77	..¾...JAByAGUAYw
026E6888	42 35 41 47 4D 41 62 41 42 6C 41 43 34 41 59 67	B5AGMAbABlAC4AYg
026E6898	42 70 41 47 34 41 41 41 42 6A 41 47 38 41 62 67	BpAG4AAABjAG8Abg
026E68A8	42 6D 41 47 6B 41 5A 77 41 75 41 47 30 41 63 77	BmAGkAZwAuAG0Acw
026E68B8	42 70 41 41 41 41 4A 41 42 33 41 47 6B 41 62 67	BpAAAAJAB3AGkAbg
026E68C8	42 6B 41 47 38 41 64 77 42 7A 41 43 34 41 66 67	BkAG8AdwBzAC4Afg
026E68D8	42 69 41 48 51 41 41 41 41 6B 41 48 63 41 61 51	BiAHQAAAAkAHcAaQ
026E68E8	42 75 41 47 51 41 62 77 42 33 41 48 4D 41 4C 67	BuAGQAbwB3AHMALg
026E68F8	42 2B 41 48 63 41 63 77 41 41 41 48 63 41 61 51	B+AHcAcwAAAHcAaQ
026E6908	42 75 41 47 51 41 62 77 42 33 41 48 4D 41 41 41	BuAGQAbwB3AHMAAA
026E6918	42 68 41 48 41 41 63 41 42 6B 41 47 45 41 64 41	BhAHAAcABkAGEAdA
026E6928	42 68 41 41 41 41 59 51 42 77 41 48 41 41 62 41	BhAAAAYQBwAHAAbA
026E6938	42 70 41 47 4D 41 59 51 42 30 41 47 6B 41 62 77	BpAGMAYQB0AGkAbw
026E6948	42 75 41 43 41 41 5A 41 42 68 41 48 51 41 59 51	BuACAAZABhAHQAYQ
026E6958	41 41 41 47 49 41 62 77 42 76 41 48 51 41 41 41	AAAGIAbwBvAHQAAA

Figure 6

The binary uses an aPLib-decompression algorithm to decrypt different strings. The following list represents the directories to avoid in the encryption process:



Address	Hex	ASCII
026EAE30	24 00 72 00 65 00 63 00 79 00 63 00 6C 00 65 00	$.r.e.c.y.c.l.e.
026EAE40	2E 00 62 00 69 00 6E 00 00 00 63 00 6F 00 6E 00	..b.i.n...c.o.n.
026EAE50	66 00 69 00 67 00 2E 00 6D 00 73 00 69 00 00 00	f.i.g...m.s.i...
026EAE60	24 00 77 00 69 00 6E 00 64 00 6F 00 77 00 73 00	$.w.i.n.d.o.w.s.
026EAE70	2E 00 7E 00 62 00 74 00 00 00 24 00 77 00 69 00	..~.b.t...$.w.i.
026EAE80	6E 00 64 00 6F 00 77 00 73 00 2E 00 7E 00 77 00	n.d.o.w.s...~.w.
026EAE90	73 00 00 00 77 00 69 00 6E 00 64 00 6F 00 77 00	s...w.i.n.d.o.w.
026EAEA0	73 00 00 00 61 00 70 00 70 00 64 00 61 00 74 00	s...a.p.p.d.a.t.
026EAEB0	61 00 00 00 61 00 70 00 70 00 6C 00 69 00 63 00	a...a.p.p.l.i.c.
026EAEC0	61 00 74 00 69 00 6F 00 6E 00 20 00 64 00 61 00	a.t.i.o.n. .d.a.
026EAED0	74 00 61 00 00 00 62 00 6F 00 6F 00 74 00 00 00	t.a...b.o.o.t...
026EAEE0	67 00 6F 00 6F 00 67 00 6C 00 65 00 00 00 6D 00	g.o.o.g.l.e...m.
026EAEF0	6F 00 7A 00 69 00 6C 00 6C 00 61 00 00 00 70 00	o.z.i.l.l.a...p.
026EAF00	72 00 6F 00 67 00 72 00 61 00 6D 00 20 00 66 00	r.o.g.r.a.m. .f.
026EAF10	69 00 6C 00 65 00 73 00 00 00 70 00 72 00 6F 00	i.l.e.s...p.r.o.
026EAF20	67 00 72 00 61 00 6D 00 20 00 66 00 69 00 6C 00	g.r.a.m. .f.i.l.
026EAF30	65 00 73 00 20 00 28 00 78 00 38 00 36 00 29 00	e.s. .(.x.8.6.).
026EAF40	00 00 70 00 72 00 6F 00 67 00 72 00 61 00 6D 00	..p.r.o.g.r.a.m.
026EAF50	64 00 61 00 74 00 61 00 00 00 73 00 79 00 73 00	d.a.t.a...s.y.s.
026EAF60	74 00 65 00 6D 00 20 00 76 00 6F 00 6C 00 75 00	t.e.m. .v.o.l.u.
026EAF70	6D 00 65 00 20 00 69 00 6E 00 66 00 6F 00 72 00	m.e. .i.n.f.o.r.
026EAF80	6D 00 61 00 74 00 69 00 6F 00 6E 00 00 00 74 00	m.a.t.i.o.n...t.
026EAF90	6F 00 72 00 20 00 62 00 72 00 6F 00 77 00 73 00	o.r. .b.r.o.w.s.
026EAFA0	65 00 72 00 00 00 77 00 69 00 6E 00 64 00 6F 00	e.r...w.i.n.d.o.
026EAFB0	77 00 73 00 2E 00 6F 00 6C 00 64 00 00 00 69 00	w.s...o.l.d...i.
026EAFC0	6E 00 74 00 65 00 6C 00 00 00 6D 00 73 00 6F 00	n.t.e.l...m.s.o.
026EAFD0	63 00 61 00 63 00 68 00 65 00 00 00 70 00 65 00	c.a.c.h.e...p.e.
026EAFE0	72 00 66 00 6C 00 6F 00 67 00 73 00 00 00 78 00	r.f.l.o.g.s...x.
026EAFF0	36 00 34 00 64 00 62 00 67 00 00 00 70 00 75 00	6.4.d.b.g...p.u.
026EB000	62 00 6C 00 69 00 63 00 00 00 61 00 6C 00 6C 00	b.l.i.c...a.l.l.
026EB010	20 00 75 00 73 00 65 00 72 00 73 00 00 00 64 00	.u.s.e.r.s...d.
026EB020	65 00 66 00 61 00 75 00 6C 00 74 00 00 00 00 00	e.f.a.u.l.t.....
026EB030	40 00 AB AB AB AB AB AB AB AB 00 00 00 00 00 00	@.«««««««««.....

Figure 7

The following files will be ignored by the ransomware:

Address	Hex	ASCII
026EB050	61 00 75 00 74 00 6F 00 72 00 75 00 6E 00 2E 00	a.u.t.o.r.u.n...
026EB060	69 00 6E 00 66 00 00 00 62 00 6F 00 6F 00 74 00	i.n.f...b.o.o.t.
026EB070	2E 00 69 00 6E 00 69 00 00 00 62 00 6F 00 6F 00	..i.n.i...b.o.o.
026EB080	74 00 66 00 6F 00 6E 00 74 00 2E 00 62 00 69 00	t.f.o.n.t...b.i.
026EB090	6E 00 00 00 62 00 6F 00 6F 00 74 00 73 00 65 00	n...b.o.o.t.s.e.
026EB0A0	63 00 74 00 2E 00 62 00 61 00 6B 00 00 00 64 00	c.t...b.a.k...d.
026EB0B0	65 00 73 00 6B 00 74 00 6F 00 70 00 2E 00 69 00	e.s.k.t.o.p...i.
026EB0C0	6E 00 69 00 00 00 69 00 63 00 6F 00 6E 00 63 00	n.i...i.c.o.n.c.
026EB0D0	61 00 63 00 68 00 65 00 2E 00 64 00 62 00 00 00	a.c.h.e...d.b...
026EB0E0	6E 00 74 00 6C 00 64 00 72 00 00 00 6E 00 74 00	n.t.l.d.r...n.t.
026EB0F0	75 00 73 00 65 00 72 00 2E 00 64 00 61 00 74 00	u.s.e.r...d.a.t.
026EB100	00 00 6E 00 74 00 75 00 73 00 65 00 72 00 2E 00	..n.t.u.s.e.r...
026EB110	64 00 61 00 74 00 2E 00 6C 00 6F 00 67 00 00 00	d.a.t...l.o.g...
026EB120	6E 00 74 00 75 00 73 00 65 00 72 00 2E 00 69 00	n.t.u.s.e.r...i.
026EB130	6E 00 69 00 00 00 74 00 68 00 75 00 6D 00 62 00	n.i...t.h.u.m.b.
026EB140	73 00 2E 00 64 00 62 00 00 00 00 00 00 00 AB AB	s...d.b.......««
026EB150	AB AB AB AB AB AB 00 00 00 00 00 00 00 00 00 00	«««««..........
026EB160	54 34 57 3C 17 3C 00 00 C0 00 6D 02 48 3B 6E 02	T4W<.<..À.m.H;n.

Figure 8

If the file's extension belongs to the following list, then the file will not be encrypted by the process:

Address	Hex	ASCII
026EB168	33 00 38 00 36 00 00 00 61 00 64 00 76 00 00 00	3.8.6...a.d.v...
026EB178	61 00 6E 00 69 00 00 00 62 00 61 00 74 00 00 00	a.n.i...b.a.t...
026EB188	62 00 69 00 6E 00 00 00 63 00 61 00 62 00 00 00	b.i.n...c.a.b...
026EB198	63 00 6D 00 64 00 00 00 63 00 6F 00 6D 00 00 00	c.m.d...c.o.m...
026EB1A8	63 00 70 00 6C 00 00 00 63 00 75 00 72 00 00 00	c.p.l...c.u.r...
026EB1B8	64 00 65 00 73 00 6B 00 74 00 68 00 65 00 6D 00	d.e.s.k.t.h.e.m.
026EB1C8	65 00 70 00 61 00 63 00 6B 00 00 00 64 00 69 00	e.p.a.c.k...d.i.
026EB1D8	61 00 67 00 63 00 61 00 62 00 00 00 64 00 69 00	a.g.c.a.b...d.i.
026EB1E8	61 00 67 00 63 00 66 00 67 00 00 00 64 00 69 00	a.g.c.f.g...d.i.
026EB1F8	61 00 67 00 70 00 6B 00 67 00 00 00 64 00 6C 00	a.g.p.k.g...d.l.
026EB208	6C 00 00 00 64 00 72 00 76 00 00 00 65 00 78 00	l...d.r.v...e.x.
026EB218	65 00 00 00 68 00 6C 00 70 00 00 00 69 00 63 00	e...h.l.p...i.c.
026EB228	6C 00 00 00 69 00 63 00 6E 00 73 00 00 00 69 00	l...i.c.n.s...i.
026EB238	63 00 6F 00 00 00 69 00 63 00 73 00 00 00 69 00	c.o...i.c.s...i.
026EB248	64 00 78 00 00 00 6C 00 64 00 66 00 00 00 6C 00	d.x...l.d.f...l.
026EB258	6E 00 6B 00 00 00 6D 00 6F 00 64 00 00 00 6D 00	n.k...m.o.d...m.
026EB268	70 00 61 00 00 00 6D 00 73 00 63 00 00 00 6D 00	p.a...m.s.c...m.
026EB278	73 00 70 00 00 00 6D 00 73 00 73 00 74 00 79 00	s.p...m.s.s.t.y.
026EB288	6C 00 65 00 73 00 00 00 6D 00 73 00 75 00 00 00	l.e.s...m.s.u...
026EB298	6E 00 6C 00 73 00 00 00 6E 00 6F 00 6D 00 65 00	n.l.s...n.o.m.e.
026EB2A8	64 00 69 00 61 00 00 00 6F 00 63 00 78 00 00 00	d.i.a...o.c.x...
026EB2B8	70 00 72 00 66 00 00 00 70 00 73 00 31 00 00 00	p.r.f...p.s.1...
026EB2C8	72 00 6F 00 6D 00 00 00 72 00 74 00 70 00 00 00	r.o.m...r.t.p...
026EB2D8	73 00 63 00 72 00 00 00 73 00 68 00 73 00 00 00	s.c.r...s.h.s...
026EB2E8	73 00 70 00 6C 00 00 00 73 00 79 00 73 00 00 00	s.p.l...s.y.s...
026EB2F8	74 00 68 00 65 00 6D 00 65 00 00 00 74 00 68 00	t.h.e.m.e...t.h.
026EB308	65 00 6D 00 65 00 70 00 61 00 63 00 6B 00 00 00	e.m.e.p.a.c.k...
026EB318	77 00 70 00 78 00 00 00 6C 00 6F 00 63 00 6B 00	w.p.x...l.o.c.k.
026EB328	00 00 6B 00 65 00 79 00 00 00 68 00 74 00 61 00	..k.e.y...h.t.a.
026EB338	00 00 6D 00 73 00 69 00 00 00 70 00 64 00 62 00	..m.s.i...p.d.b.

Figure 9

The binary intends to delete folders that contain the word "backup" in their name:

Address	Hex	ASCII
026E3000	62 00 61 00 63 00 6B 00 75 00 70 00 00 00 00 00	b.a.c.k.u.p.....

Figure 10

A feature not used by the malware would use the following strings decompressed as the other ones (our guess is that the actor would try to kill the SQL-related processes in order to encrypt databases):

Address	Hex	ASCII
026E4EB8	73 00 71 00 6C 00 00 00 73 00 71 00 6C 00 69 00	s.q.l...s.q.l.i.
026E4EC8	74 00 65 00 00 00 00 00 00 00 AB AB AB AB AB AB	t.e.......««««««

Figure 11

The following processes will not be terminated by the file:

Address	Hex	ASCII
026EB558	76 00 6D 00 63 00 6F 00 6D 00 70 00 75 00 74 00	v.m.c.o.m.p.u.t.
026EB568	65 00 2E 00 65 00 78 00 65 00 00 00 76 00 6D 00	e...e.x.e...v.m.
026EB578	6D 00 73 00 2E 00 65 00 78 00 65 00 00 00 76 00	m.s...e.x.e...v.
026EB588	6D 00 77 00 70 00 2E 00 65 00 78 00 65 00 00 00	m.w.p...e.x.e...
026EB598	73 00 76 00 63 00 68 00 6F 00 73 00 74 00 2E 00	s.v.c.h.o.s.t...
026EB5A8	65 00 78 00 65 00 00 00 54 00 65 00 61 00 6D 00	e.x.e...T.e.a.m.
026EB5B8	56 00 69 00 65 00 77 00 65 00 72 00 2E 00 65 00	V.i.e.w.e.r...e.
026EB5C8	78 00 65 00 00 00 65 00 78 00 70 00 6C 00 6F 00	x.e...e.x.p.l.o.
026EB5D8	72 00 65 00 72 00 2E 00 65 00 78 00 65 00 00 00	r.e.r...e.x.e...

Figure 12

If a process name contains any of the following strings, it will be killed by the binary:

Address	Hex	ASCII
026EB608	73 00 71 00 6C 00 00 00 6F 00 72 00 61 00 63 00	s.q.l...o.r.a.c.
026EB618	6C 00 65 00 00 00 6F 00 63 00 73 00 73 00 64 00	l.e...o.c.s.s.d.
026EB628	00 00 64 00 62 00 73 00 6E 00 6D 00 70 00 00 00	..d.b.s.n.m.p...
026EB638	73 00 79 00 6E 00 63 00 74 00 69 00 6D 00 65 00	s.y.n.c.t.i.m.e.
026EB648	00 00 61 00 67 00 6E 00 74 00 73 00 76 00 63 00	..a.g.n.t.s.v.c.
026EB658	00 00 69 00 73 00 71 00 6C 00 70 00 6C 00 75 00	..i.s.q.l.p.l.u.
026EB668	73 00 73 00 76 00 63 00 00 00 78 00 66 00 73 00	s.s.v.c...x.f.s.
026EB678	73 00 76 00 63 00 63 00 6F 00 6E 00 00 00 6D 00	s.v.c.c.o.n...m.
026EB688	79 00 64 00 65 00 73 00 6B 00 74 00 6F 00 70 00	y.d.e.s.k.t.o.p.
026EB698	73 00 65 00 72 00 76 00 69 00 63 00 65 00 00 00	s.e.r.v.i.c.e...
026EB6A8	6F 00 63 00 61 00 75 00 74 00 6F 00 75 00 70 00	o.c.a.u.t.o.u.p.
026EB6B8	64 00 73 00 00 00 65 00 6E 00 63 00 73 00 76 00	d.s...e.n.c.s.v.
026EB6C8	63 00 00 00 66 00 69 00 72 00 65 00 66 00 6F 00	c...f.i.r.e.f.o.
026EB6D8	78 00 00 00 74 00 62 00 69 00 72 00 64 00 63 00	x...t.b.i.r.d.c.
026EB6E8	6F 00 6E 00 66 00 69 00 67 00 00 00 6D 00 79 00	o.n.f.i.g...m.y.
026EB6F8	64 00 65 00 73 00 6B 00 74 00 6F 00 70 00 71 00	d.e.s.k.t.o.p.q.
026EB708	6F 00 73 00 00 00 6F 00 63 00 6F 00 6D 00 6D 00	o.s...o.c.o.m.m.
026EB718	00 00 64 00 62 00 65 00 6E 00 67 00 35 00 30 00	..d.b.e.n.g.5.0.
026EB728	00 00 73 00 71 00 62 00 63 00 6F 00 72 00 65 00	..s.q.b.c.o.r.e.
026EB738	73 00 65 00 72 00 76 00 69 00 63 00 65 00 00 00	s.e.r.v.i.c.e...
026EB748	65 00 78 00 63 00 65 00 6C 00 00 00 69 00 6E 00	e.x.c.e.l...i.n.
026EB758	66 00 6F 00 70 00 61 00 74 00 68 00 00 00 6D 00	f.o.p.a.t.h...m.
026EB768	73 00 61 00 63 00 63 00 65 00 73 00 73 00 00 00	s.a.c.c.e.s.s...
026EB778	6D 00 73 00 70 00 75 00 62 00 00 00 6F 00 6E 00	m.s.p.u.b...o.n.
026EB788	65 00 6E 00 6F 00 74 00 65 00 00 00 6F 00 75 00	e.n.o.t.e...o.u.
026EB798	74 00 6C 00 6F 00 6F 00 6B 00 00 00 70 00 6F 00	t.l.o.o.k...p.o.
026EB7A8	77 00 65 00 72 00 70 00 6E 00 74 00 00 00 73 00	w.e.r.p.n.t...s.
026EB7B8	74 00 65 00 61 00 6D 00 00 00 74 00 68 00 65 00	t.e.a.m...t.h.e.
026EB7C8	62 00 61 00 74 00 00 00 74 00 68 00 75 00 6E 00	b.a.t...t.h.u.n.
026EB7D8	64 00 65 00 72 00 62 00 69 00 72 00 64 00 00 00	d.e.r.b.i.r.d...
026EB7E8	76 00 69 00 73 00 69 00 6F 00 00 00 77 00 69 00	v.i.s.i.o...w.i.
026EB7F8	6E 00 77 00 6F 00 72 00 64 00 00 00 77 00 6F 00	n.w.o.r.d...w.o.
026EB808	72 00 64 00 70 00 61 00 64 00 00 00 6E 00 6F 00	r.d.p.a.d...n.o.
026EB818	74 00 65 00 70 00 61 00 64 00 00 00 00 00 00 00	t.e.p.a.d.......

Figure 13

There is also a list of services to be stopped and deleted, as shown in the figure below:

Address	Hex	ASCII
026EB840	76 00 73 00 73 00 00 00 73 00 71 00 6C 00 00 00	v.s.s...s.q.l...
026EB850	73 00 76 00 63 00 24 00 00 00 6D 00 65 00 6D 00	s.v.c.$...m.e.m.
026EB860	74 00 61 00 73 00 00 00 6D 00 65 00 70 00 6F 00	t.a.s...m.e.p.o.
026EB870	63 00 73 00 00 00 73 00 6F 00 70 00 68 00 6F 00	c.s...s.o.p.h.o.
026EB880	73 00 00 00 76 00 65 00 65 00 61 00 6D 00 00 00	s...v.e.e.a.m...
026EB890	62 00 61 00 63 00 6B 00 75 00 70 00 00 00 47 00	b.a.c.k.u.p...G.
026EB8A0	78 00 56 00 73 00 73 00 00 00 47 00 78 00 42 00	x.V.s.s...G.x.B.
026EB8B0	6C 00 72 00 00 00 47 00 78 00 46 00 57 00 44 00	l.r...G.x.F.W.D.
026EB8C0	00 00 47 00 78 00 43 00 56 00 44 00 00 00 47 00	..G.x.C.V.D...G.
026EB8D0	78 00 43 00 49 00 4D 00 67 00 72 00 00 00 00 00	x.C.I.M.g.r.....

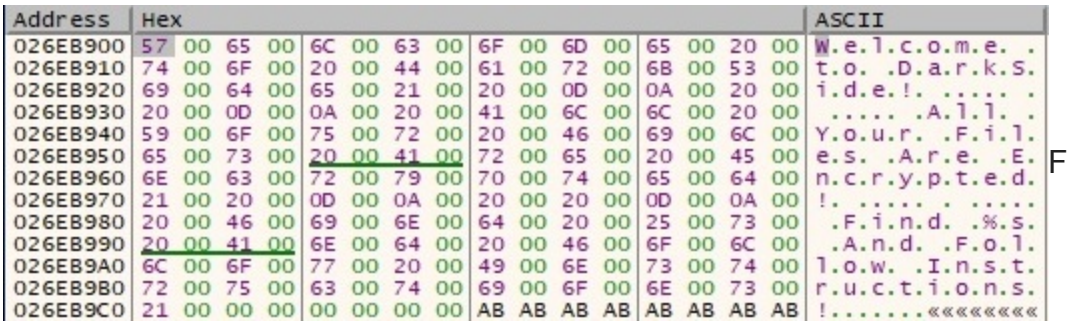
Figure 14

The list of C2 servers is also obtained using the same algorithm:

Address	Hex	ASCII
026E4C00	62 00 61 00 72 00 6F 00 71 00 75 00 65 00 74 00	b.a.r.o.q.u.e.t.
026E4C10	65 00 65 00 73 00 2E 00 63 00 6F 00 6D 00 00 00	e.e.s...c.o.m...
026E4C20	72 00 75 00 6D 00 61 00 68 00 73 00 69 00 61 00	r.u.m.a.h.s.i.a.
026E4C30	2E 00 63 00 6F 00 6D 00 00 00 00 00 00 00 AB AB	..c.o.m.......««

Figure 15

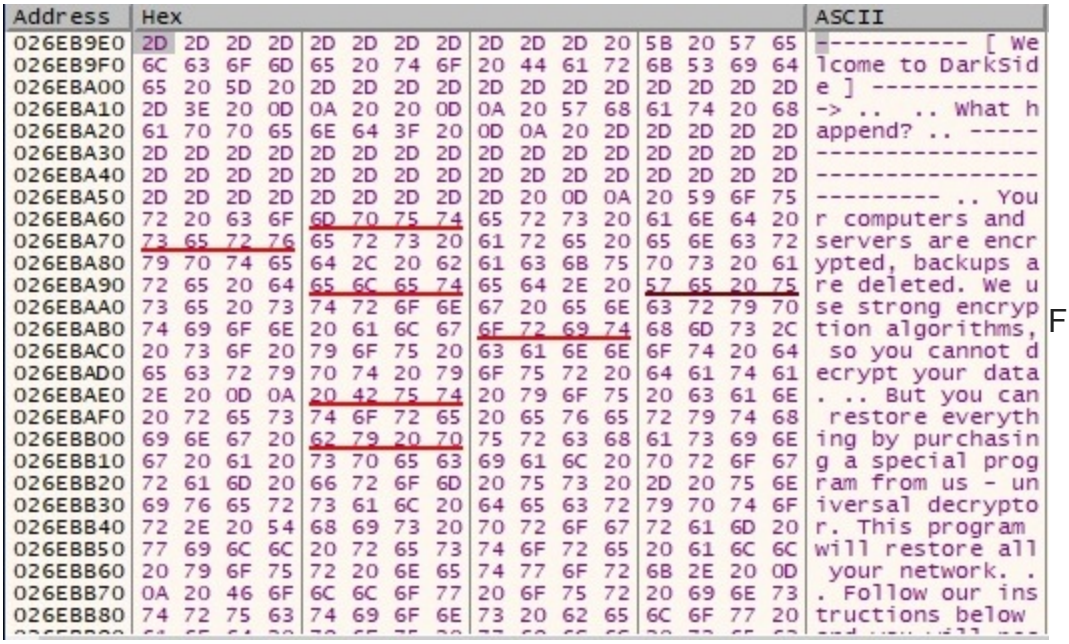
The process reveals a message that will be utilized to set a custom wallpaper that contains important instructions for the victim:



Address	Hex	ASCII
026EB900	57 00 65 00 6C 00 63 00 6F 00 6D 00 65 00 20 00	W.e.l.c.o.m.e. .
026EB910	74 00 6F 00 20 00 44 00 61 00 72 00 6B 00 53 00	t.o. .D.a.r.k.S.
026EB920	69 00 64 00 65 00 21 00 20 00 0D 00 0A 00 20 00	i.d.e.!. ..... .
026EB930	20 00 0D 00 0A 00 20 00 41 00 6C 00 6C 00 20 00	..... .A.l.l. .
026EB940	59 00 6F 00 75 00 72 00 20 00 46 00 69 00 6C 00	Y.o.u.r. .F.i.l.
026EB950	65 00 73 00 20 00 41 00 72 00 65 00 20 00 45 00	e.s. .A.r.e. .E.
026EB960	6E 00 63 00 72 00 79 00 70 00 74 00 65 00 64 00	n.c.r.y.p.t.e.d.
026EB970	21 00 20 00 0D 00 0A 00 20 00 20 00 0D 00 0A 00	!. ....
026EB980	20 00 46 00 69 00 6E 00 64 00 20 00 25 00 73 00	.F.i.n.d. .%.s.
026EB990	20 00 41 00 6E 00 64 00 20 00 46 00 6F 00 6C 00	.A.n.d. .F.o.l.
026EB9A0	6C 00 6F 00 77 00 20 00 49 00 6E 00 73 00 74 00	l.o.w. .I.n.s.t.
026EB9B0	72 00 75 00 63 00 74 00 69 00 6F 00 6E 00 73 00	r.u.c.t.i.o.n.s.
026EB9C0	21 00 00 00 00 00 00 00 AB AB AB AB AB AB AB AB	!.......««««««««

Figure 16

The content of the ransom note is also written in the process memory, as shown in figure 17:



Address	Hex	ASCII
026EB9E0	2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 20 5B 20 57 65	---------- [We
026EB9F0	6C 63 6F 6D 65 20 74 6F 20 44 61 72 6B 53 69 64	lcome to DarkSid
026EBA00	65 20 5D 20 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D	e] ----------
026EBA10	2D 3E 20 0D 0A 20 20 0D 0A 20 57 68 61 74 20 68	-> .. .. What h
026EBA20	61 70 70 65 6E 64 3F 20 0D 0A 20 2D 2D 2D 2D 2D	append? .. -----
026EBA30	2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D	----------------
026EBA40	2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D	----------------
026EBA50	2D 2D 2D 2D 2D 2D 2D 2D 2D 20 0D 0A 20 59 6F 75	--------- .. You
026EBA60	72 20 63 6F 6D 70 75 74 65 72 73 20 61 6E 64 20	r computers and
026EBA70	73 65 72 76 65 72 73 20 61 72 65 20 65 6E 63 72	servers are encr
026EBA80	79 70 74 65 64 2C 20 62 61 63 6B 75 70 73 20 61	ypted, backups a
026EBA90	72 65 20 64 65 6C 65 74 65 64 2E 20 57 65 20 75	re deleted. We u
026EBAA0	73 65 20 73 74 72 6F 6E 67 20 65 6E 63 72 79 70	se strong encryp
026EBAB0	74 69 6F 6E 20 61 6C 67 6F 72 69 74 68 6D 73 2C	tion algorithms,
026EBAC0	20 73 6F 20 79 6F 75 20 63 61 6E 6E 6F 74 20 64	so you cannot d
026EBAD0	65 63 72 79 70 74 20 79 6F 75 72 20 64 61 74 61	ecrypt your data
026EBAE0	2E 20 0D 0A 20 42 75 74 20 79 6F 75 20 63 61 6E	. .. But you can
026EBAF0	20 72 65 73 74 6F 72 65 20 65 76 65 72 79 74 68	restore everyth
026EBB00	69 6E 67 20 62 79 20 70 75 72 63 68 61 73 69 6E	ing by purchasin
026EBB10	67 20 61 20 73 70 65 63 69 61 6C 20 70 72 6F 67	g a special prog
026EBB20	72 61 6D 20 66 72 6F 6D 20 75 73 20 2D 20 75 6E	ram from us - un
026EBB30	69 76 65 72 73 61 6C 20 64 65 63 72 79 70 74 6F	iversal decrypto
026EBB40	72 2E 20 54 68 69 73 20 70 72 6F 67 72 61 6D 20	r. This program
026EBB50	77 69 6C 6C 20 72 65 73 74 6F 72 65 20 61 6C 6C	will restore all
026EBB60	20 79 6F 75 72 20 6E 65 74 77 6F 72 6B 2E 20 0D	your network. .
026EBB70	0A 20 46 6F 6C 6C 6F 77 20 6F 75 72 20 69 6E 73	. Follow our ins
026EBB80	74 72 75 63 74 69 6F 6E 73 20 62 65 6C 6F 77 20	tructions below

Figure 17

The following table describes the actions that the malware takes depending on the configuration decrypted above:

Offset	Enabled	Description
0x00	Yes	FAST encryption mode
0x01	Yes	Unknown (not used)
0x02	No	Attempt to log on as a user on the machine
0x03	Yes	Encrypt DRIVE_REMOVABLE, DRIVE_FIXED and DRIVE_REMOTE type of drives
0x04	Yes	Retrieve the domain controllers and probably an attempt to spread further
0x05	Yes	Check system language and avoid the Russian language
0x06	Yes	Delete volume shadow copies
0x07	Yes	Delete files and folders from Recycle Bin

0x08	No	Self deletion
0x09	Yes	Ignore specific directories
0x0a	Yes	Ignore specific files
0x0b	Yes	Ignore specific file extensions
0x0c	Yes	Wipe “backup” directories
0x0d	Yes	Unknown (not used)
0x0e	Yes	Kill specific processes
0x0f	Yes	Stop and delete specific services
0x10	Yes	Set Desktop wallpaper
0x11	Yes	Drop ransom note
0x12	Yes	Change icon of new encrypted files
0x13	Yes	Create a mutex
0x14	Yes	Unknown (not used)
0x15	Yes	Communication with the C2 servers

The malware uses the NtQueryInstallUILanguage and NtQueryDefaultUILanguage APIs to determine the language of the system and compares the result with 0x419 (Russian language identifier). If there is a match between these two values, then the malware exits:

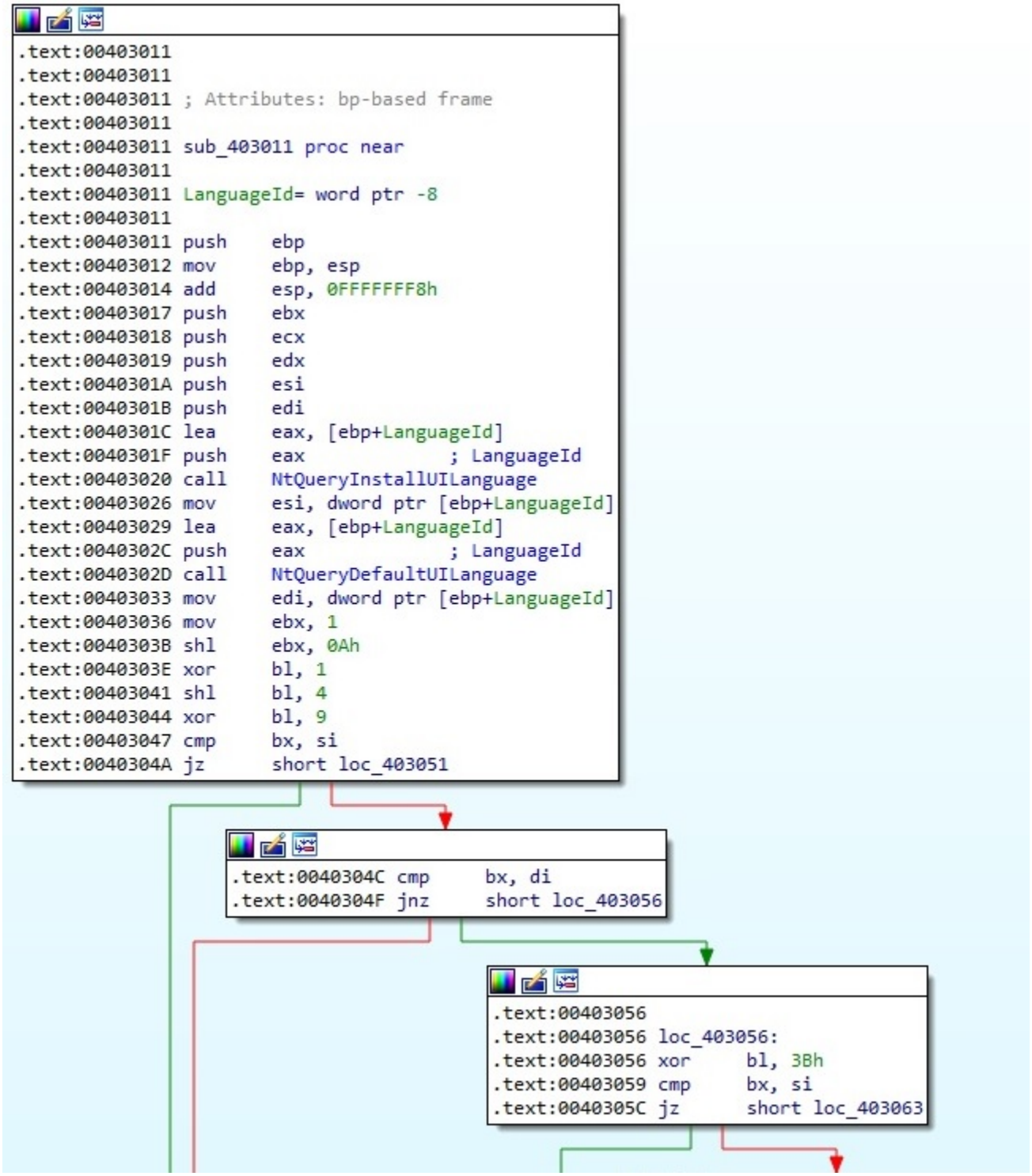


Figure 18

There is a call to the RegCreateKeyExW function, which is supposed to create (or open if it already exists) the “Software\Microsoft\Cryptography” registry key, as follows:

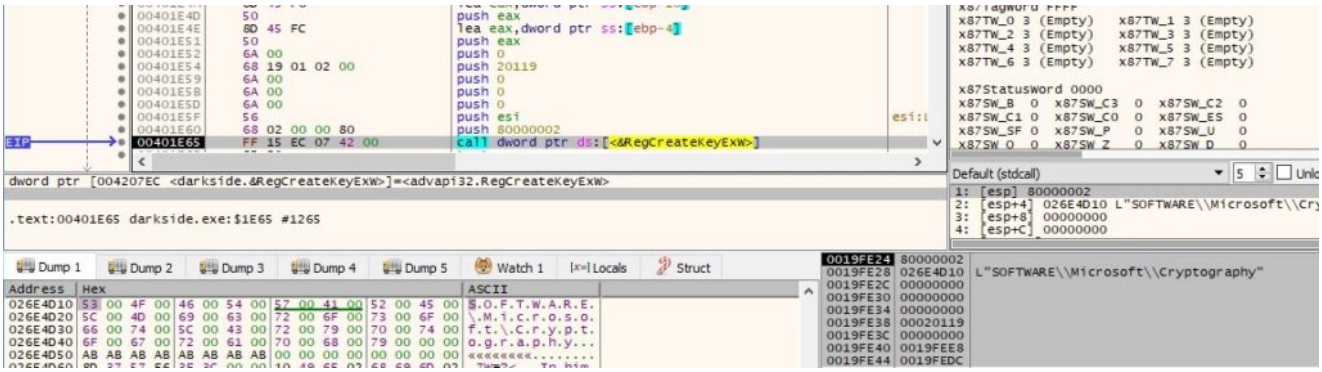


Figure 19

The malware extracts the “MachineGuid” value from the above registry key, as presented in the next figure:

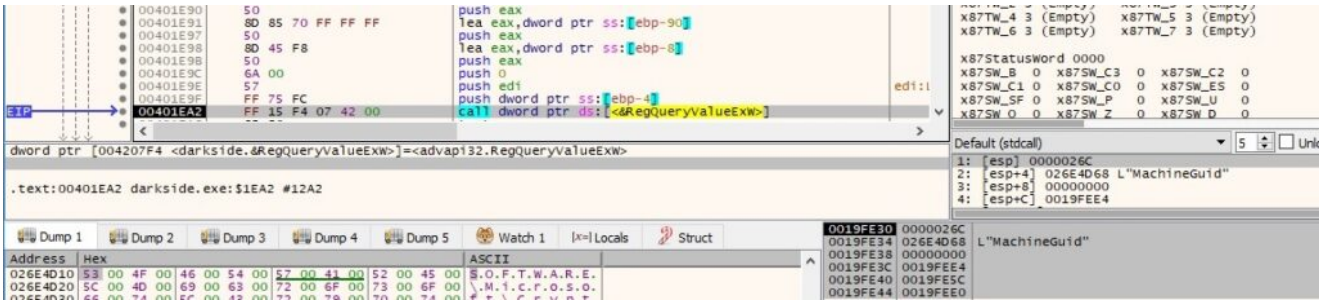


Figure 20

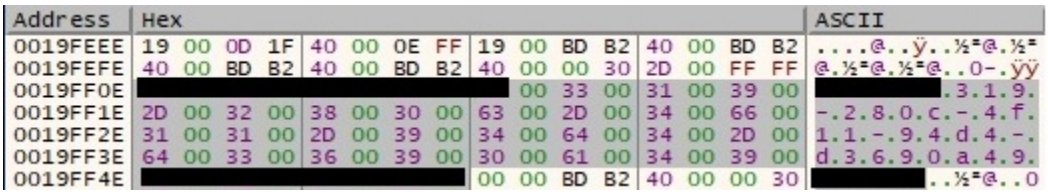


Figure 21

A custom hashing algorithm that generates 8 lowercase hexadecimal characters is implemented by the process (the “MachineGuid” value is the input, and the algorithm applies 8 times):

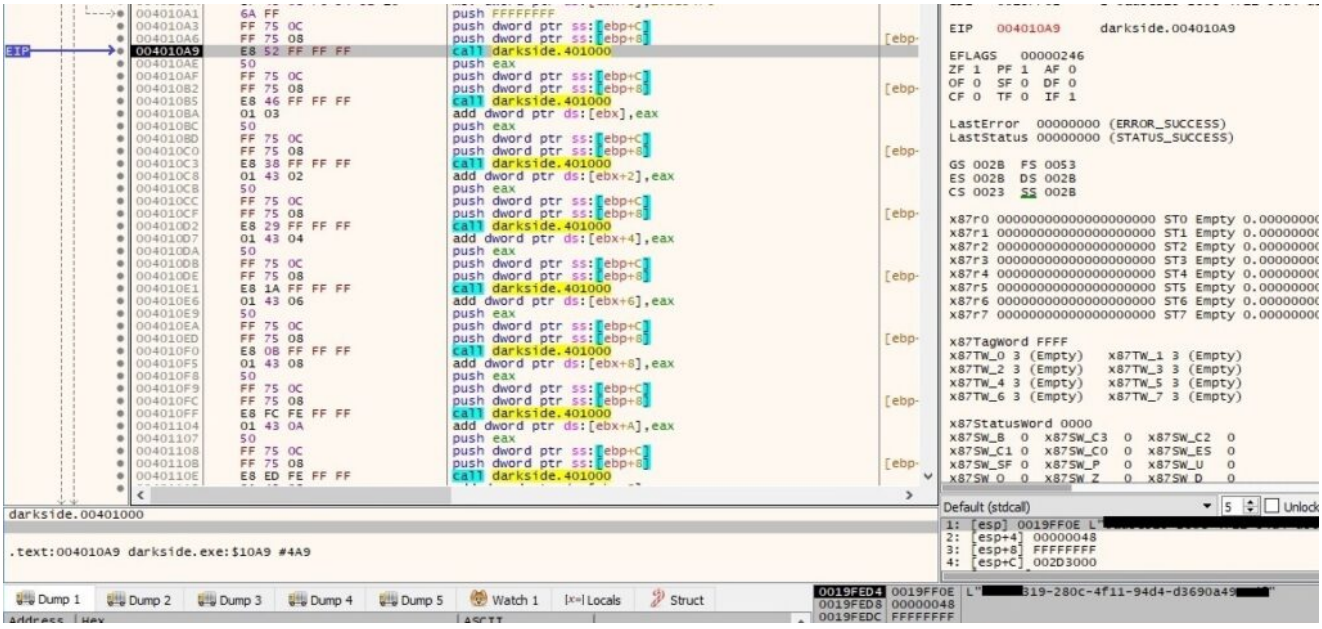


Figure 22

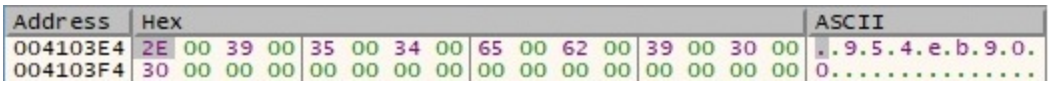


Figure 23

The value computed above (let's call it **RansomPseudoValue**) will be used in the following constructions:

- Service name: <RansomPseudoValue>
- Service display name: <RansomPseudoValue>
- Ransom note: README<RansomPseudoValue>.TXT
- Wallpaper: %PROGRAMDATA%\<RansomPseudoValue>.BMP
- Each encrypted file will have the following name: <Original filename> <RansomPseudoValue>
- Icon file: %PROGRAMDATA%\<RansomPseudoValue>.ico
- Registry key created: HKCR\ <RansomPseudoValue>\DefaultIcon=%PROGRAMDATA%\ <RansomPseudoValue>.ico

The binary uses the SHTestTokenMembership API to verify if the user belongs to the Administrators groups (0x220 = 544 in decimal):

Figure 24

We'll split the analysis into 3 different parts depending on the user's privileges: low level privileges, administrative privileges, and SYSTEM privileges.

Low Level privileges

The malware attempts a UAC bypass that uses the CMSTPLUA COM interface as described at https://gist.github.com/api0cradle/d4aaef39db0d845627d819b2b6b30512. It utilizes ZwOpenProcessToken to open the access token associated with the process (0x8 = **TOKEN_QUERY** – required to query an access token):

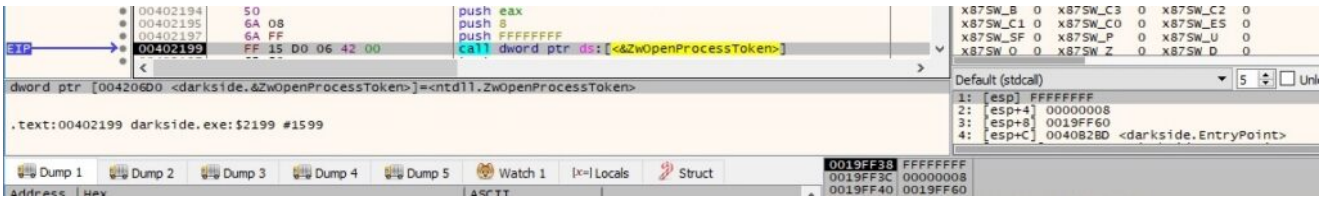


Figure 25

The NtQueryInformationToken function is used to get the group accounts associated with the token (0x2 = **TokenGroups**) and it checks if the administrators group can be found in the TOKEN_GROUPS structure:

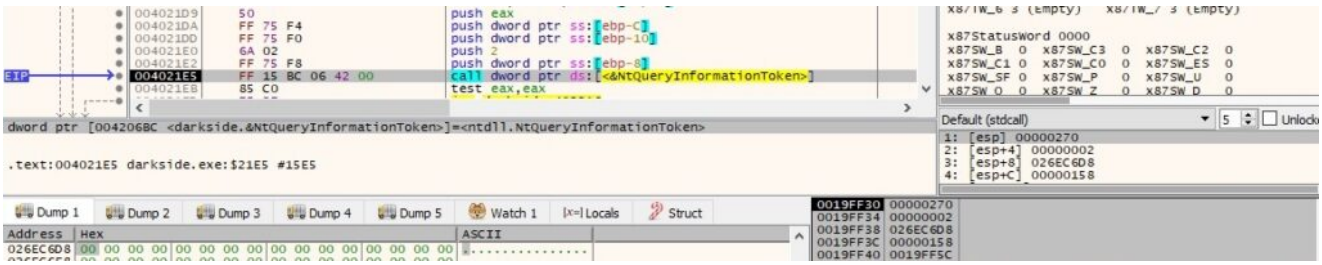


Figure 26

There is a call to the CoInitialize routine in order to initialize the COM library on the current thread, as highlighted in figure 27:

Figure 27

As presented so far, the binary uses a lot of lower level APIs (from ntdll). It allocates a new memory area using the ZwAllocateVirtualMemory API (0x3000 = **MEM_COMMIT** | **MEM_RESERVE** and 0x4 = **PAGE_READWRITE**):

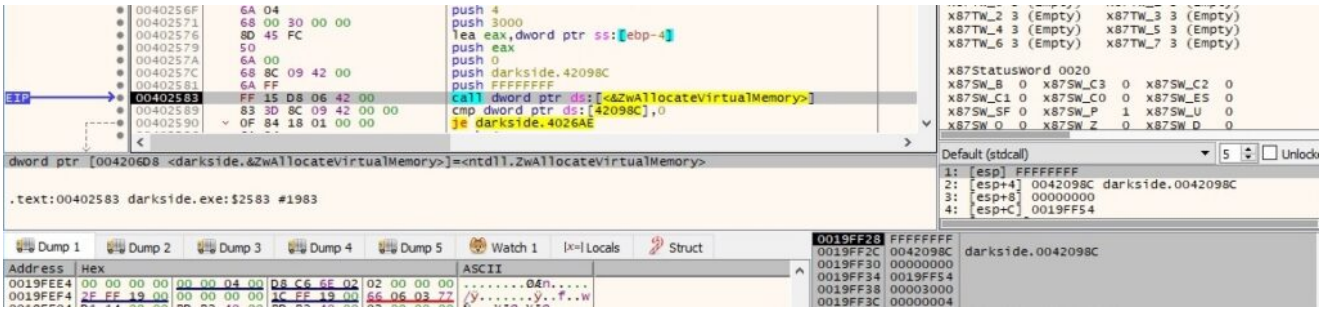


Figure 28

We have encountered a call to an undocumented API function called LdrEnumerateLoadedModules:

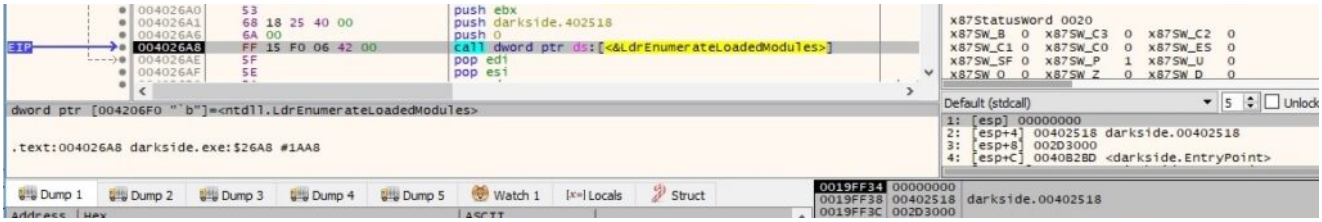


Figure 29

The file executes CoGetObject with the object name as **Elevation:Administrator!new: {3E5FC7F9-9A51-4367-9063-A120244FBEC7}**, as highlighted below:

Figure 30

Basically, it will relaunch the malware with SYSTEM privileges:

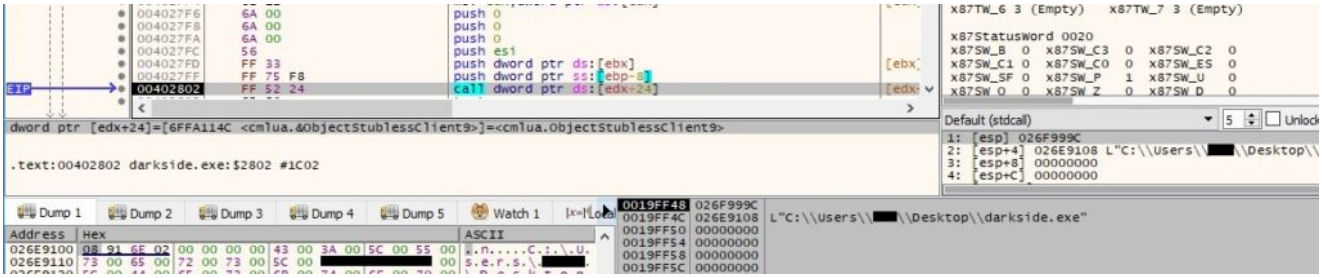


Figure 31

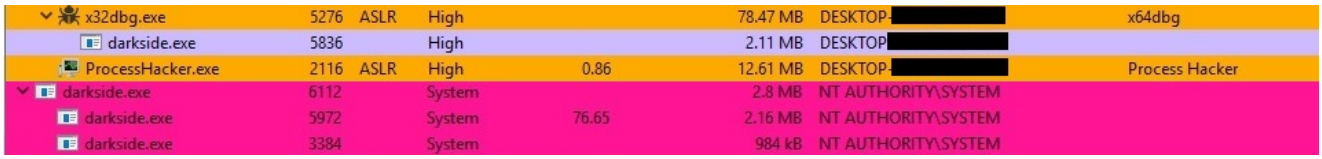


Figure 32

Administrative privileges

As in the first case, the binary uses ZwOpenProcessToken to open the access token associated with the process (0x8 = **TOKEN_QUERY** – required to query an access token):

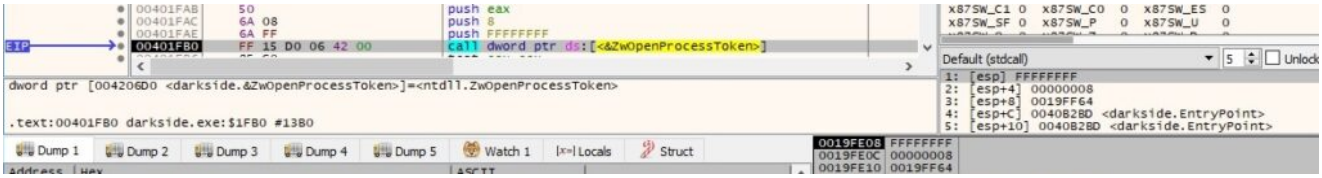


Figure 33

The NtQueryInformationToken API is utilized to retrieve the token's user account (0x1 = **TokenUser**):

Figure 34

The malicious process uses LookupAccountSidW to obtain the name of the account associated with the SID provided as the input, as shown in figure 35:

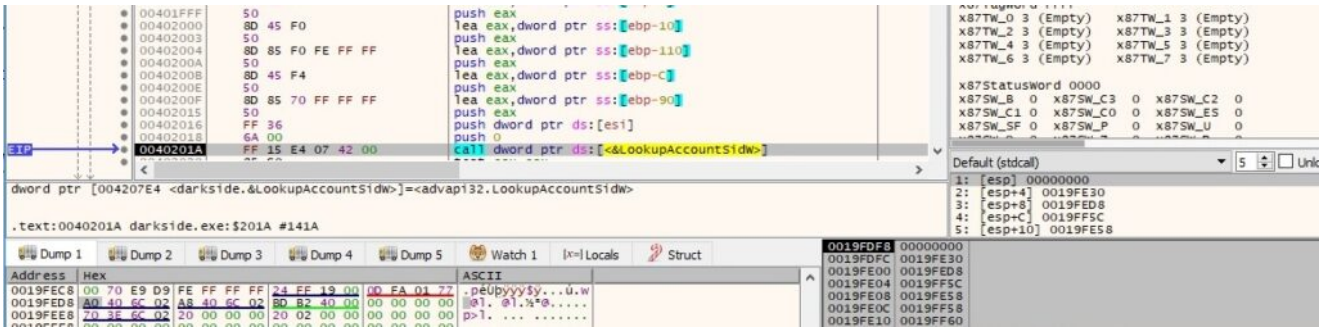


Figure 35

There are 3 different comparison operations that compare the domain name (the name of the computer in our case) with “NT AUTHORITY”, “AUTORITE NT” and “NT-AUTORITAT” (basically, it tries to determine if the user account is SYSTEM or not):

```
.text:0040203C call    _wcsicmp
.text:00402042 add     esp, 8
.text:00402045 test    eax, eax
.text:00402047 jnz     short loc_402050
```

```
.text:00402050
.text:00402050 loc_402050:             ; P
.text:00402050 push    edi
.text:00402051 push    0               ; Flags
.text:00402053 push    HeapHandle      ; HeapHandle
.text:00402059 call    RtlFreeHeap
.text:0040205F push    offset unk_40E03F
.text:00402064 call    Alloc_Decrypt
.text:00402069 mov     edi, eax
.text:0040206B push    edi             ; Str2
.text:0040206C lea     eax, [ebp+Str1]
.text:00402072 push    eax             ; Str1
.text:00402073 call    _wcsicmp
.text:00402079 add     esp, 8
.text:0040207C test    eax, eax
.text:0040207E jnz     short loc_402087
```

```
.text:00402087
.text:00402087 loc_402087:             ; P
.text:00402087 push    edi
.text:00402088 push    0               ; Flags
.text:0040208A push    HeapHandle      ; HeapHandle
.text:00402090 call    RtlFreeHeap
.text:00402096 push    offset unk_40E05B
.text:0040209B call    Alloc_Decrypt
.text:004020A0 mov     edi, eax
.text:004020A2 push    edi             ; Str2
.text:004020A3 lea     eax, [ebp+Str1]
.text:004020A9 push    eax             ; Str1
.text:004020AA call    _wcsicmp
.text:004020B0 add     esp, 8
.text:004020B3 test    eax, eax
.text:004020B5 jnz     short loc_4020BC
```

Figure 36

The OpenSCManagerW routine is utilized to establish a connection to the service control manager:

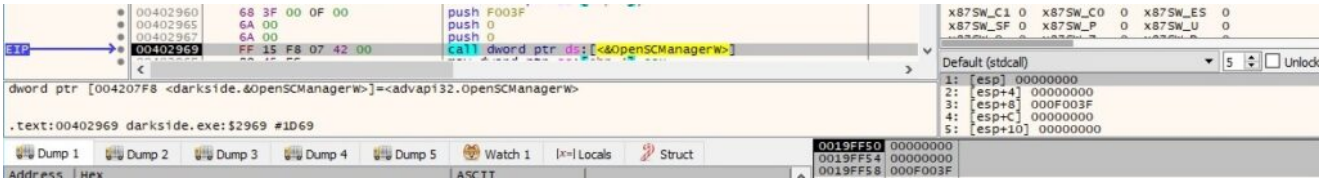


Figure 37

The process tries to open a service called <RansomPseudoValue> (which doesn't exist at this time):

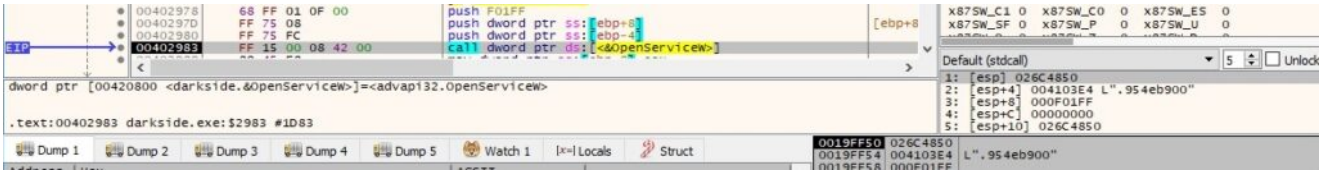


Figure 38

Because the service doesn't exist, it will be created by the malware for persistence purposes, as shown in the following pictures:

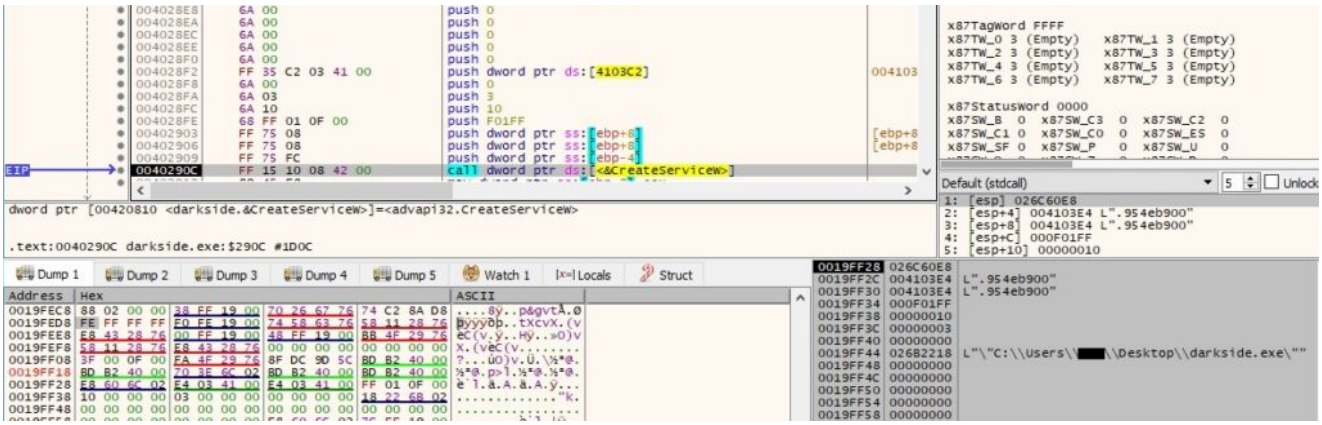


Figure 39

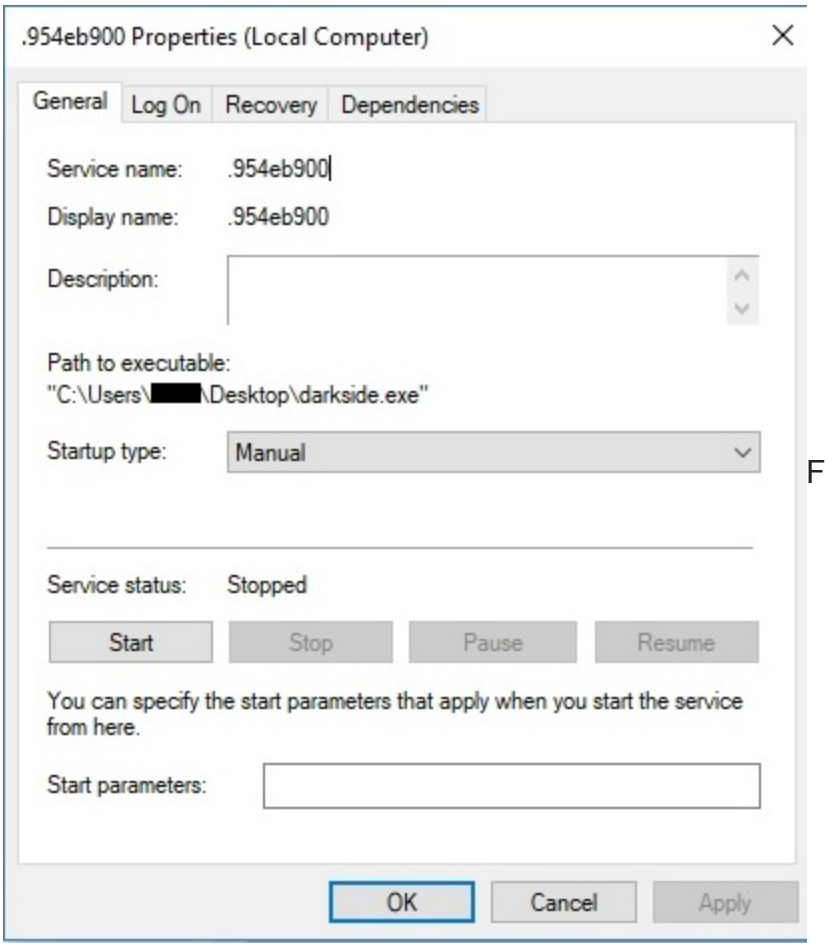


Figure 40

The newly created service is started, and the binary launches itself as a service:

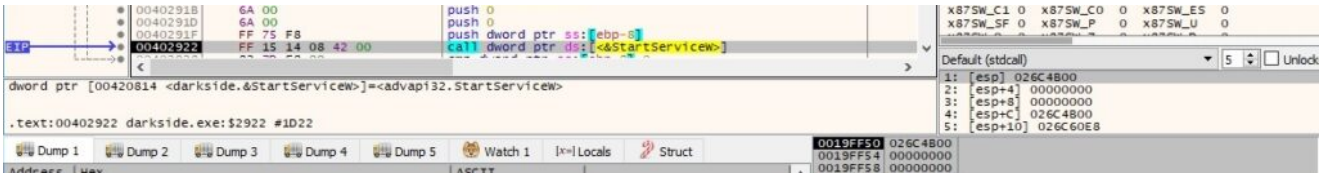


Figure 41

SYSTEM privileges

The malicious binary can run with no arguments, one, two, or three arguments (these cases will be described later on). As we can see below, it uses CommandLineToArgvW to obtain pointers to the command line arguments (argv[0] is the executable name) + the number of arguments:

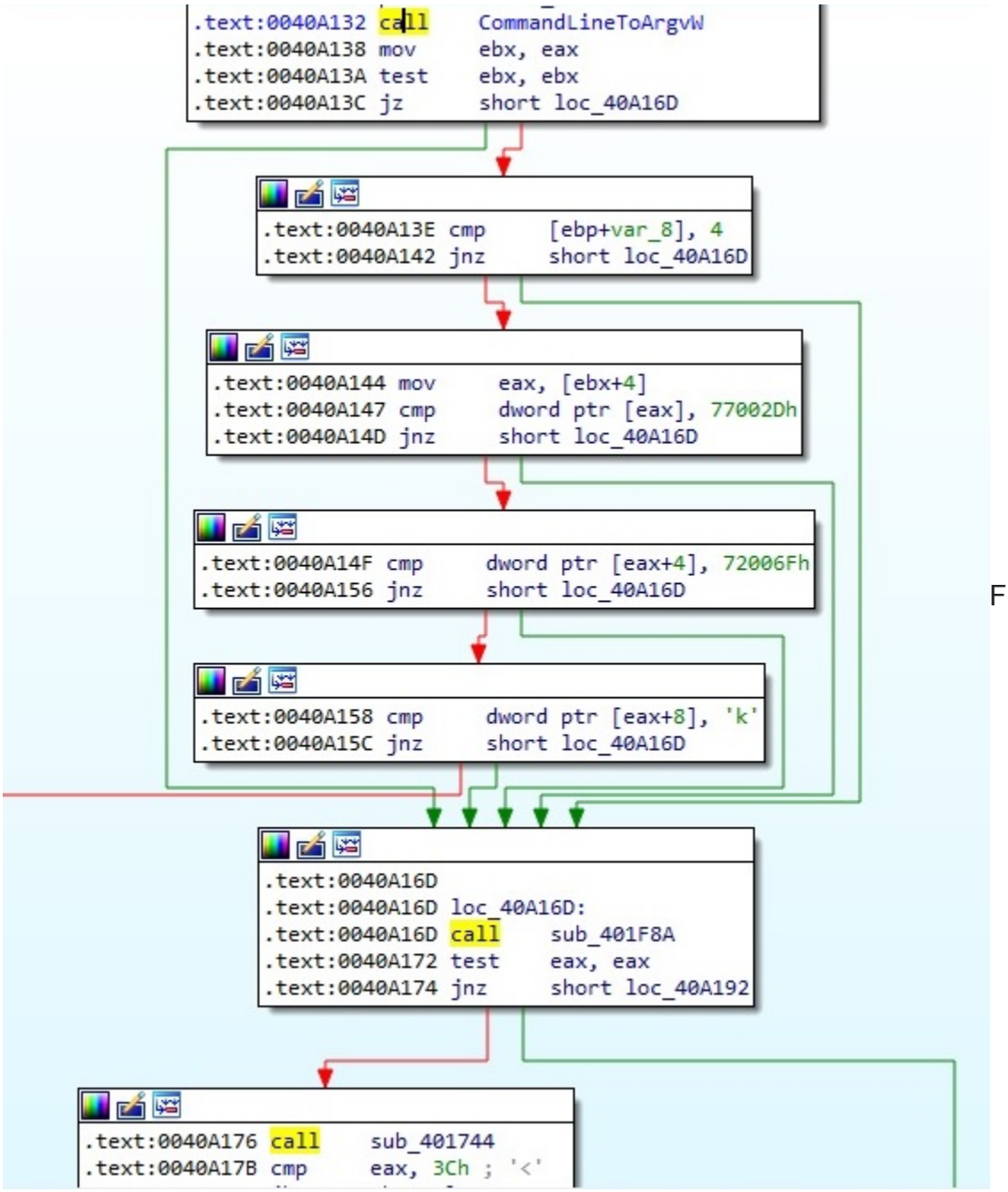


Figure 42

The WTSQueryUserToken API is utilized to obtain the primary access token of the logged-on user specified by session 1:

Figure 43

OpenWindowStationW is used to open the “Winsta0” windows station (the interactive window station), 0x40000 – **WRITE_DAC** – modify the DACL in the security descriptor for the object:

Figure 44

The DACL (discretionary access control list) of the “Winsta0” windows station is modified by calling the NtSetSecurityObject routine with the 0x4 = **DACL_SECURITY_INFORMATION** parameter:

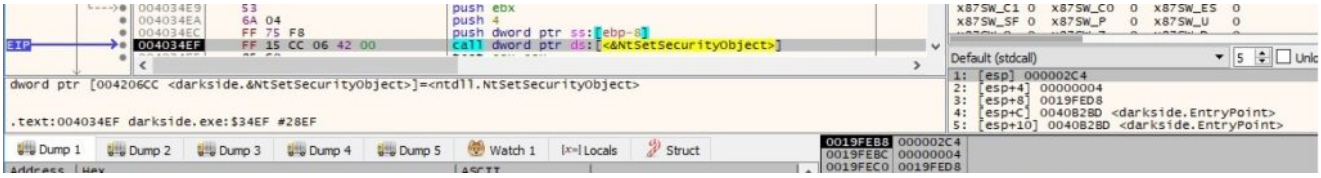


Figure 45

There is a call to OpenDesktopW that is utilized to open the “Default” desktop object with the argument 0x40081 = **WRITE_DAC** | **DESKTOP_WRITEOBJECTS** | **DESKTOP_READOBJECTS**, as follows:

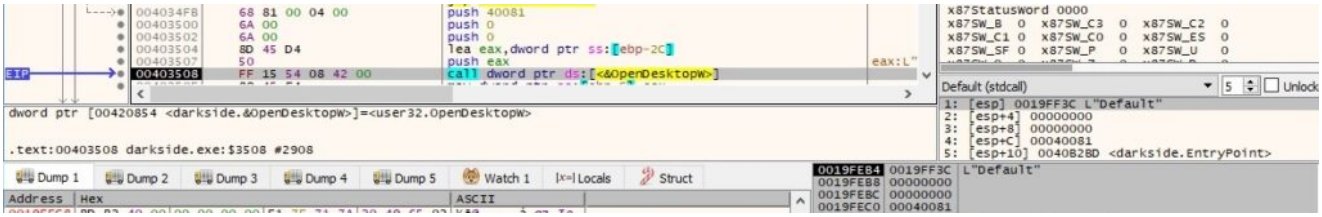


Figure 46

The DACL of the “Default” desktop object is modified by calling the NtSetSecurityObject function with the 0x4 = **DACL_SECURITY_INFORMATION** parameter:

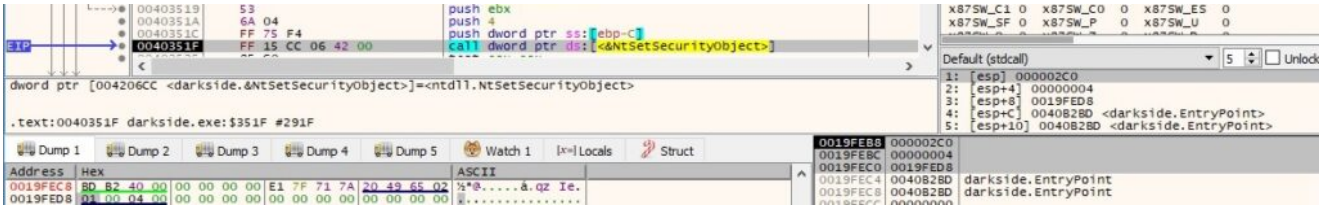


Figure 47

The malware creates a mutex called “Global\4787658f1cc4202b8a15e05dd0323fde” (this value has been computed before this operation and represents a custom “hash” value of the malware), which makes sure that there is only one instance of the ransomware running at a time (if the mutex already exists, then the malware quits):

Figure 48

Figure 49

The ransomware forces the system not to enter sleep mode and not to turn off the display while the process is running, one of the parameters being 0x80000001 = **ES_CONTINUOUS | ES_SYSTEM_REQUIRED**:

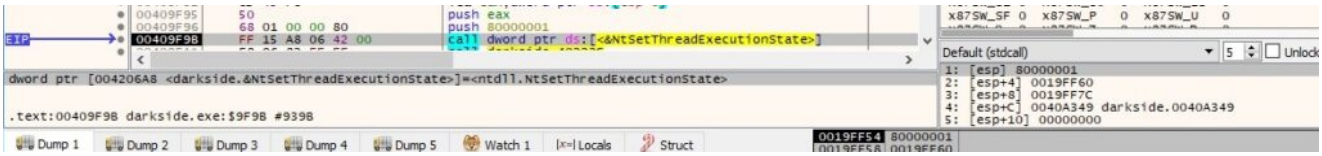


Figure 50

The file changes the privilege to **SE_PRIVILEGE_ENABLED** in order to enable the token's privileges (note the **TOKEN_PRIVILEGES** structure) by a function call to ZwAdjustPrivilegesToken:

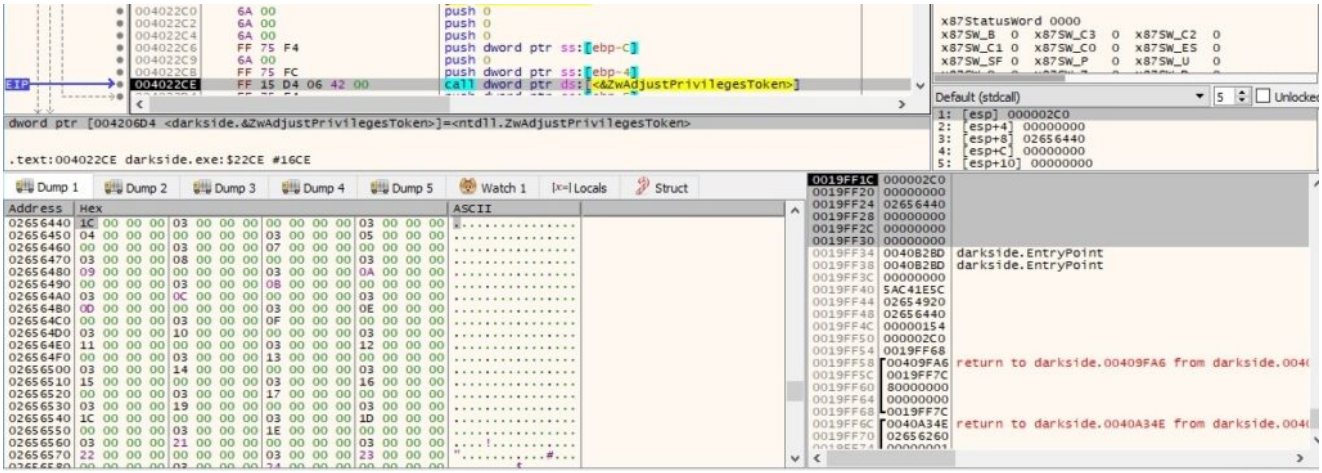


Figure 51

The CreateThread API is used to create a new thread, as described in the next figure:

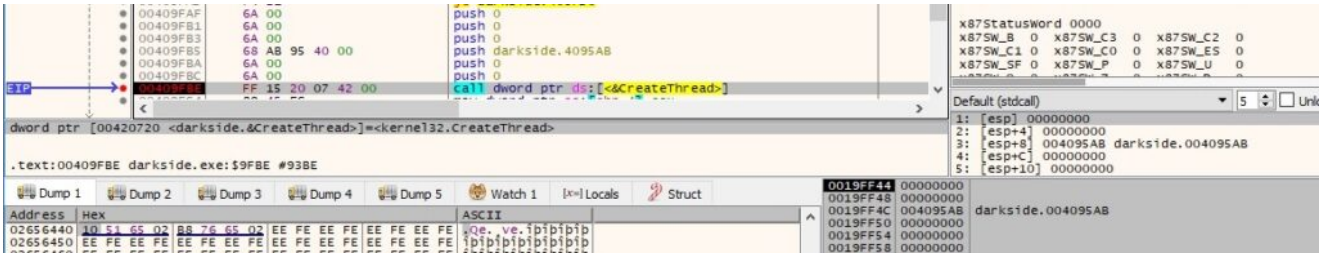


Figure 52

A list of valid drives on the system is extracted using the GetLogicalDriveStringsW routine:

Figure 53

The ransomware is looking for **DRIVE_REMOVABLE** (0x2) and **DRIVE_FIXED** (0x3) drives, as highlighted in figure 54:

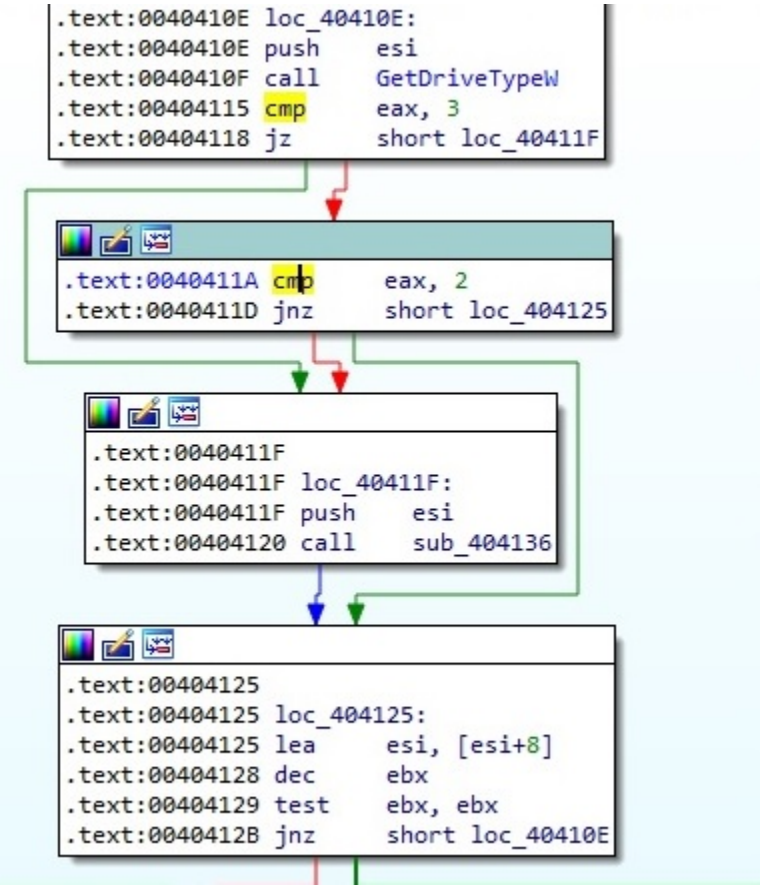


Figure 54

All files and directories from Recycle Bin are deleted by the process. It starts to enumerate via a FindFirstFileExW API call:

Figure 55

As presented below, the files are deleted using the DeleteFileW function, and the directories are removed using the RemoveDirectoryW routine:

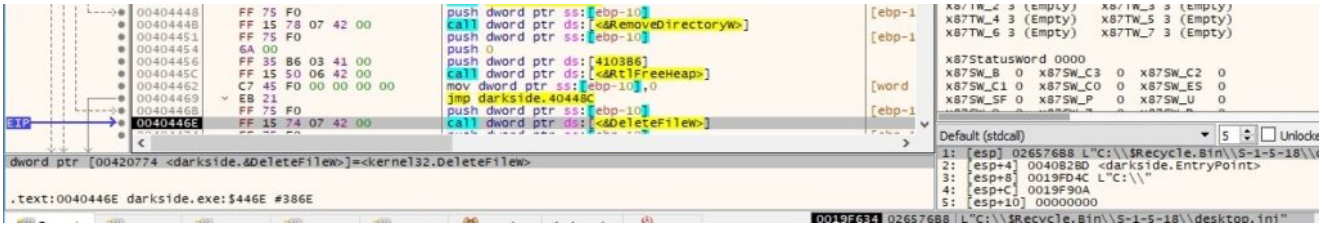


Figure 56

The binary uses COM objects and WMI commands to delete volume shadow copies. It calls the CoCreateInstance function to create a single object of the class IWbemLocator with the CLSID {dc12a687-737f-11cf-884d-00aa004b2e24} (Ref. https://forum.powerbasic.com/forum/user-to-user-discussions/source-code/25222-wmi-wrapper-functions):

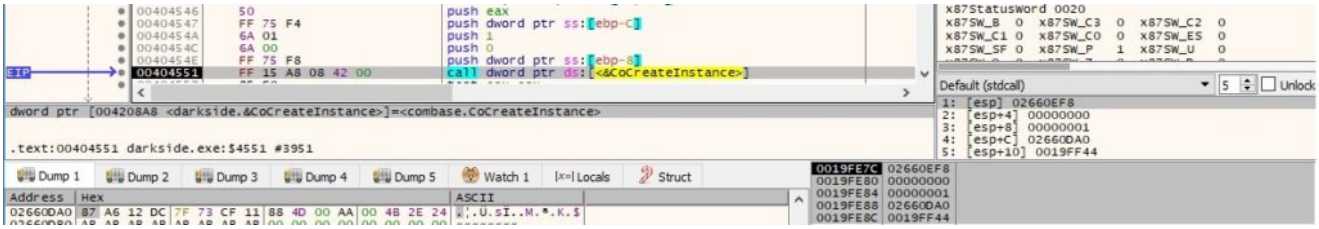


Figure 57

There is also a new IWbemContext interface with the CLSID {44aca674-e8fc-11d0-a07c-00c04fb68820} (Ref. https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wmi/3485541f-6950-4e6d-98cb-1ed4bb143441) created via a CoCreateInstance function call:

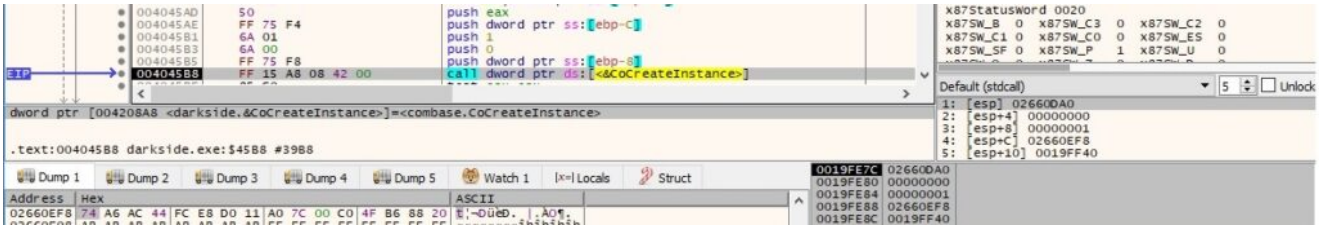


Figure 58

Using the IWbemLocator object, the process calls the ConnectServer API to connect to the local “ROOT\CIMV2” namespace and retrieves a pointer to a IWbemServices object, as follows:

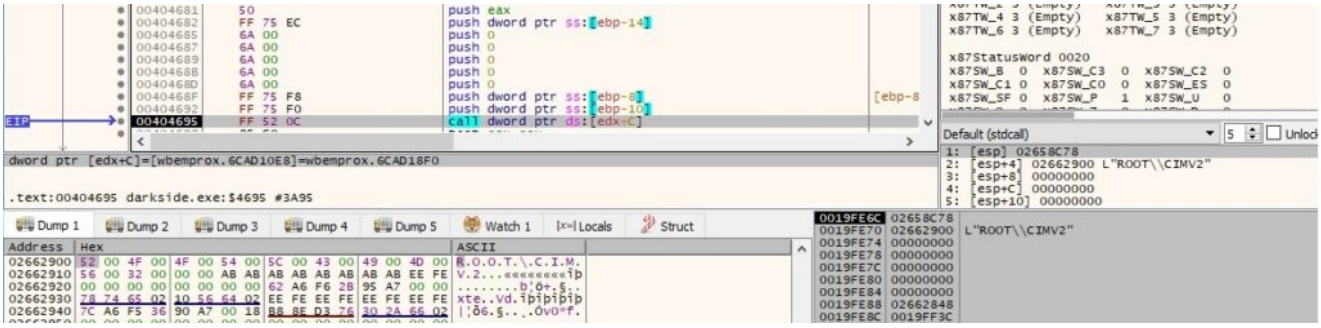


Figure 59

There is a call to CoSetProxyBlanket performed by the ransomware, as described in the next figure (0xA = **RPC_C_AUTHN_WINNT** – NTLMSSP, 0x3 = **RPC_C_AUTHN_LEVEL_CALL** and 0x3 = **RPC_C_IMP_LEVEL_IMPERSONATE**):

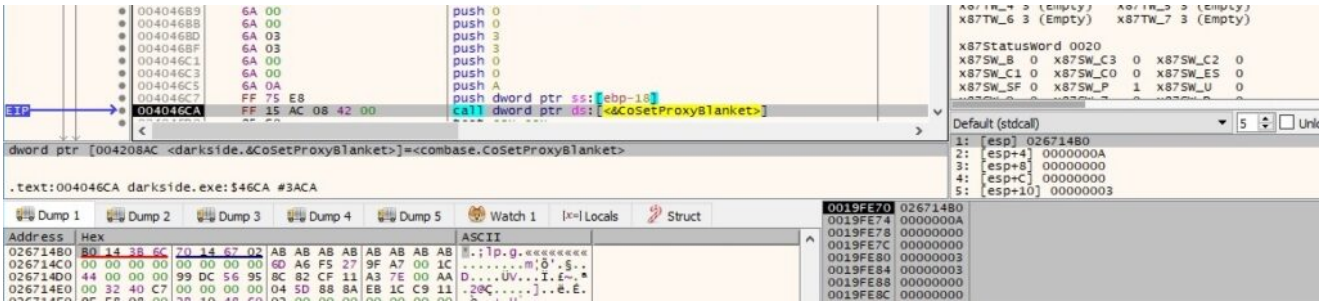


Figure 60

The process executes the following SQL query “SELECT * FROM Win32_ShadowCopy” to obtain an enumerator of all shadow copies, and then it deletes each of the shadow copy objects via the DeleteInstance method:

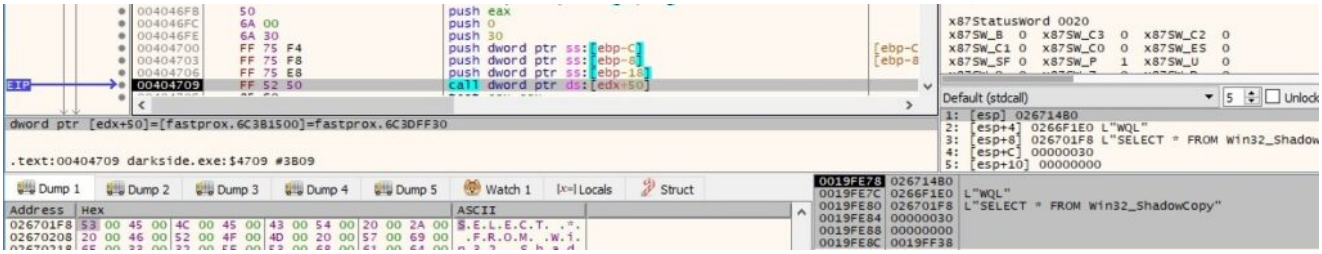


Figure 61

A list of all services and their status is retrieved by calling the EnumServicesStatusExW function (0x30 = **SERVICE_WIN32**, 0x3 = **SERVICE_STATE_ALL**):

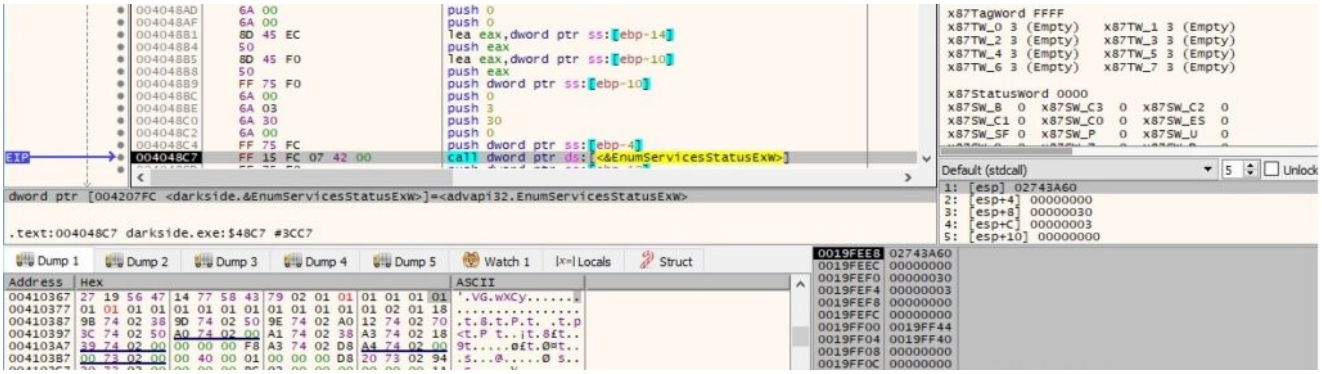


Figure 62

Each service name is compared to the list that was decrypted at the beginning of the analysis:

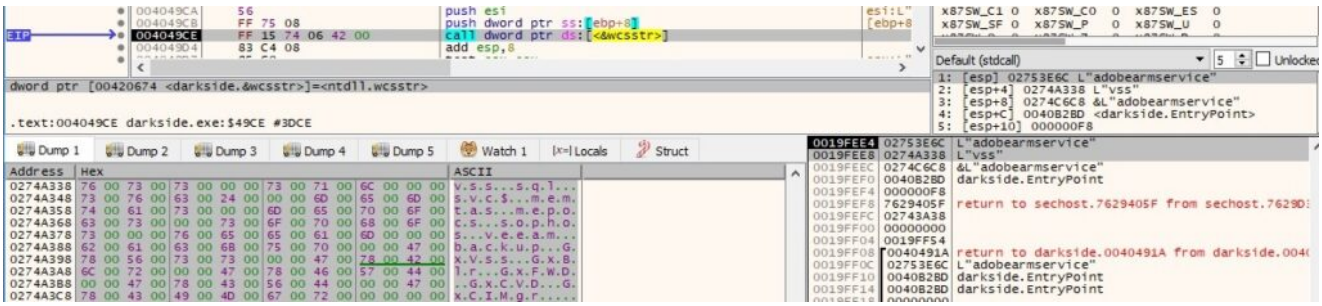


Figure 63

The malware opens the targeted services by calling the OpenServiceW routine (0x10020 = **DELETE** | **SERVICE_STOP**):

Figure 64

Every targeted service is stopped and deleted using ControlService and DeleteService, as displayed in figure 65:

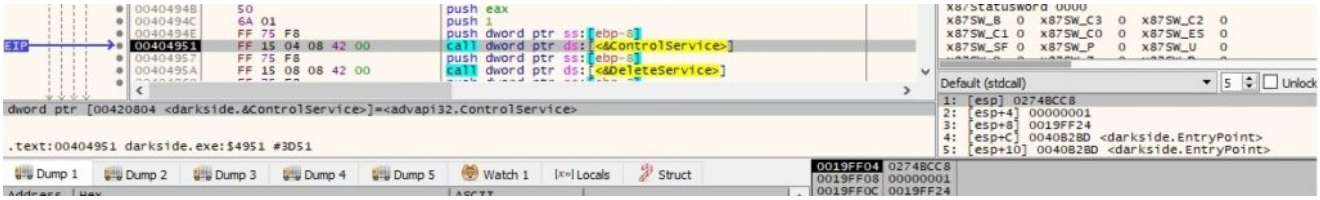


Figure 65

The NtQuerySystemInformation API returns an array of **SYSTEM_PROCESS_INFORMATION** structures (one for each process running on the system, 0x5 = **SystemProcessInformation**):

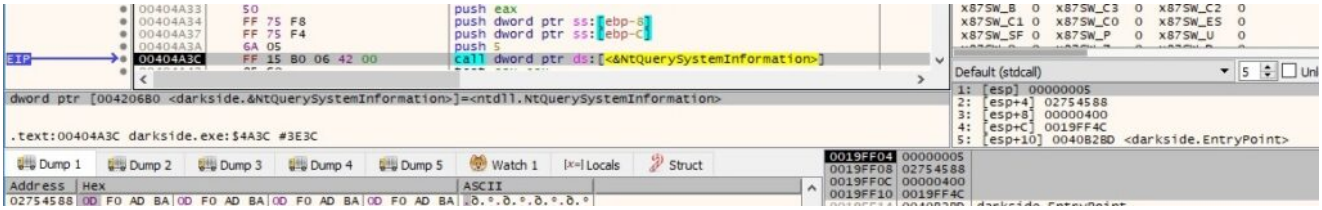


Figure 66

Each process name is compared to the list that was decrypted in the beginning, as displayed below:

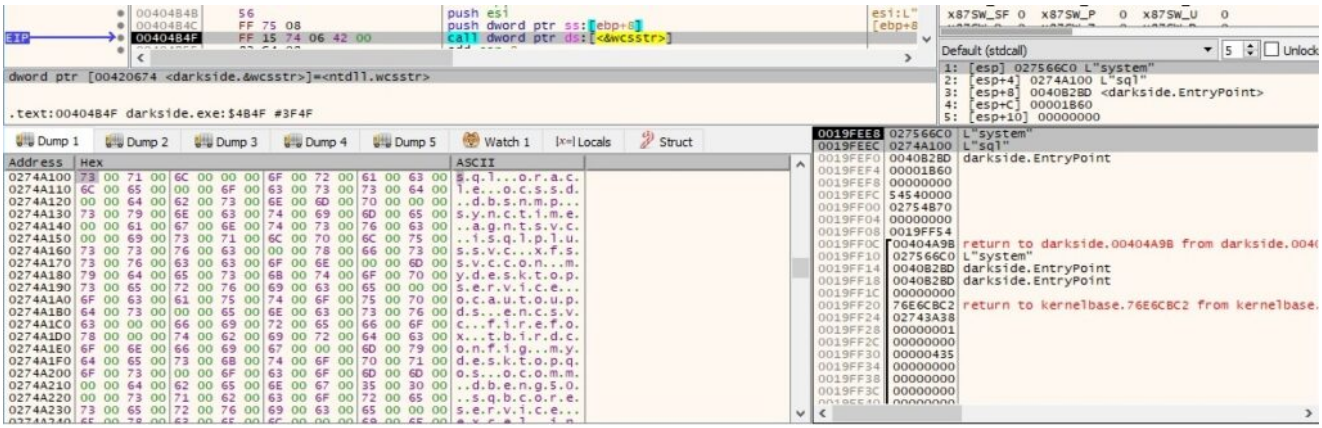


Figure 67

For every targeted process, the binary opens the process and terminates it and all of its threads:

```
.text:00404AD8 lea     eax, [ebp+ClientId]
.text:00404ADB push    eax             ; ClientId
.text:00404ADC lea     eax, [ebp+ObjectAttributes]
.text:00404ADF push    eax             ; ObjectAttributes
.text:00404AE0 push    1               ; DesiredAccess
.text:00404AE2 lea     eax, [ebp+ProcessHandle]
.text:00404AE5 push    eax             ; ProcessHandle
.text:00404AE6 call    NtOpenProcess
.text:00404AEC test    eax, eax
.text:00404AEE jnz     short loc_404B04
```

```
.text:00404AF0 push    0               ; ExitStatus
.text:00404AF2 push    [ebp+ProcessHandle] ; ProcessHandle
.text:00404AF5 call    NtTerminateProcess
.text:00404AFB push    [ebp+ProcessHandle] ; Handle
.text:00404AFE call    NtClose
```

Figure 68

The binary creates an ico file called <RansomPseudoValue>.ico, as displayed below:

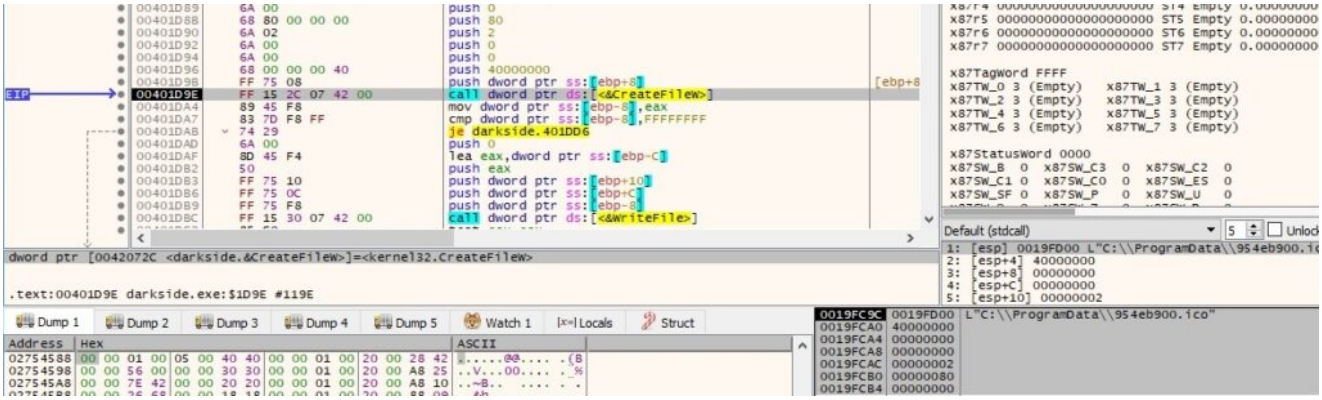


Figure 69

A new registry key called <RansomPseudoValue> is created using the RegCreateKeyExW function, as shown in figure 70:

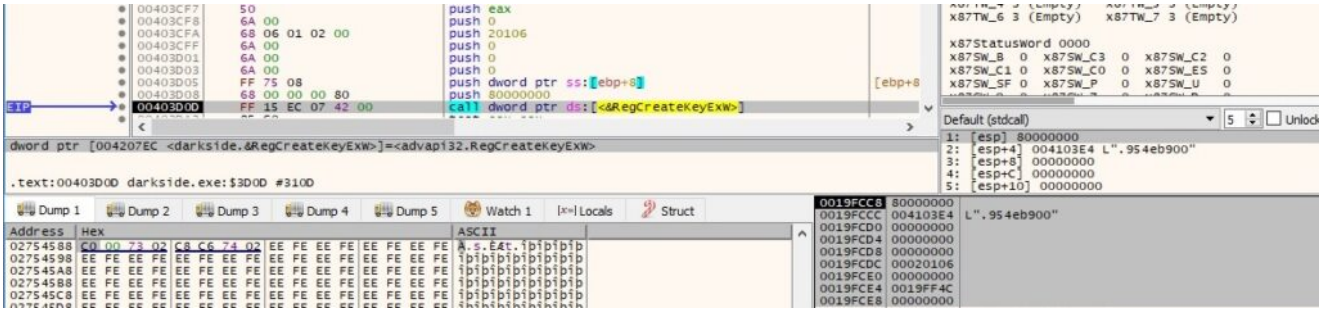


Figure 70

The DefaultIcon subkey is created, and it specifies the path for the newly created ico file:

Figure 71

The malware calls the SHChangeNotify routine to notify the shell to update its icon cache (0x08000000 = **SHCNE_ASSOCCHANGED**, 0x1000 = **SHCNF_FLUSH**):

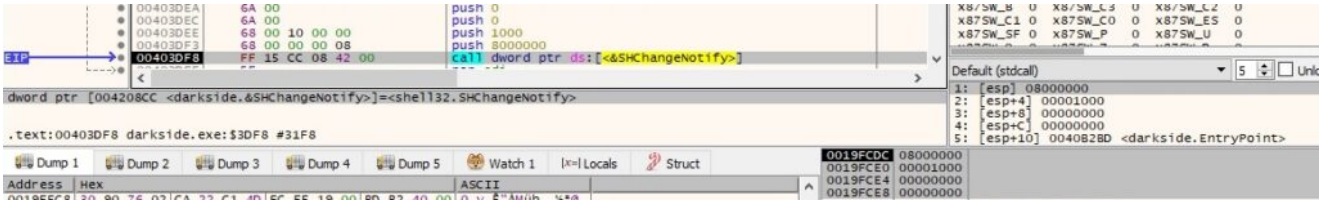


Figure 72

A new file called %PROGRAMDATA%\<RansomPseudoValue>.BMP is created using the CreateFileW function:

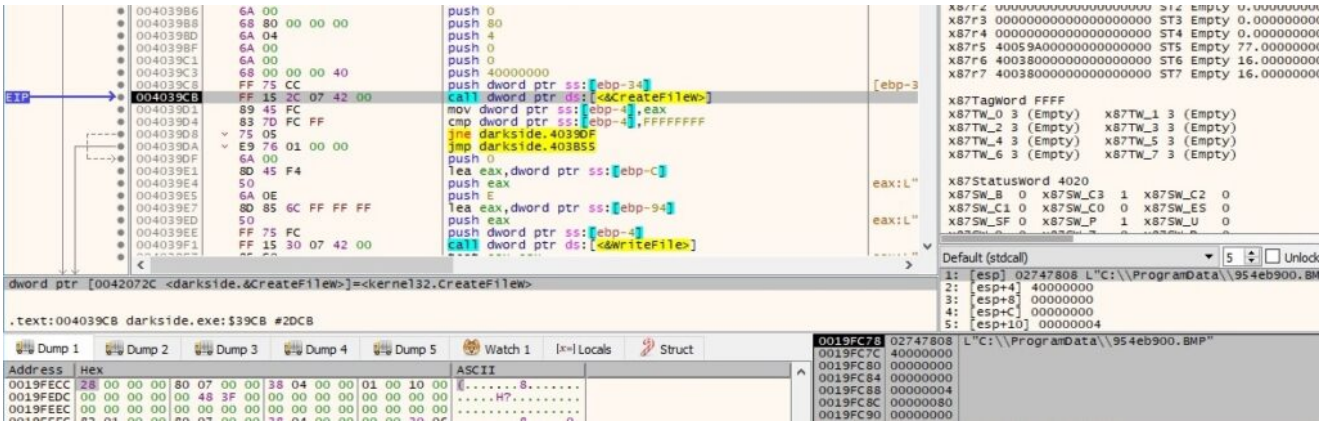


Figure 73

Moving forward, there is a registry key opened by calling the RegCreateKeyExW API, as shown in the next picture:

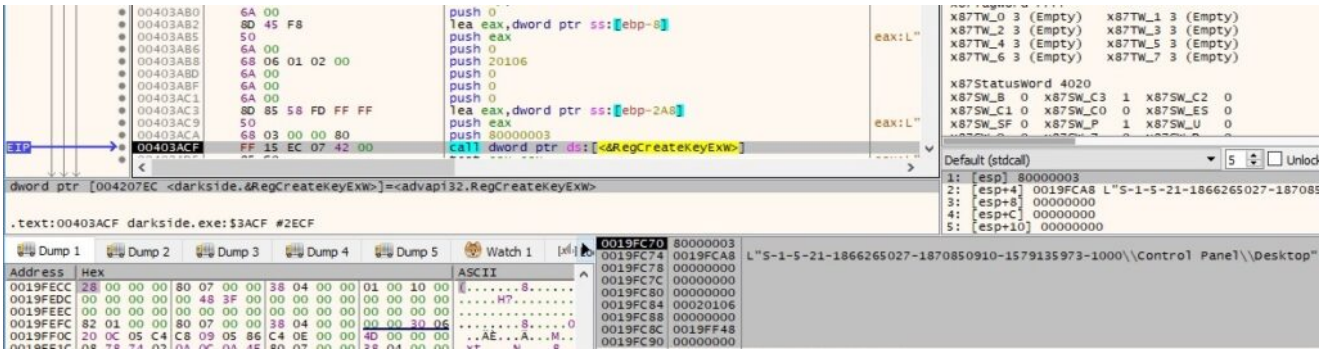


Figure 74

The “WallPaper” value inside the registry key is changed to the location of the newly created BMP file:

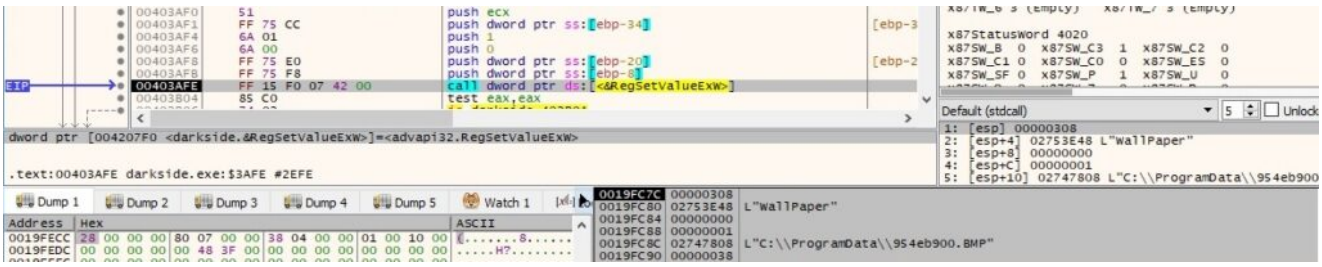


Figure 75

After all of these activities, the Desktop has been changed to the following image:

Figure 76

Thread activity – sub_4095AB

The thread starts by decrypting the following information:

Address	Hex	ASCII
02753E00	7B 0D 0A 22 62 6F 74 22 3A 7B 0D 0A 22 76 65 72	{.."bot":{.."ver
02753E10	22 3A 22 25 73 22 2C 0D 0A 22 75 69 64 22 3A 22	":"%s",.."uid":"
02753E20	25 73 22 0D 0A 7D 2C 0D 0A 25 73 0D 0A 7D 00 AB	%s"..},..%s..}.«

Figure 77

The version of the Darkside ransomware is also decrypted and represents the latest version analyzed in the wild (2.1.2.3):

Address	Hex	ASCII
027435C8	32 2E 31 2E 32 2E 33 00 AB AB AB AB AB AB AB AB	2.1.2.3.««««««««

Figure 78

Another JSON structure is decrypted by the binary and will be used to collect data about the local machine:

Address	Hex	ASCII
027471A8	22 00 6F 00 73 00 22 00 3A 00 7B 00 0D 00 0A 00	".o.s.":.{.....
027471B8	22 00 6C 00 61 00 6E 00 67 00 22 00 3A 00 22 00	".l.a.n.g.":.".
027471C8	25 00 73 00 22 00 2C 00 0D 00 0A 00 22 00 75 00	%.s.",.....".u.
027471D8	73 00 65 00 72 00 6E 00 61 00 6D 00 65 00 22 00	s.e.r.n.a.m.e.".
027471E8	3A 00 22 00 25 00 73 00 22 00 2C 00 0D 00 0A 00	:.".%.s.",.....
027471F8	22 00 68 00 6F 00 73 00 74 00 6E 00 61 00 6D 00	".h.o.s.t.n.a.m.
02747208	65 00 22 00 3A 00 22 00 25 00 73 00 22 00 2C 00	e.".:.".%.s.",.
02747218	0D 00 0A 00 22 00 64 00 6F 00 6D 00 61 00 69 00	....".d.o.m.a.i.
02747228	6E 00 22 00 3A 00 22 00 25 00 73 00 22 00 2C 00	n.".:.".%.s.",.
02747238	0D 00 0A 00 22 00 6F 00 73 00 5F 00 74 00 79 00	....".o.s._.t.y.
02747248	70 00 65 00 22 00 3A 00 22 00 77 00 69 00 6E 00	p.e.":.".w.i.n.
02747258	64 00 6F 00 77 00 73 00 22 00 2C 00 0D 00 0A 00	d.o.w.s.",.....
02747268	22 00 6F 00 73 00 5F 00 76 00 65 00 72 00 73 00	".o.s._.v.e.r.s.
02747278	69 00 6F 00 6E 00 22 00 3A 00 22 00 25 00 73 00	i.o.n.":.".%.s.
02747288	22 00 2C 00 0D 00 0A 00 22 00 6F 00 73 00 5F 00	".,.....".o.s._.
02747298	61 00 72 00 63 00 68 00 22 00 3A 00 22 00 25 00	a.r.c.h.":.".%.
027472A8	73 00 22 00 2C 00 0D 00 0A 00 22 00 64 00 69 00	s.",.....".d.i.
027472B8	73 00 6B 00 73 00 22 00 3A 00 22 00 25 00 73 00	s.k.s.":.".%.s.
027472C8	22 00 2C 00 0D 00 0A 00 22 00 69 00 64 00 22 00	".,.....".i.d.".
027472D8	3A 00 22 00 25 00 73 00 22 00 0D 00 0A 00 7D 00	:.".%.s.".....}.

Figure 79

One more time, the process checks the type of the drives and is looking for **DRIVE_REMOVABLE** (0x2), **DRIVE_FIXED** (0x3) and **DRIVE_REMOTE** (0x4):

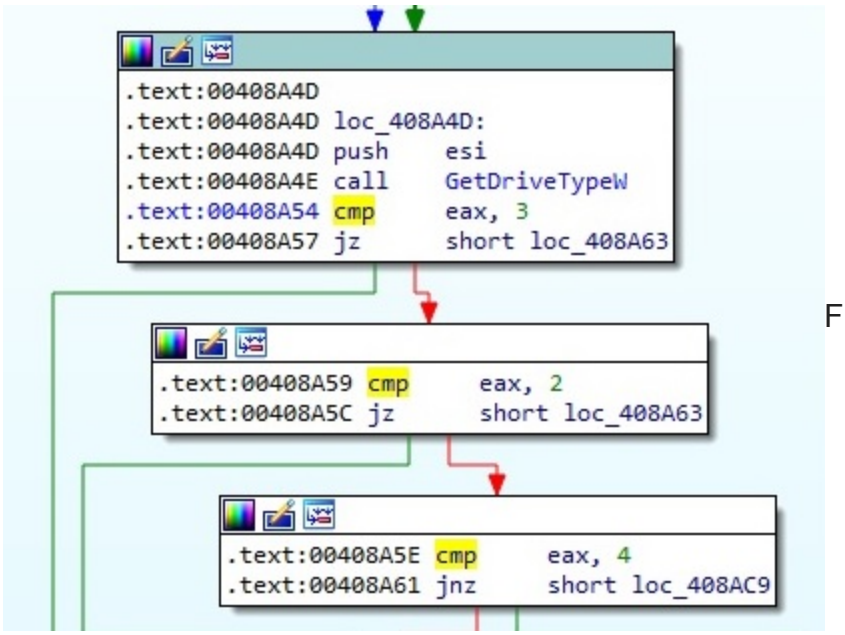


Figure 80

The GetDiskFreeSpaceExW function is used to retrieve information about the targeted drives, such as the total amount of space and the total amount of free space:

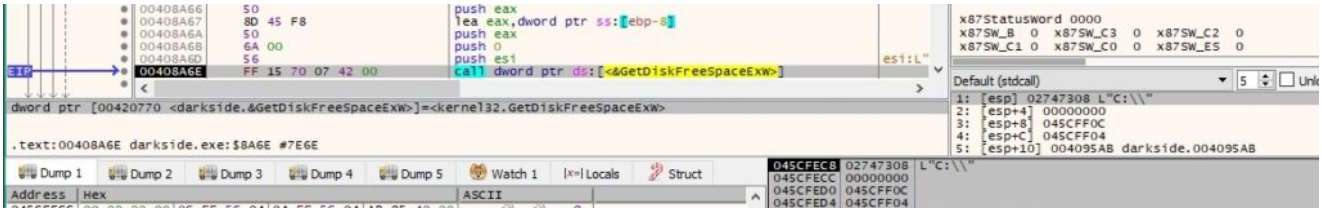


Figure 81

NtDuplicateToken is utilized to duplicate an existing token and to obtain a handle to a new access token (0xC = **TOKEN_DUPLICATE** | **TOKEN_IMPERSONATE** | **TOKEN_QUERY** and 0x2 = **TokenImpersonation**):

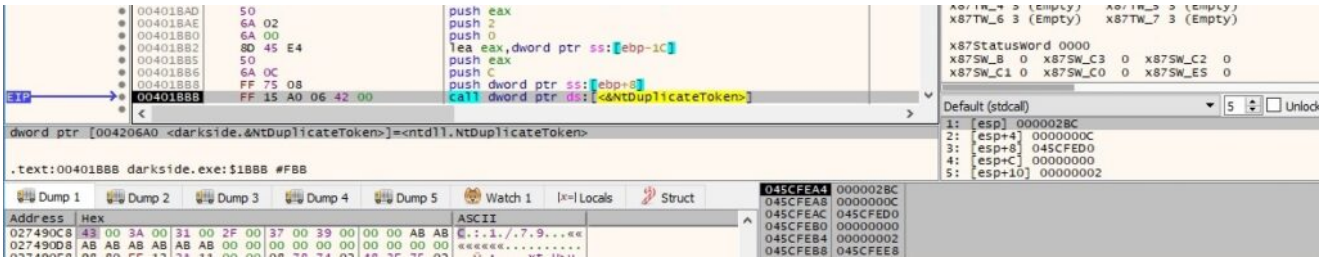


Figure 82

The thread's impersonation token is changed via a call to the ZwSetInformationThread routine, as shown in figure 83 (0x5 = **ThreadImpersonationToken**):

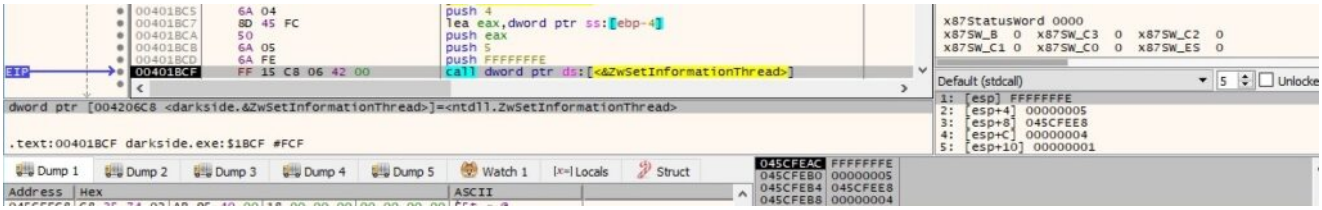


Figure 83

The ransomware retrieves the username associated with the current thread, as well as the NetBIOS name of the local machine:

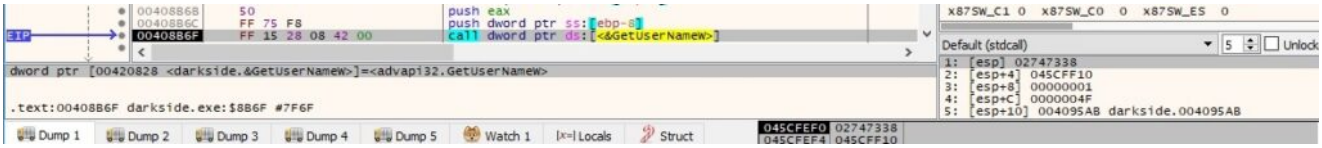


Figure 84

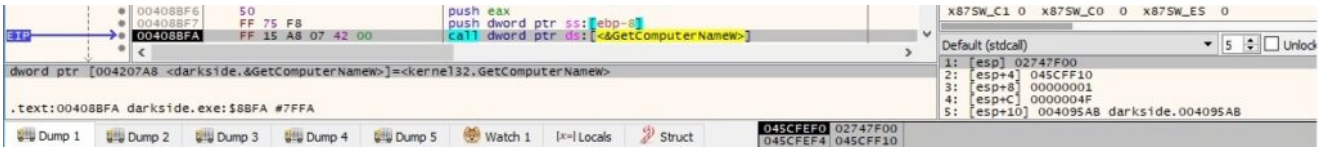


Figure 85

The current language of the machine is retrieved from the “LocaleName” value, as presented below:

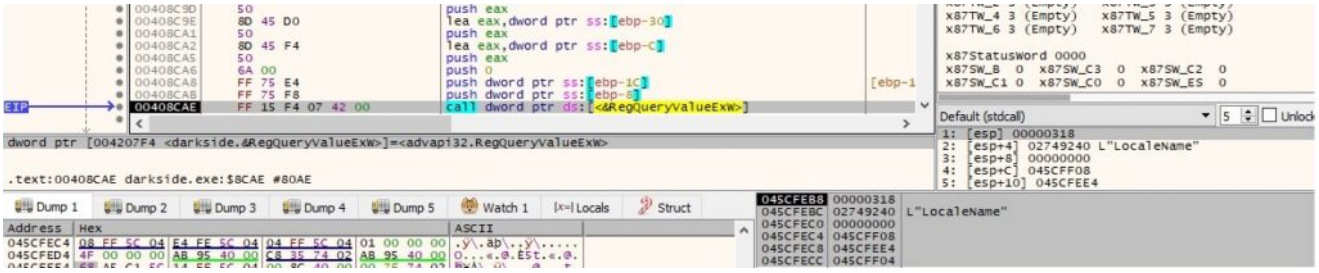


Figure 86

NetGetJoinInformation is used to get the join status information for the local computer:

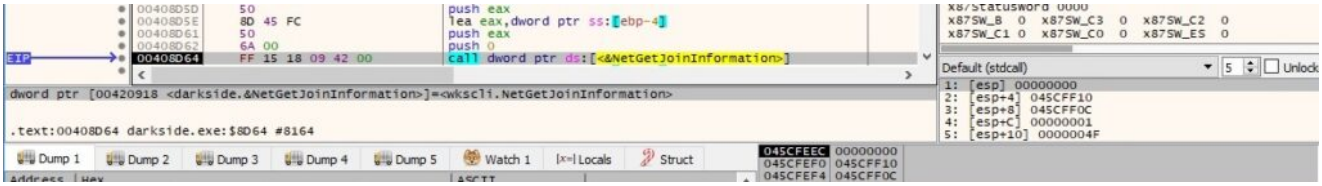


Figure 87

The product name of Windows can be extracted by querying the “ProductName” value and the Windows product ID can be extracted by querying the “ProductId” value, as shown in the following pictures:

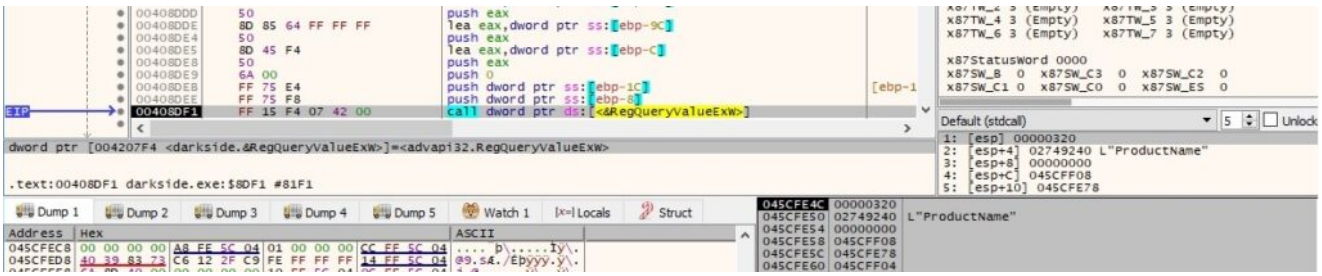


Figure 88

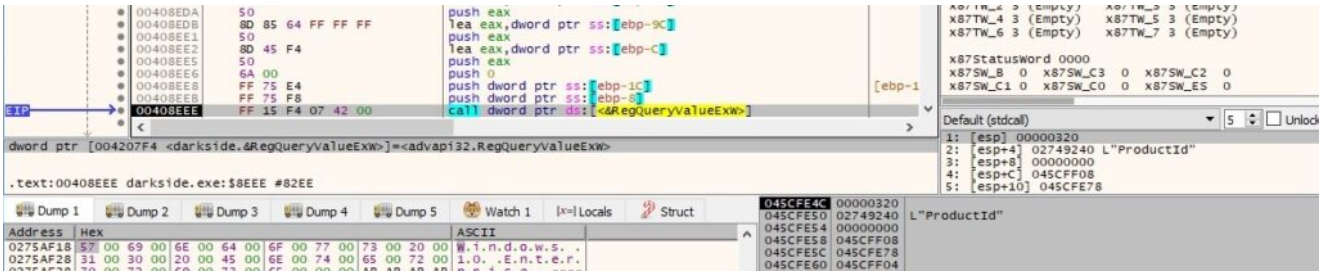


Figure 89

The malware constructs the following JSON, which contains data to be exfiltrated to the C2 server:

Figure 90

The final data looks like in the following JSON form:

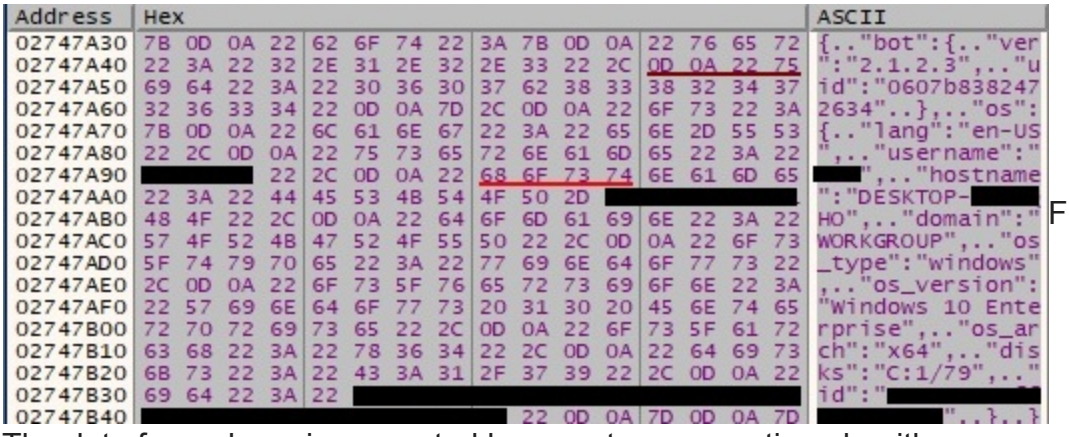


Figure 91

The data from above is encrypted by a custom encryption algorithm:

Figure 92

Address	Hex	ASCII
0273BDE8	22 52 AC 24 DF 0D D1 DD 73 83 BE D9 64 A1 EE C7	"R¬$ß.ÑÝs.¾Ùd¡îÇ
0273BDF8	53 25 AA EF 4B FF 3E 8C 59 CC AC 1A 83 B4 C1 D2	S%ªïKÿ>.YÌ¬..´ÁÒ
0273BE08	60 38 8B CB 91 29 2B 1C 71 3B B4 EA 12 18 89 87	`8.Ë.)+.q;´ê....
0273BE18	D6 5F 30 4B 15 78 AE B5 0A FC F3 61 19 DD 2E 64	Ö_0K.x®µ.üóa.Ý.d
0273BE28	38 09 1E 25 FB B3 6F 34 58 F2 7E 57 56 17 12 15	8..%û³o4Xò~WV...
0273BE38	79 1C BD AF 03 2F C3 00 B4 A8 B4 ED 30 D8 5E 88	y.½¯./Ã.´¨´í0Ø^.
0273BE48	42 BB 12 8E 45 ED 03 33 8D 5D A1 14 CF CE 22 51	B».ŽEí.3.]¡.ÏÎ"Q
0273BE58	B1 B8 2B D9 06 4B 5C 4C DF 1A C1 07 67 D8 D2 46	±¸+Ù.K\Lß.Á.gØÒF
0273BE68	78 72 98 B3 02 30 67 78 CB A9 E6 12 F0 44 AB 0D	xr.³.0gxË©æ.ðD«.
0273BE78	AD 34 A9 4C 85 DB DD EE 25 AB 0B 2D 59 2C 6F B5	.4©L.ÛÝî%«.-Y,oµ
0273BE88	FA AB F1 3F E1 37 D5 6A B2 37 82 19 AA BF 18 BB	ú«ñ?á7Õj²7..ª¿.»
0273BE98	0D 48 01 70 F6 38 5D D8 A0 FF EA A9 4C EA 09 61	.H.pö8]Ø ÿê©Lê.a
0273BEA8	D2 DF 5C 50 DB 36 17 8F 21 34 22 A8 1D 9A 70 E1	Òß\PÛ6..!4"¨..pá
0273BEB8	83 CD 72 11 10 BA 39 E3 1A 85 41 CE A0 07 05 8A	.Ír..º9ã..AÎ ....
0273BEC8	BB ED 0B A5 11 C1 79 EF 56 E2 D8 4D 49 90 28 95	»í.¥.ÁyïVâØMI.(.
0273BED8	A5 97 F9 36 E9 CA 06 39 B6 CD 0D E8 1B 2F FD AF	¥.ù6éÊ.9¶Í.è./ý¯
0273BEE8	7C D3 BC B6 C5 88 28 B1 15 FD F3 57 DD 20 4B 1F	|Ó¼¶Å.(±.ýóWÝ K.
0273BEF8	D2 38 53 74 13 D7 3E E9 68 98 C8 AB 82 B6 21 CA	Ò8St.×>éh.È«.¶!Ê

Figure 93

The result of the encryption operation is base64-encoded, as shown below:

```
.text:0040133A mov     eax, 'DCBA'
.text:0040133F stosd
.text:00401340 mov     eax, 'HGFE'
.text:00401345 stosd
.text:00401346 mov     eax, 'LKJI'
.text:0040134B stosd
.text:0040134C mov     eax, 'PONM'
.text:00401351 stosd
.text:00401352 mov     eax, 'TSRQ'
.text:00401357 stosd
.text:00401358 mov     eax, 'XWVU'
.text:0040135D stosd
.text:0040135E mov     eax, 'baZY'
.text:00401363 stosd
.text:00401364 mov     eax, 'fedc'
.text:00401369 stosd
.text:0040136A mov     eax, 'jihg'
.text:0040136F stosd
.text:00401370 mov     eax, 'nmlk'
.text:00401375 stosd
.text:00401376 mov     eax, 'rqpo'
.text:0040137B stosd
.text:0040137C mov     eax, 'vuts'
.text:00401381 stosd
.text:00401382 mov     eax, 'zyxw'
.text:00401387 stosd
.text:00401388 mov     eax, '3210'
.text:0040138D stosd
.text:0040138E mov     eax, '7654'
.text:00401393 stosd
.text:00401394 mov     eax, '/+98'
.text:00401399 stosd
.text:0040139A mov     esi, [ebp+arg_0]
.text:0040139D mov     eax, [ebp+arg_4]
.text:004013A0 mov     [ebp+var_4], eax
.text:004013A3 mov     edi, [ebp+arg_8]
```

Figure 94

```
.text:004013A6
.text:004013A6 loc_4013A6:
.text:004013A6 cmp     [ebp+var_4], 0
.text:004013AA jnz     short loc_4013AE
```

Address	Hex	ASCII
0273E0E8	49 6C 4B 73 4A 4E 38 4E 30 64 31 7A 67 37 37 5A	IlKsJN8N0d1zg77Z
0273E0F8	5A 4B 48 75 78 31 4D 6C 71 75 39 4C 2F 7A 36 4D	ZKHux1Mlqu9L/z6M
0273E108	57 63 79 73 47 6F 4F 30 77 64 4A 67 4F 49 76 4C	WcysGoO0wdJgOIvL
0273E118	6B 53 6B 72 48 48 45 37 74 4F 6F 53 47 49 6D 48	kSkrHHE7tOoSGImH
0273E128	31 6C 38 77 53 78 56 34 72 72 55 4B 2F 50 4E 68	1l8wSxV4rrUK/PNh
0273E138	47 64 30 75 5A 44 67 4A 48 69 58 37 73 32 38 30	Gd0uZDgJHiX7s280
0273E148	57 50 4A 2B 56 31 59 58 45 68 56 35 48 4C 32 76	WPJ+V1YXEhV5HL2v
0273E158	41 79 2F 44 41 4C 53 6F 74 4F 30 77 32 46 36 49	Ay/DALSotO0w2F6I
0273E168	51 72 73 53 6A 6B 58 74 41 7A 4F 4E 58 61 45 55	QrsSjkXtAzONXaEU
0273E178	7A 38 34 69 55 62 47 34 4B 39 6B 47 53 31 78 4D	z84iUbG4K9kGS1xM
0273E188	33 78 72 42 42 32 66 59 30 6B 5A 34 63 70 69 7A	3xrBB2fY0kZ4cpiz
0273E198	41 6A 42 6E 65 4D 75 70 35 68 4C 77 52 4B 73 4E	AjBneMup5hLwRKsN
0273E1A8	72 54 53 70 54 49 58 62 33 65 34 6C 71 77 73 74	rTSpTIXb3e4lqwst
0273E1B8	57 53 78 76 74 66 71 72 38 54 2F 68 4E 39 56 71	WSxvtfqr8T/hN9Vq
0273E1C8	73 6A 65 43 47 61 71 2F 47 4C 73 4E 53 41 46 77	sjeCGaq/GLsNSAFw
0273E1D8	39 6A 68 64 32 4B 44 2F 36 71 6C 4D 36 67 6C 68	9jhd2KD/6qlM6glh
0273E1E8	30 74 39 63 55 4E 73 32 46 34 38 68 4E 43 4B 6F	0t9cUNs2F48hNCKo
0273E1F8	48 5A 70 77 34 59 50 4E 63 68 45 51 75 6A 6E 6A	HZpw4YPNchEQujnj
0273E208	47 6F 56 42 7A 71 41 48 42 59 71 37 37 51 75 6C	GoVBzqAHBYq77Qul
0273E218	45 63 46 35 37 31 62 69 32 45 31 4A 6B 43 69 56	EcF571bi2E1JkCiV

Figure 95

The following function is used to generate 2 random 4-byte values that will be utilized in the network communications. It uses instructions such as RDRAND and RDSEED to generate random numbers (if these are supported), but we'll provide a deeper understanding of it when we discuss file encryption (it's also used to generate the Salsa20 matrix):

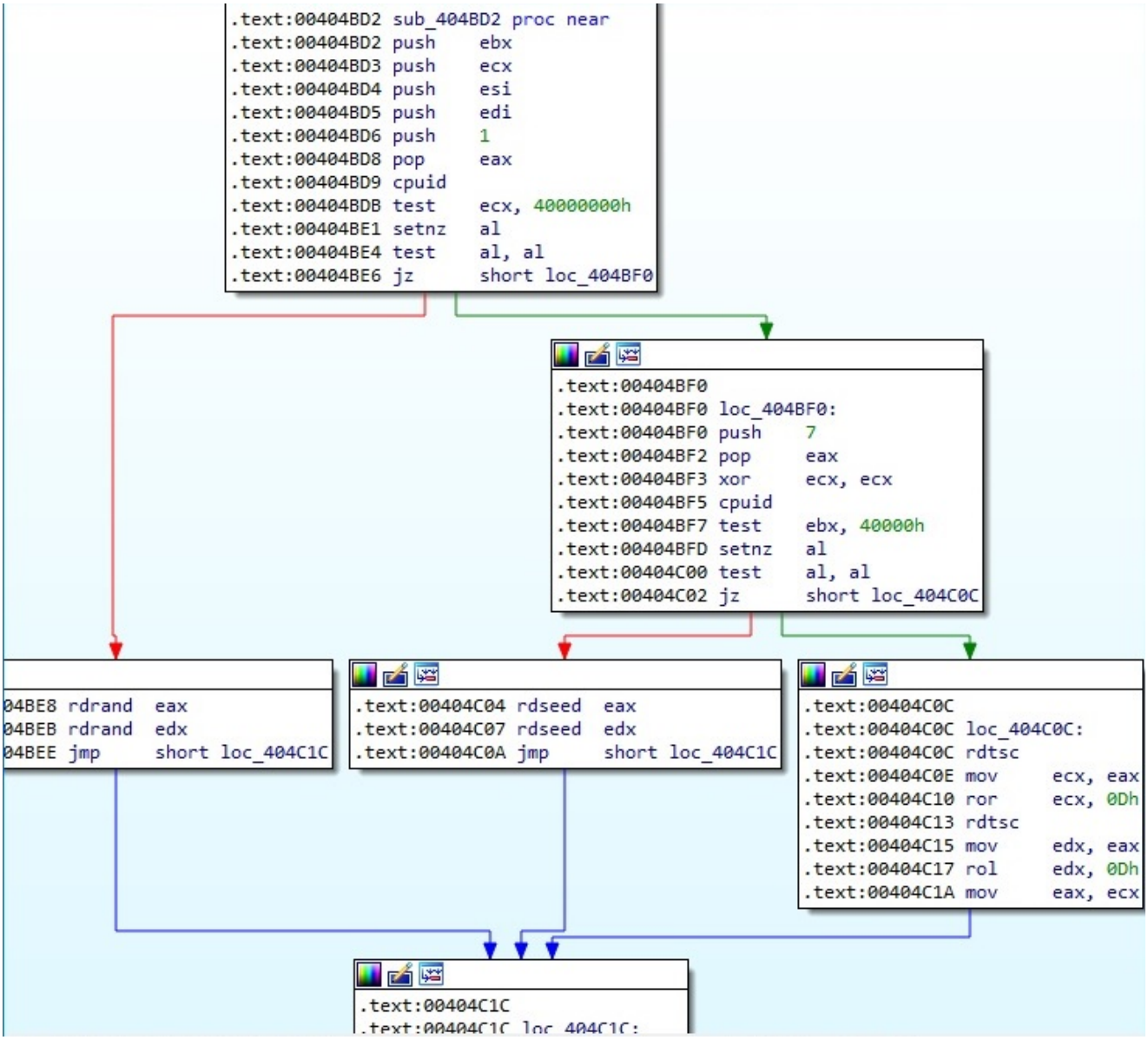
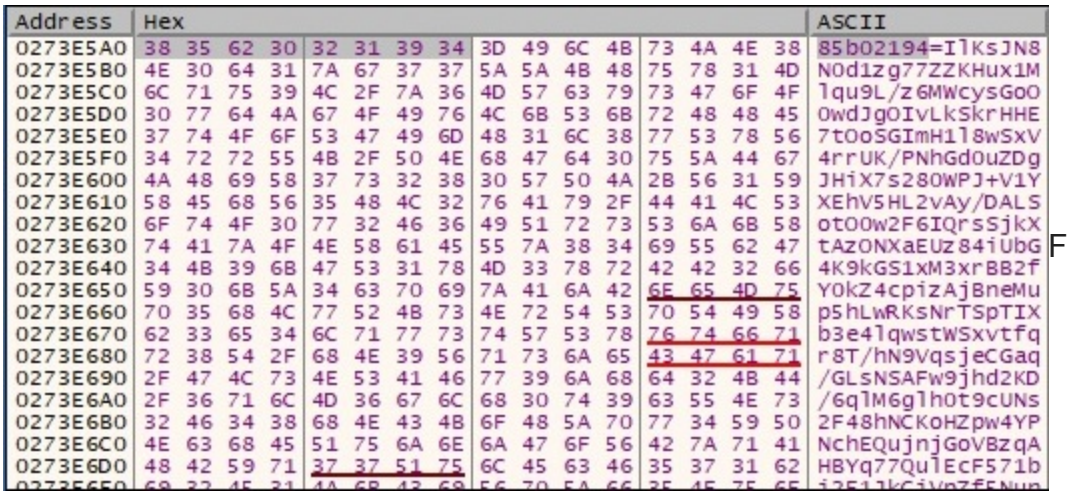


Figure 96

The parameters of the network request have the following structure:
random_number1=base64(encryptionresult)&random_number2=victim_uid:



Address	Hex																ASCII
0273E5A0	38	35	62	30	32	31	39	34	3D	49	6C	4B	73	4A	4E	38	85b02194=IlKsJN8
0273E5B0	4E	30	64	31	7A	67	37	37	5A	5A	4B	48	75	78	31	4D	N0d1zg77ZZKHux1M
0273E5C0	6C	71	75	39	4C	2F	7A	36	4D	57	63	79	73	47	6F	4F	lqu9L/z6MWcysGoO
0273E5D0	30	77	64	4A	67	4F	49	76	4C	6B	53	6B	72	48	48	45	0wdJgOIvLkSkrHHE
0273E5E0	37	74	4F	6F	53	47	49	6D	48	31	6C	38	77	53	78	56	7tOoSGImH1l8wSxV
0273E5F0	34	72	72	55	4B	2F	50	4E	68	47	64	30	75	5A	44	67	4rrUK/PNhGd0uZDg
0273E600	4A	48	69	58	37	73	32	38	30	57	50	4A	2B	56	31	59	JHiX7s280WPJ+V1Y
0273E610	58	45	68	56	35	48	4C	32	76	41	79	2F	44	41	4C	53	XEhV5HL2vAy/DALS
0273E620	6F	74	4F	30	77	32	46	36	49	51	72	73	53	6A	6B	58	otO0w2F6IQrsSjkX
0273E630	74	41	7A	4F	4E	58	61	45	55	7A	38	34	69	55	62	47	tAzONXaEUz84iUbG
0273E640	34	4B	39	6B	47	53	31	78	4D	33	78	72	42	42	32	66	4K9kGS1xM3xrBB2f
0273E650	59	30	6B	5A	34	63	70	69	7A	41	6A	42	6E	65	4D	75	Y0kZ4cpizAjBneMu
0273E660	70	35	68	4C	77	52	4B	73	4E	72	54	53	70	54	49	58	p5hLwRKsNrTSpTIX
0273E670	62	33	65	34	6C	71	77	73	74	57	53	78	76	74	66	71	b3e4lqwstWSxvtfq
0273E680	72	38	54	2F	68	4E	39	56	71	73	6A	65	43	47	61	71	r8T/hN9VqsjeCGaq
0273E690	2F	47	4C	73	4E	53	41	46	77	39	6A	68	64	32	4B	44	/GLsNSAFw9jhd2KD
0273E6A0	2F	36	71	6C	4D	36	67	6C	68	30	74	39	63	55	4E	73	/6qlM6glh0t9cUNs
0273E6B0	32	46	34	38	68	4E	43	4B	6F	48	5A	70	77	34	59	50	2F48hNCKoHZpw4YP
0273E6C0	4E	63	68	45	51	75	6A	6E	6A	47	6F	56	42	7A	71	41	NchEQujnjGoVBzqA
0273E6D0	48	42	59	71	37	37	51	75	6C	45	63	46	35	37	31	62	HBYq77QulEcF571b

Figure 97

The InternetOpenW function is called using a user agent decrypted by the malware as a parameter:

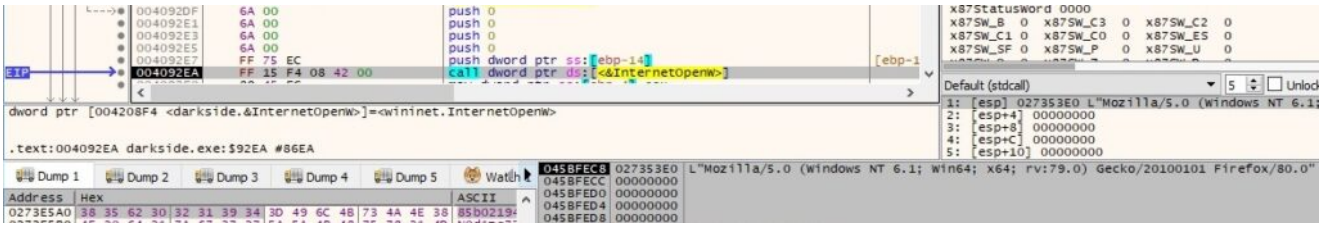


Figure 98

InternetConnectW is utilized to connect to one of the C2 servers (baroquetees[.]com) on port 443:

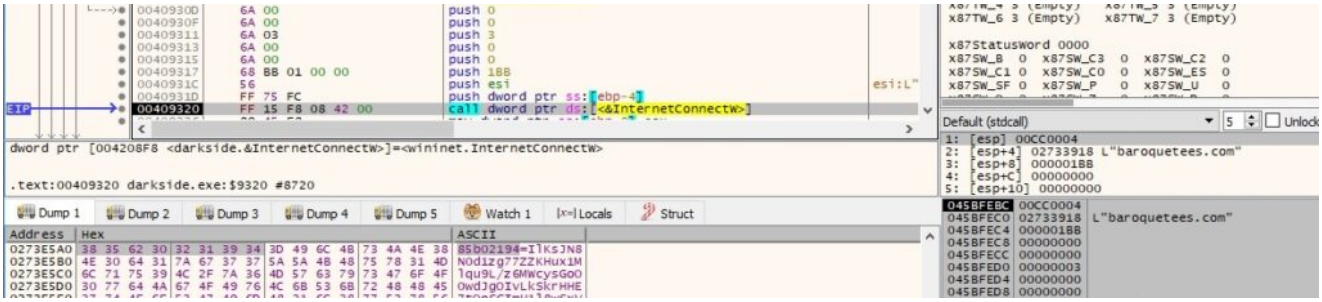


Figure 99

The process creates an HTTP request handle using the HttpOpenRequestW routine, as shown in figure 100:

Figure 100

There is also a call to the InternetSetOptionW API that is used to set the security flags for the handle (0x1f = **INTERNET_OPTION_SECURITY_FLAGS**):

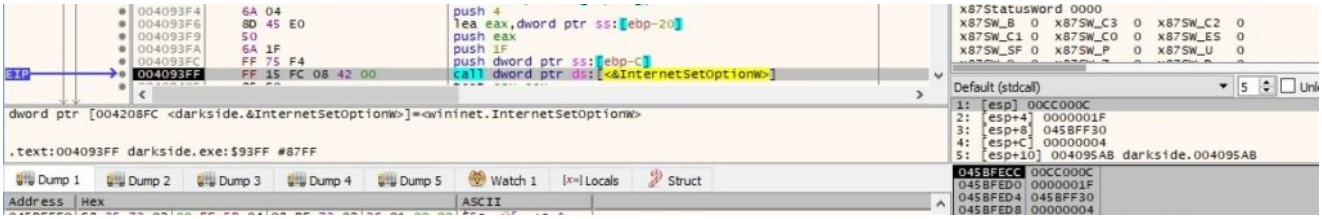


Figure 101

The binary sends the POST request to the C2 server using HttpSendRequestW:

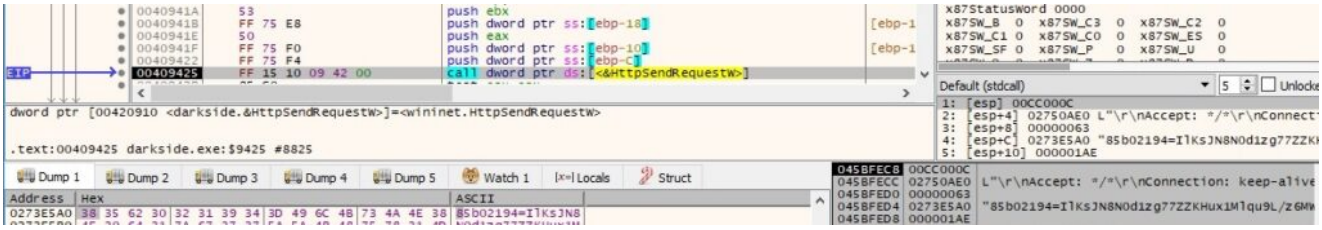


Figure 102

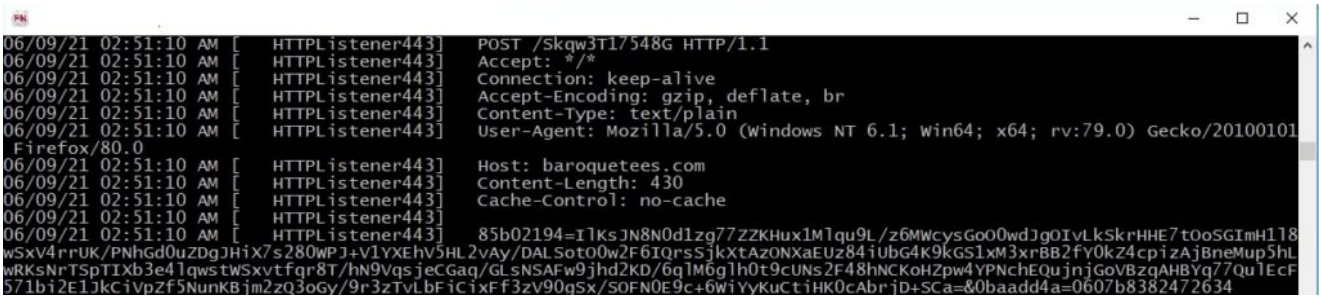


Figure 103

The status code returned by the server is retrieved using the HttpQueryInfoW API (0x13 = **HTTP_QUERY_STATUS_CODE**):

Figure 104

Interestingly, the ransomware doesn't expect a 200 status code but a 500 (Internal Server Error). If the status code isn't 500, then the process repeats the steps described so far using the second C2 server, rumahsia[.]com:

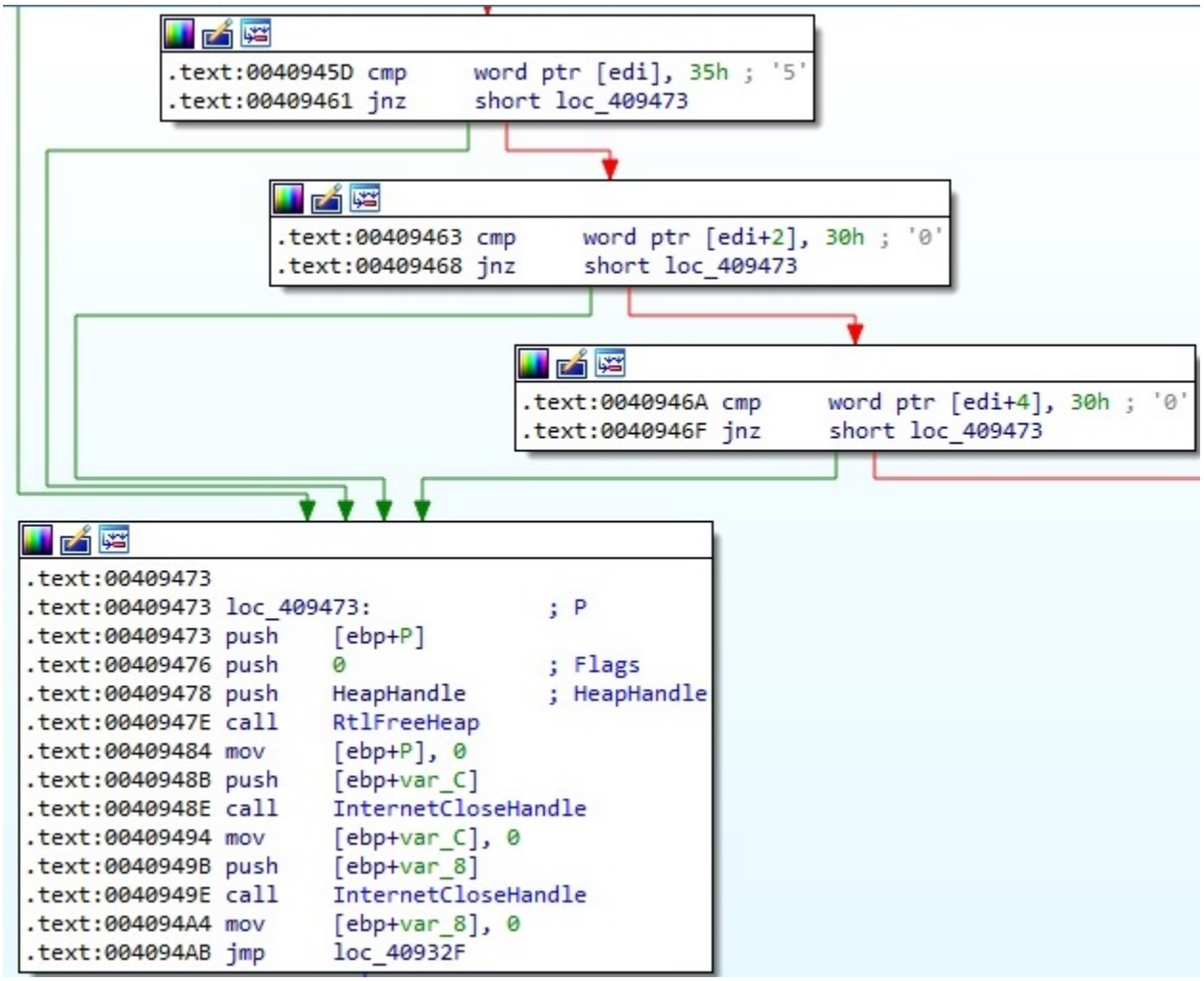


Figure 105

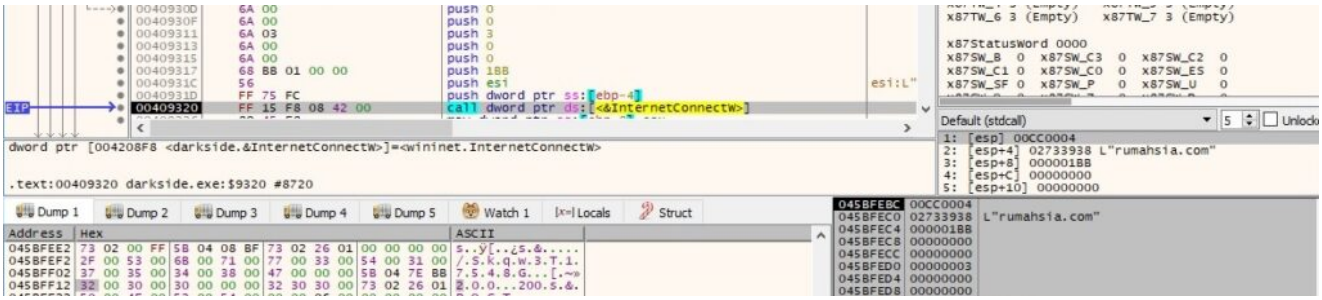


Figure 106

This last idea concludes our analysis of this thread. We continue to analyze the main thread.

The binary enumerates the volumes available on the machine and uses the CreateFileW routine to open them:

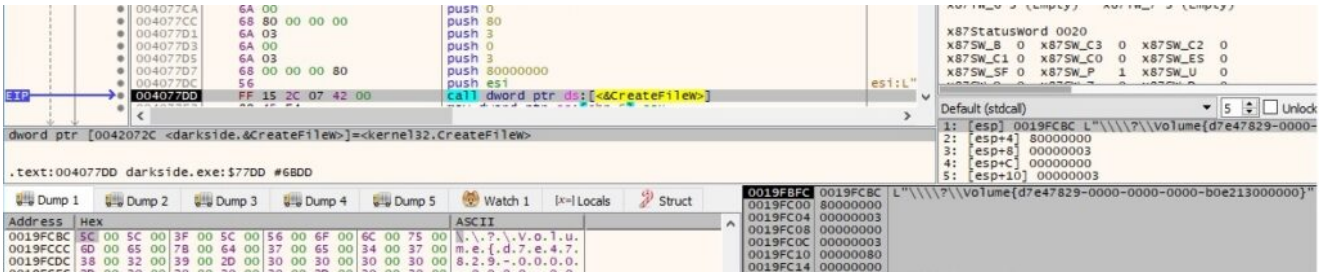


Figure 107

DeviceIoControl is utilized to get information about the type, size, and nature of a disk partition (0x70048 = **IOCTL_DISK_GET_PARTITION_INFO_EX**):

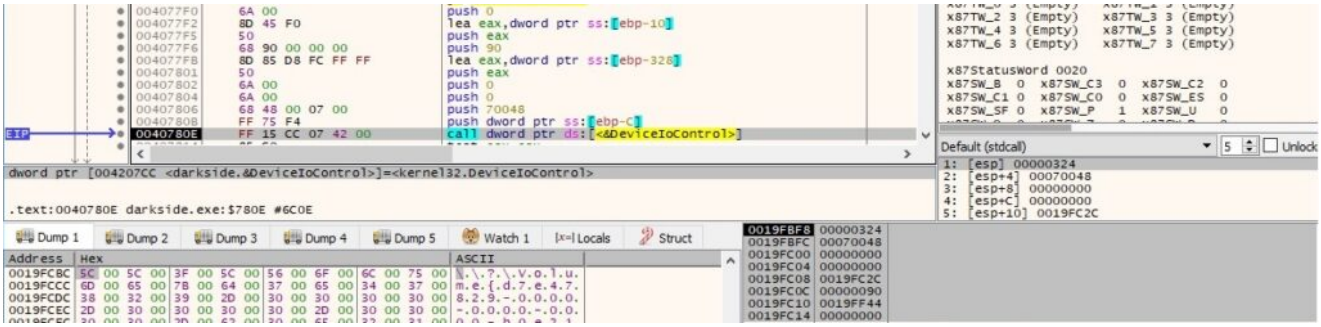


Figure 108

A new thread is created by the file using CreateThread:

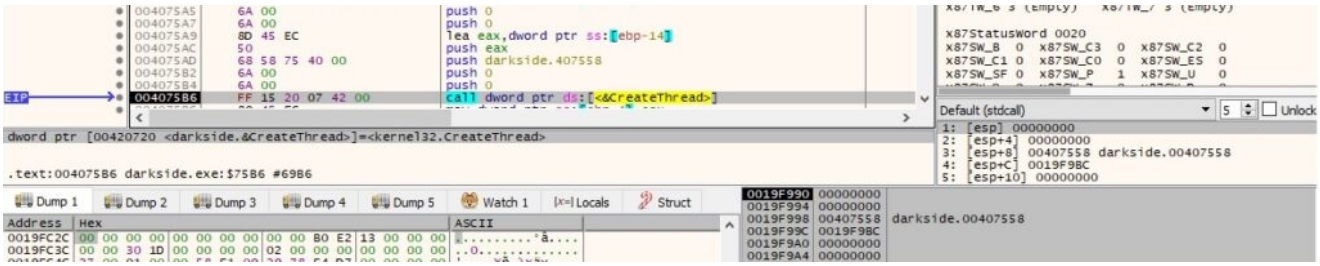


Figure 109

Thread activity – sub_407558

The only action the thread does is using the GetLogicalDriveStringsW API to retrieve the valid drives on the local machine:

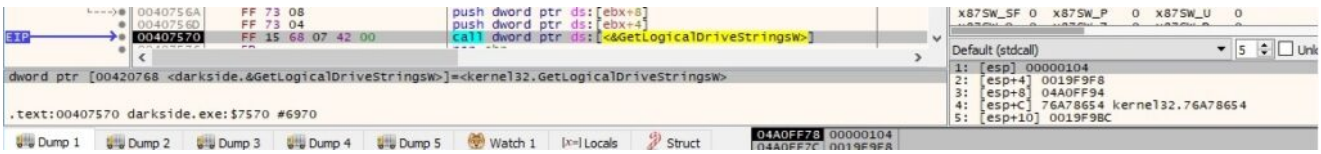


Figure 110

If a volume doesn't have a drive letter associated with it, then the ransomware does that using the SetVolumeMountPointW API, as highlighted in the following picture:

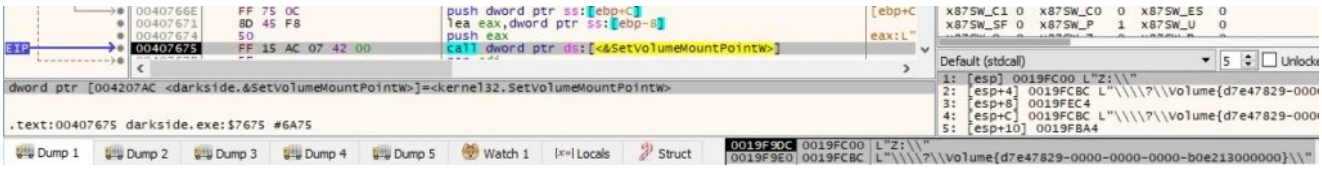


Figure 111

The malicious process targets the following types of drives –
DRIVE_REMOVABLE (0x2), **DRIVE_FIXED** (0x3) and **DRIVE_REMOTE** (0x4):

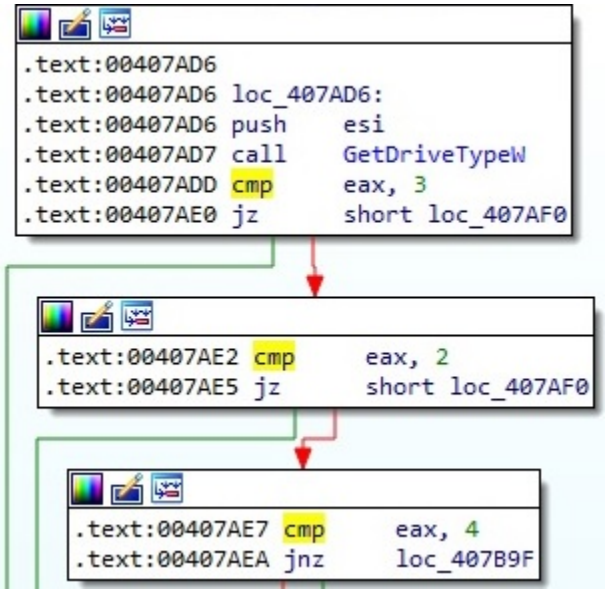


Figure 112

The CreateFileMappingW function is used to create a named file mapping object (name “Local\\job0-<Process Id>” means the object is created in the session namespace):

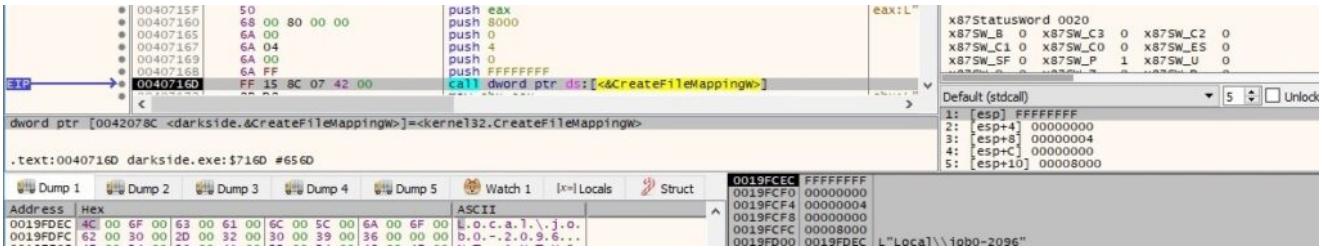


Figure 113

The binary maps a view of the file mapping into the address space of the process by calling the MapViewOfFile routine (0xf001f = **FILE_MAP_ALL_ACCESS**):

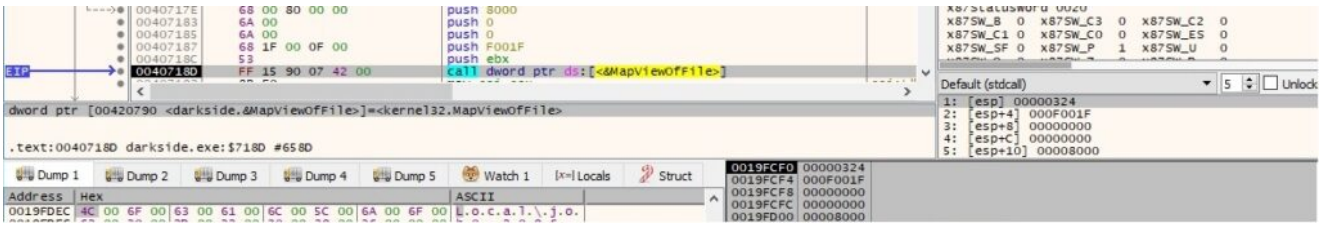


Figure 114

A named event object called “Local\\job0-<Process Id>-Event” is created by the binary:

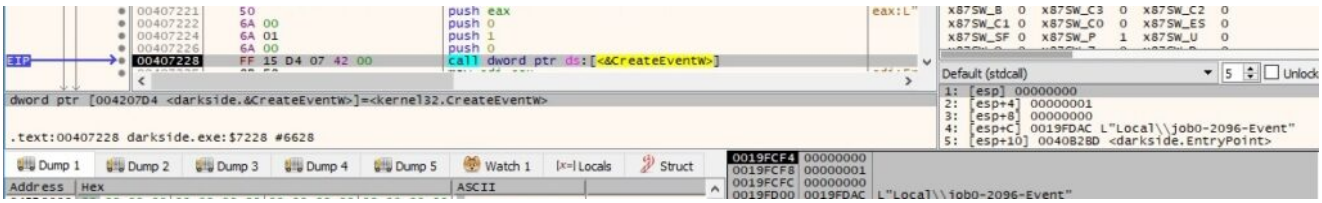


Figure 115

The ransomware launches itself with 3 parameters, and the new process will execute the encryption operations:

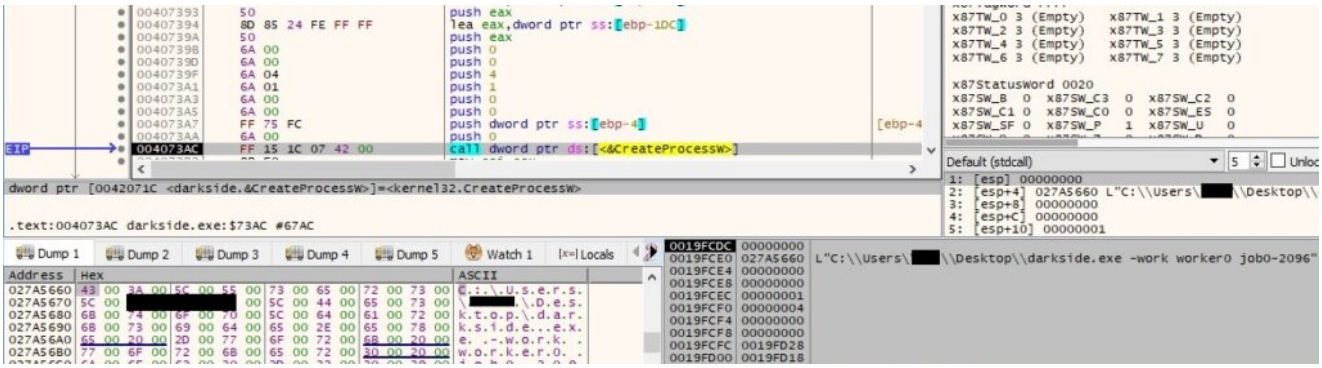


Figure 116

OpenMutexW is utilized to open a named mutex called “Global\\T-job0-<Process Id>” (which doesn’t exist at this time) – 0x100000 = **SYNCHRONIZE**:

Figure 117

The event object created earlier is opened by calling the OpenEventW API (0x1f0003 = **EVENT_ALL_ACCESS**), as displayed in figure 118:

Figure 118

The file creates an I/O completion port that isn’t associated with a file handle, which will be used by the main thread to send data that will be encrypted to worker threads:

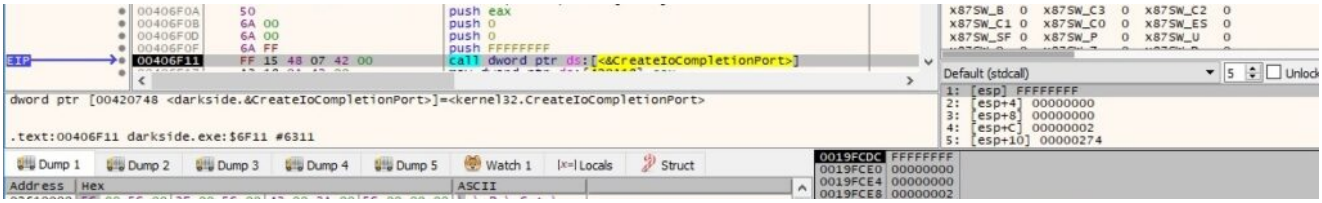


Figure 119

Two different threads that will take care of the files’ encryption are created using the CreateThread routine:

Figure 120

The ransom note README<RansomPseudoValue>.TXT is created and populated in every directory the malware encrypts:

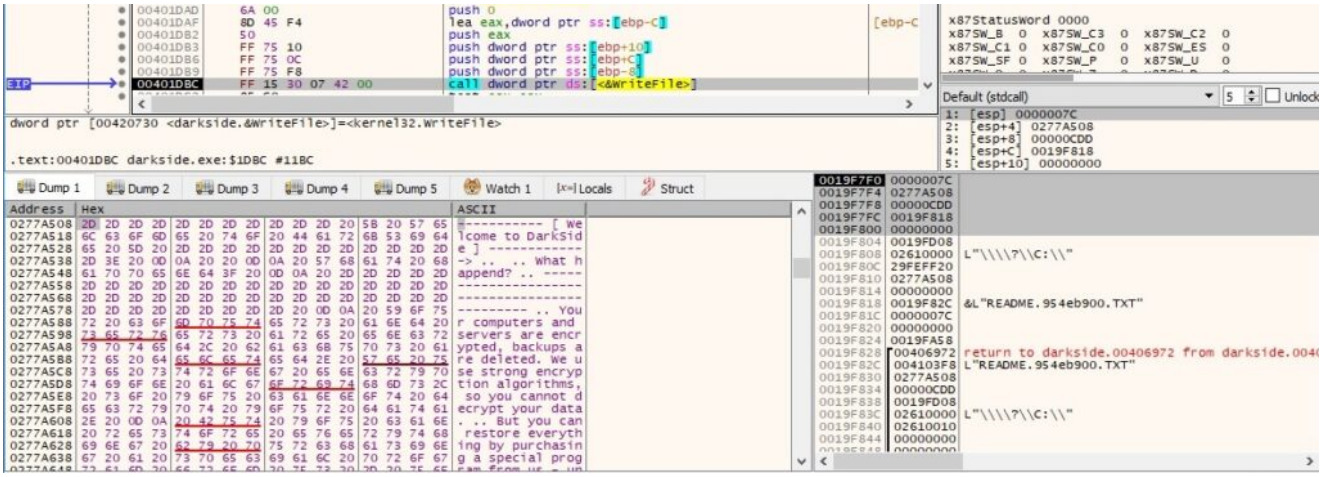


Figure 121

The process doesn't encrypt some certain files, as displayed in the next figure:

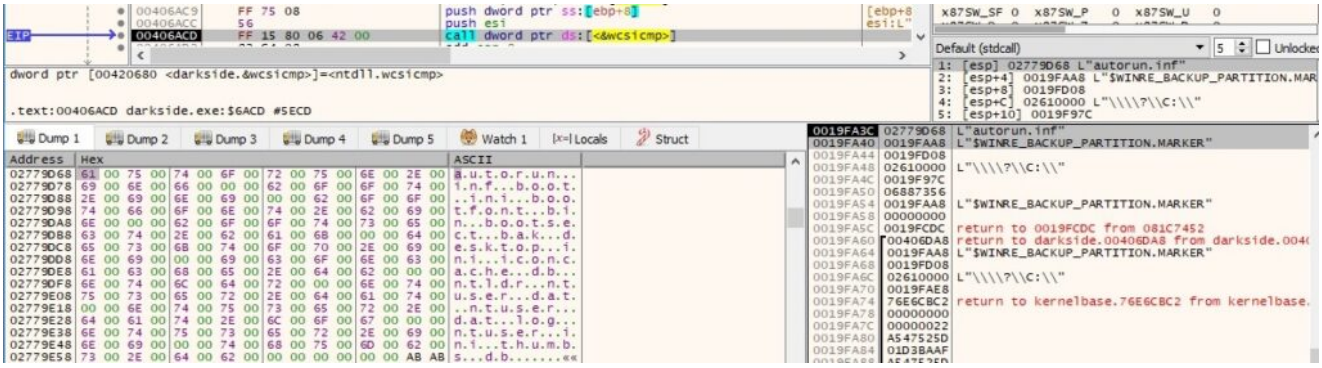


Figure 122

A list of file extensions decrypted at the beginning of the execution is also excluded from the encryption process:

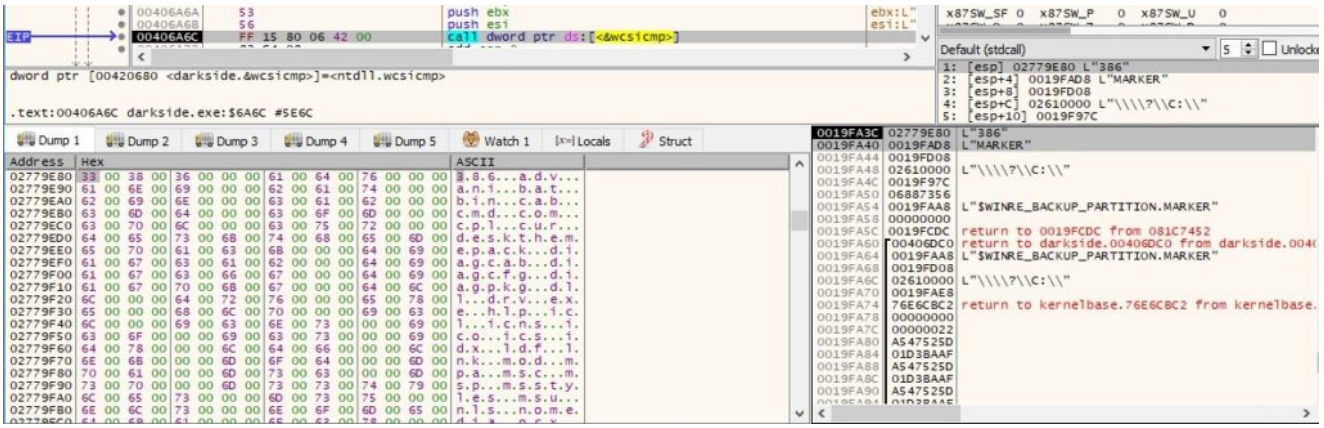


Figure 123

Every targeted file is opened and read using the CreateFileW and ReadFile functions:

Figure 124

Figure 125

The file extension is changed to also include <RansomPseudoValue>, as shown below:

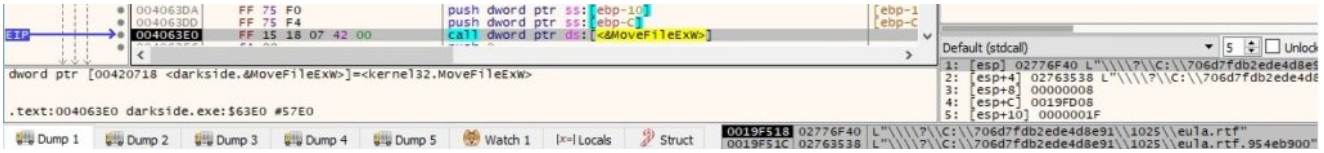


Figure 126

There is a second function call to CreateIoCompletionPort that associates the existing I/O completion port with the FileHandle parameter:

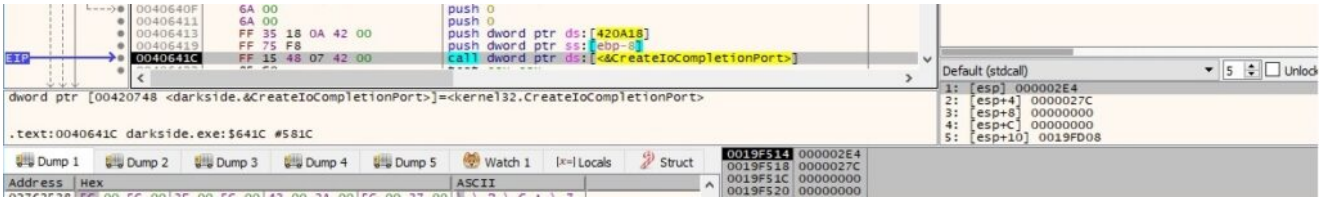


Figure 127

The RSA public exponent and the RSA modulus will be used in the encryption process of the Salsa20 matrix, as we'll describe later on:

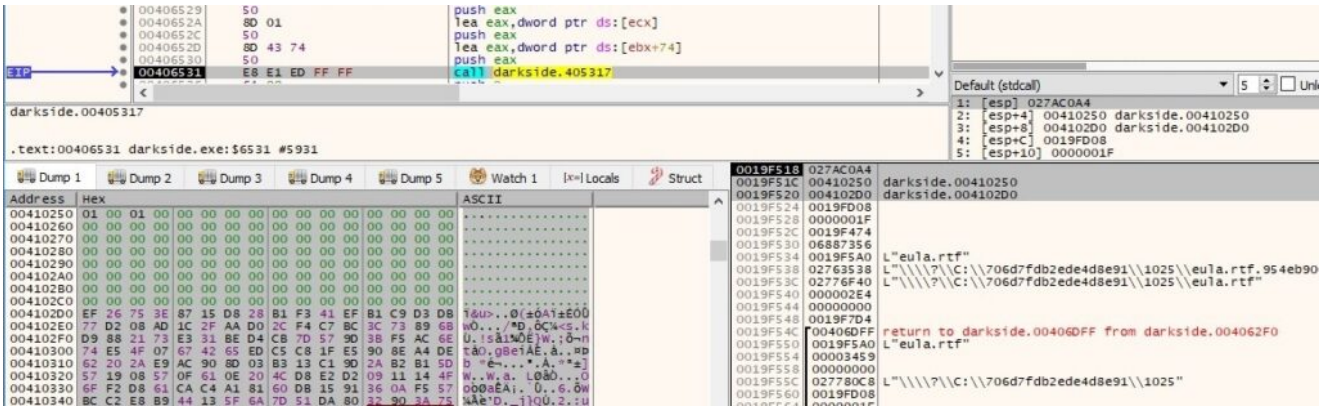


Figure 128

The ransomware checks to see if the RDRAND and RDSEED instructions are supported by the processor. If that's the case, it will use one of them to generate 56 random bytes, and 8 NULL bytes are added to the resulting buffer (Salsa20 matrix -> custom Salsa20 implementation). If none of these are supported, the malware uses the rdtsc instruction to generate deterministic timestamps that will provide a 64-byte Salsa20 matrix:

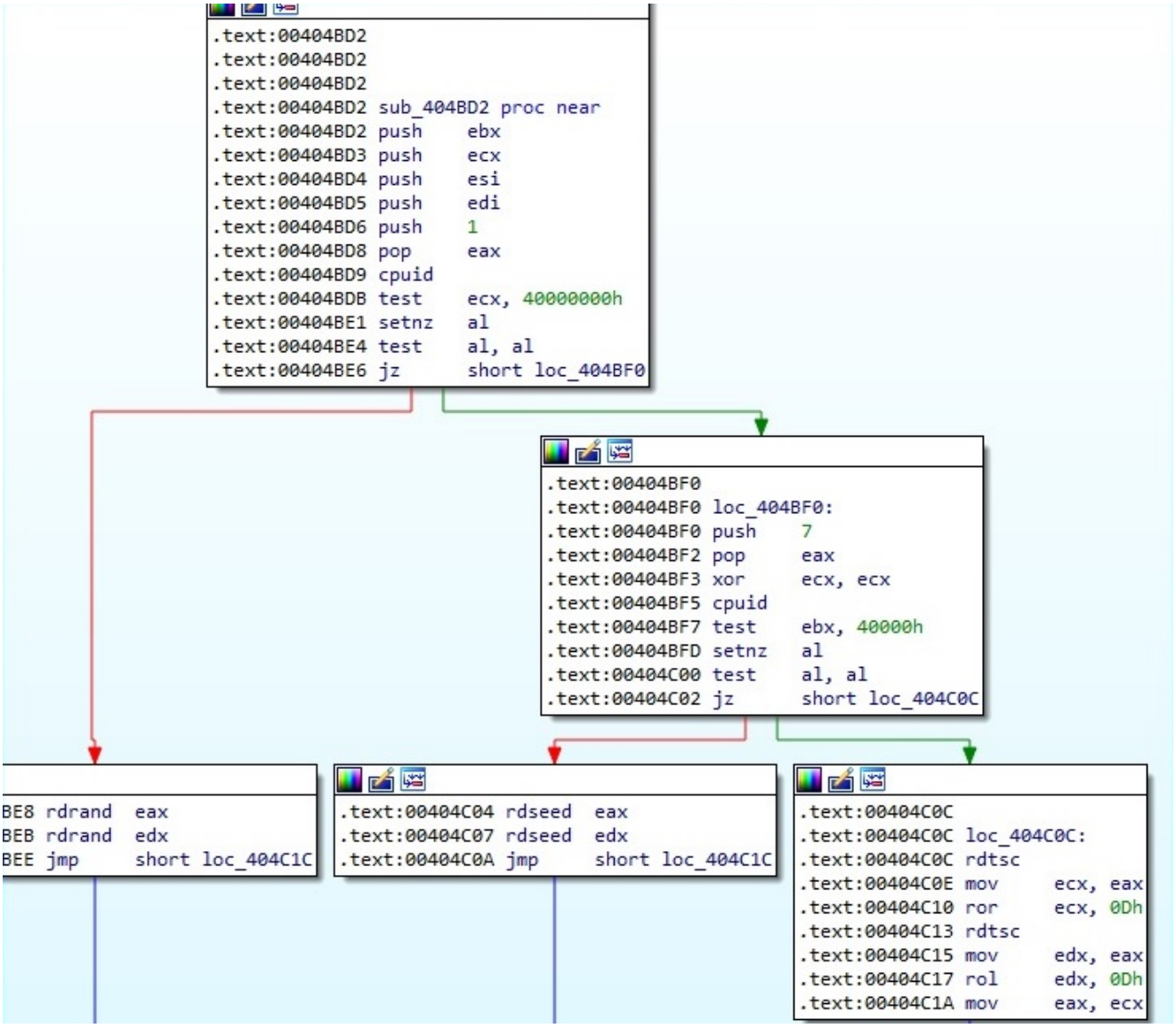
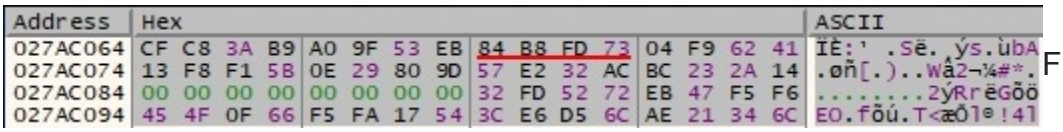


Figure 129



Address	Hex	ASCII
027AC064	CF C8 3A B9 A0 9F 53 EB 84 B8 FD 73 04 F9 62 41	ÏÈ:¹ .Së.¸ýs.ùbA
027AC074	13 F8 F1 5B 0E 29 80 9D 57 E2 32 AC BC 23 2A 14	.øñ[.)..Wâ2¬¼#*.
027AC084	00 00 00 00 00 00 00 00 32 FD 52 72 EB 47 F5 F6	........2ýRrëGõö
027AC094	45 4F 0F 66 F5 FA 17 54 3C E6 D5 6C AE 21 34 6C	EO.fõú.T<æÕl®!4l

Figure 130

The thread poses a custom implementation of the RSA-1024 algorithm (it doesn't rely on Windows APIs). Basically, the data d will produce a ciphertext = (d^exponent)%modulus. The raw modulus calculation is performed using addition and subtraction and part of the implementation is presented in the following figures:

```
.text:00405258 sub_405258 proc near
.text:00405258 mov     eax, [esi]
.text:0040525A mov     ebx, [esi+4]
.text:0040525D mov     ecx, [esi+8]
.text:00405260 mov     edx, [esi+0Ch]
.text:00405263 sbb     [edi], eax
.text:00405265 sbb     [edi+4], ebx
.text:00405268 sbb     [edi+8], ecx
.text:0040526B sbb     [edi+0Ch], edx
.text:0040526E mov     eax, [esi+10h]
.text:00405271 mov     ebx, [esi+14h]
.text:00405274 mov     ecx, [esi+18h]
.text:00405277 mov     edx, [esi+1Ch]
.text:0040527A sbb     [edi+10h], eax
.text:0040527D sbb     [edi+14h], ebx
.text:00405280 sbb     [edi+18h], ecx
.text:00405283 sbb     [edi+1Ch], edx
.text:00405286 mov     eax, [esi+20h]
.text:00405289 mov     ebx, [esi+24h]
.text:0040528C mov     ecx, [esi+28h]
.text:0040528F mov     edx, [esi+2Ch]
.text:00405292 sbb     [edi+20h], eax
.text:00405295 sbb     [edi+24h], ebx
.text:00405298 sbb     [edi+28h], ecx
.text:0040529B sbb     [edi+2Ch], edx
.text:0040529E mov     eax, [esi+30h]
.text:004052A1 mov     ebx, [esi+34h]
.text:004052A4 mov     ecx, [esi+38h]
.text:004052A7 mov     edx, [esi+3Ch]
.text:004052AA sbb     [edi+30h], eax
.text:004052AD sbb     [edi+34h], ebx
.text:004052B0 sbb     [edi+38h], ecx
.text:004052B3 sbb     [edi+3Ch], edx
.text:004052B6 mov     eax, [esi+40h]
.text:004052B9 mov     ebx, [esi+44h]
.text:004052BC mov     ecx, [esi+48h]
.text:004052BF mov     edx, [esi+4Ch]
.text:004052C2 sbb     [edi+40h], eax
.text:004052C5 sbb     [edi+44h], ebx
.text:004052C8 sbb     [edi+48h], ecx
.text:004052CB sbb     [edi+4Ch], edx
.text:004052CE mov     eax, [esi+50h]
```

Figure 131

```
.text:00405199 sub_405199 proc near
.text:00405199 mov     eax, [esi]
.text:0040519B mov     ebx, [esi+4]
.text:0040519E mov     ecx, [esi+8]
.text:004051A1 mov     edx, [esi+0Ch]
.text:004051A4 adc     [edi], eax
.text:004051A6 adc     [edi+4], ebx
.text:004051A9 adc     [edi+8], ecx
.text:004051AC adc     [edi+0Ch], edx
.text:004051AF mov     eax, [esi+10h]
.text:004051B2 mov     ebx, [esi+14h]
.text:004051B5 mov     ecx, [esi+18h]
.text:004051B8 mov     edx, [esi+1Ch]
.text:004051BB adc     [edi+10h], eax
.text:004051BE adc     [edi+14h], ebx
.text:004051C1 adc     [edi+18h], ecx
.text:004051C4 adc     [edi+1Ch], edx
.text:004051C7 mov     eax, [esi+20h]
.text:004051CA mov     ebx, [esi+24h]
.text:004051CD mov     ecx, [esi+28h]
.text:004051D0 mov     edx, [esi+2Ch]
.text:004051D3 adc     [edi+20h], eax
.text:004051D6 adc     [edi+24h], ebx
.text:004051D9 adc     [edi+28h], ecx
.text:004051DC adc     [edi+2Ch], edx
.text:004051DF mov     eax, [esi+30h]
.text:004051E2 mov     ebx, [esi+34h]
.text:004051E5 mov     ecx, [esi+38h]
.text:004051E8 mov     edx, [esi+3Ch]
.text:004051EB adc     [edi+30h], eax
.text:004051EE adc     [edi+34h], ebx
.text:004051F1 adc     [edi+38h], ecx
.text:004051F4 adc     [edi+3Ch], edx
.text:004051F7 mov     eax, [esi+40h]
.text:004051FA mov     ebx, [esi+44h]
.text:004051FD mov     ecx, [esi+48h]
.text:00405200 mov     edx, [esi+4Ch]
.text:00405203 adc     [edi+40h], eax
.text:00405206 adc     [edi+44h], ebx
.text:00405209 adc     [edi+48h], ecx
.text:0040520C adc     [edi+4Ch], edx
.text:0040520F mov     eax, [esi+50h]
.text:00405212 mov     ebx, [esi+54h]
.text:00405215 mov     ecx, [esi+58h]
```

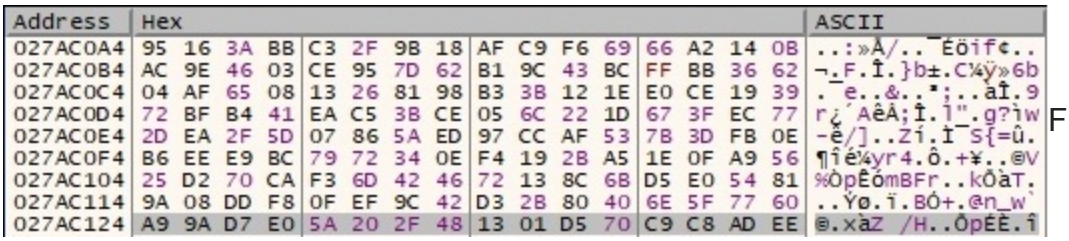
Figure 132

The Salsa20 matrix is encrypted using the custom RSA implementation, as shown in figure 133:

Address	Hex	ASCII
027AC0A4	95 16 3A BB C3 2F 9B 18 AF C9 F6 69 66 A2 14 0B	.:»Å/.. ¯ÉöifÇ..
027AC0B4	AC 9E 46 03 CE 95 7D 62 B1 9C 43 BC FF BB 36 62	¬.F.Î.}b±.C¼ÿ»6b
027AC0C4	04 AF 65 08 13 26 81 98 B3 3B 12 1E E0 CE 19 39	. ¯e..&..³;..àÎ.9
027AC0D4	72 BF B4 41 EA C5 3B CE 05 6C 22 1D 67 3F EC 77	r¿´AêÅ;Î.l".g?ìw
027AC0E4	2D EA 2F 5D 07 86 5A ED 97 CC AF 53 7B 3D FB 0E	-ê/]..Zí.Ì¯S{=û.
027AC0F4	B6 EE E9 BC 79 72 34 0E F4 19 2B A5 1E 0F A9 56	¶îé¼yr4.ô.+¥..©V
027AC104	25 D2 70 CA F3 6D 42 46 72 13 8C 6B D5 E0 54 81	%ÒpÊómBFr..kÕàT.
027AC114	9A 08 DD F8 0F EF 9C 42 D3 2B 80 40 6E 5F 77 60	..Ýø.ï.BÓ+.@n_w`

Figure 133

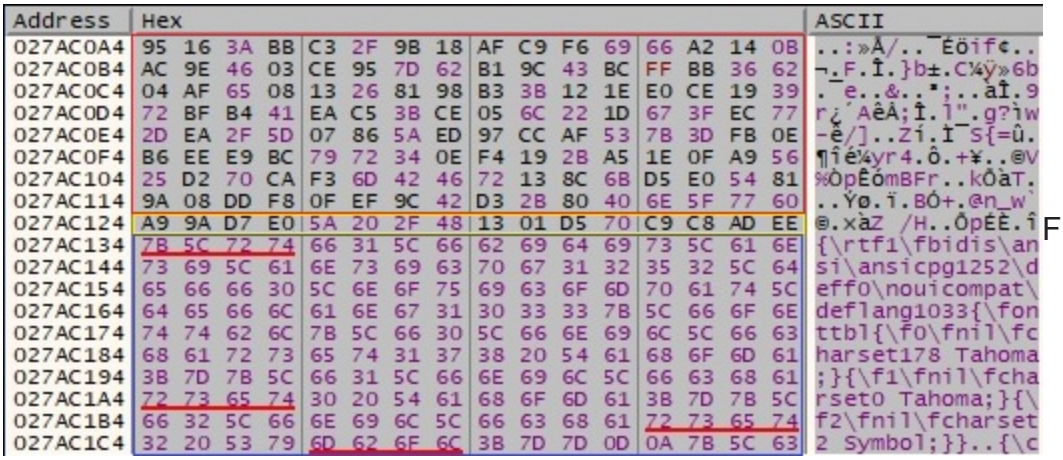
There is a custom “hash” function applied to the above encryption result, which produces a 16-byte output:



Address	Hex	ASCII
027AC0A4	95 16 3A BB C3 2F 9B 18 AF C9 F6 69 66 A2 14 0B	..:»Å/.. ¯Éöif¢..
027AC0B4	AC 9E 46 03 CE 95 7D 62 B1 9C 43 BC FF BB 36 62	¬.F.Î.}b±.C¼ÿ»6b
027AC0C4	04 AF 65 08 13 26 81 98 B3 3B 12 1E E0 CE 19 39	. ¯e..&..³;..àÎ.9
027AC0D4	72 BF B4 41 EA C5 3B CE 05 6C 22 1D 67 3F EC 77	r¿´AêÅ;Î.l"_.g?ìw
027AC0E4	2D EA 2F 5D 07 86 5A ED 97 CC AF 53 7B 3D FB 0E	-ê/]..Zí.Ì¯S{=û.
027AC0F4	B6 EE E9 BC 79 72 34 0E F4 19 2B A5 1E 0F A9 56	¶îé¼yr4.ô.+¥..©V
027AC104	25 D2 70 CA F3 6D 42 46 72 13 8C 6B D5 E0 54 81	%ÒpÊómBFr..kÕàT.
027AC114	9A 08 DD F8 0F EF 9C 42 D3 2B 80 40 6E 5F 77 60	..Ýø.ï.BÓ+.@n_w`
027AC124	A9 9A D7 E0 5A 20 2F 48 13 01 D5 70 C9 C8 AD EE	©.×àZ /H..ÕpÉÈ.î

Figure 134

The file content that will be encrypted is appended to the buffer that will be sent to the worker threads:



Address	Hex	ASCII
027AC0A4	95 16 3A BB C3 2F 9B 18 AF C9 F6 69 66 A2 14 0B	..:»Å/.. ¯Éöif¢..
027AC0B4	AC 9E 46 03 CE 95 7D 62 B1 9C 43 BC FF BB 36 62	¬.F.Î.}b±.C¼ÿ»6b
027AC0C4	04 AF 65 08 13 26 81 98 B3 3B 12 1E E0 CE 19 39	. ¯e..&..³;..àÎ.9
027AC0D4	72 BF B4 41 EA C5 3B CE 05 6C 22 1D 67 3F EC 77	r¿´AêÅ;Î.l"_.g?ìw
027AC0E4	2D EA 2F 5D 07 86 5A ED 97 CC AF 53 7B 3D FB 0E	-ê/]..Zí.Ì¯S{=û.
027AC0F4	B6 EE E9 BC 79 72 34 0E F4 19 2B A5 1E 0F A9 56	¶îé¼yr4.ô.+¥..©V
027AC104	25 D2 70 CA F3 6D 42 46 72 13 8C 6B D5 E0 54 81	%ÒpÊómBFr..kÕàT.
027AC114	9A 08 DD F8 0F EF 9C 42 D3 2B 80 40 6E 5F 77 60	..Ýø.ï.BÓ+.@n_w`
027AC124	A9 9A D7 E0 5A 20 2F 48 13 01 D5 70 C9 C8 AD EE	©.×àZ /H..ÕpÉÈ.î
027AC134	7B 5C 72 74 66 31 5C 66 62 69 64 69 73 5C 61 6E	{\rtf1\fbidis\an
027AC144	73 69 5C 61 6E 73 69 63 70 67 31 32 35 32 5C 64	si\ansicpg1252\d
027AC154	65 66 66 30 5C 6E 6F 75 69 63 6F 6D 70 61 74 5C	eff0\nouicompat\
027AC164	64 65 66 6C 61 6E 67 31 30 33 33 7B 5C 66 6F 6E	deflang1033{\fon
027AC174	74 74 62 6C 7B 5C 66 30 5C 66 6E 69 6C 5C 66 63	ttbl{\f0\fnil\fc
027AC184	68 61 72 73 65 74 31 37 38 20 54 61 68 6F 6D 61	harset178 Tahoma
027AC194	3B 7D 7B 5C 66 31 5C 66 6E 69 6C 5C 66 63 68 61	;}{\f1\fnil\fcha
027AC1A4	72 73 65 74 30 20 54 61 68 6F 6D 61 3B 7D 7B 5C	rset0 Tahoma;}{\
027AC1B4	66 32 5C 66 6E 69 6C 5C 66 63 68 61 72 73 65 74	f2\fnil\fcharset
027AC1C4	32 20 53 79 6D 62 6F 6C 3B 7D 7D 0D 0A 7B 5C 63	2 Symbol;}}..{\c

Figure 135

The Salsa20 matrix is also added to the buffer, and it will be utilized by the worker threads to encrypt the files:

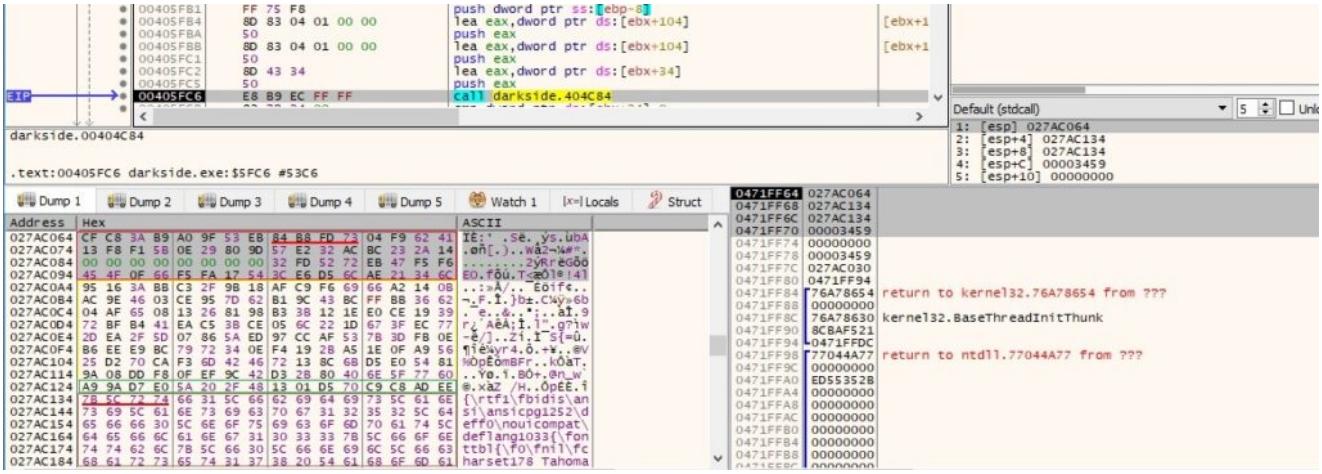


Figure 136

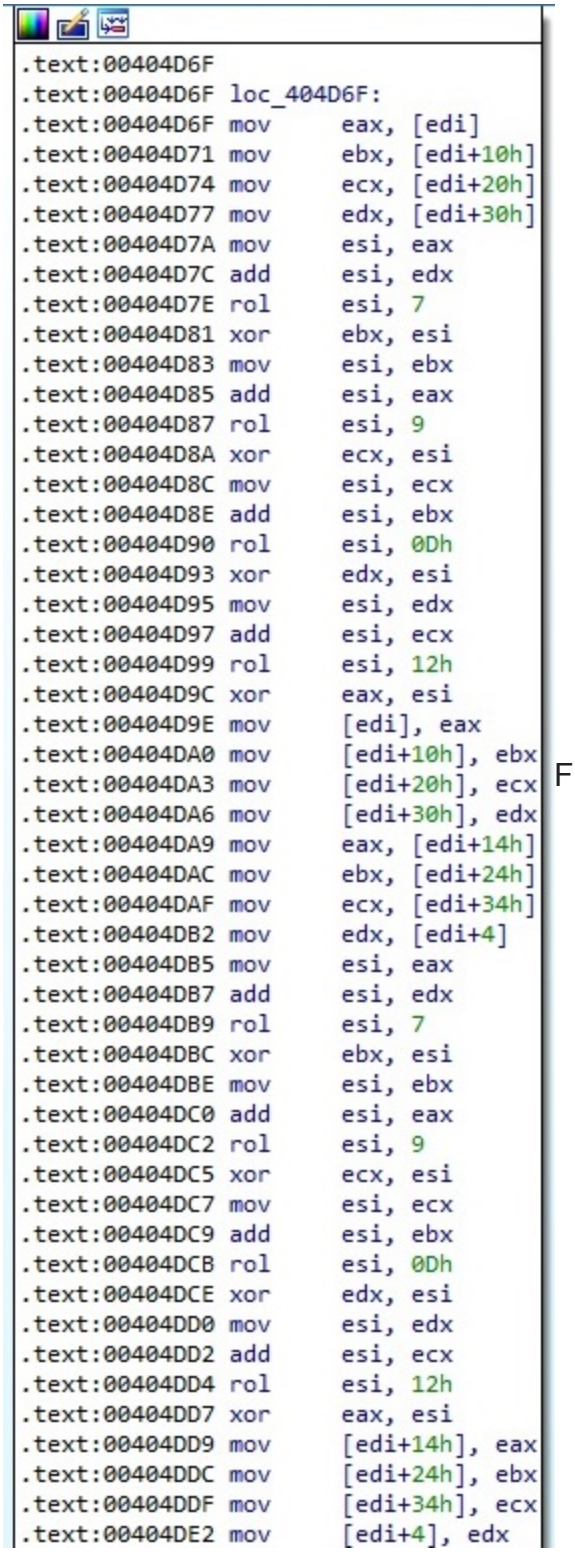
Thread activity – sub_405E7C (File encryption)

The file content is encrypted using a custom Salsa20 implementation and the ciphertext overwrites the plaintext in the buffer:

Address	Hex																ASCII
027AC064	CF	C8	3A	B9	A0	9F	53	EB	84	B8	FD	73	04	F9	62	41	ÏÈ:¹ .Së.„ýs.ùbA
027AC074	13	F8	F1	5B	0E	29	80	9D	57	E2	32	AC	BC	23	2A	14	.øñ[.)..Wâ2¬¼#*.
027AC084	D2	00	00	00	00	00	00	00	32	FD	52	72	EB	47	F5	F6	Ò.......2ýRrëGõö
027AC094	45	4F	0F	66	F5	FA	17	54	3C	E6	D5	6C	AE	21	34	6C	EO.fõú.T<æÕl®!4l
027AC0A4	95	16	3A	BB	C3	2F	9B	18	AF	C9	F6	69	66	A2	14	0B	..:»Ã/.. ¯Éöif¢..
027AC0B4	AC	9E	46	03	CE	95	7D	62	B1	9C	43	BC	FF	BB	36	62	¬.F.Î.}b±.C¼ÿ»6b
027AC0C4	04	AF	65	08	13	26	81	98	B3	3B	12	1E	E0	CE	19	39	. ¯e..&..³;..àÎ.9
027AC0D4	72	BF	B4	41	EA	C5	3B	CE	05	6C	22	1D	67	3F	EC	77	r¿´AêÅ;Î.l".g?ìw
027AC0E4	2D	EA	2F	5D	07	86	5A	ED	97	CC	AF	53	7B	3D	FB	0E	-ê/]..Zí.Ì¯S{=û.
027AC0F4	B6	EE	E9	BC	79	72	34	0E	F4	19	2B	A5	1E	0F	A9	56	¶îé¼yr4.ô.+¥..©V
027AC104	25	D2	70	CA	F3	6D	42	46	72	13	8C	6B	D5	E0	54	81	%ÒpÊómBFr..kÕàT.
027AC114	9A	08	DD	F8	0F	EF	9C	42	D3	2B	80	40	6E	5F	77	60	..Ýø.ï.BÓ+.@n_w`
027AC124	A9	9A	D7	E0	5A	20	2F	48	13	01	D5	70	C9	C8	AD	EE	©.×àZ /H..ÕpÉÈ.î
027AC134	C6	04	10	49	A5	C4	5B	BD	93	3B	7E	74	59	43	E0	05	Æ..I¥Ä[½.;~tYCà.
027AC144	22	C4	64	E3	76	50	A3	DE	DF	A8	9C	78	F1	27	09	D6	"ÄdãvP£Þß¨.xñ'.Ö
027AC154	90	8A	6C	CE	D1	80	A7	01	ED	64	36	B4	FC	4E	2F	D9	..lÎÑ.§.íd6´üN/Ù
027AC164	85	62	88	92	A1	47	88	02	03	83	A5	D6	BA	19	3F	08	.b..¡G....¥Ö°.?.
027AC174	36	9F	F9	95	FA	99	0D	31	5D	C4	36	EA	6A	B3	1B	85	6.ù.ú..1]Ä6êj³..
027AC184	05	E7	A2	A4	32	FC	A8	7E	93	7A	17	76	F1	62	C6	FD	.ç¢¤2ü¨~.z.vñbÆý

Figure 137

A snippet of the custom implementation is presented below:



```
.text:00404D6F
.text:00404D6F loc_404D6F:
.text:00404D6F mov     eax, [edi]
.text:00404D71 mov     ebx, [edi+10h]
.text:00404D74 mov     ecx, [edi+20h]
.text:00404D77 mov     edx, [edi+30h]
.text:00404D7A mov     esi, eax
.text:00404D7C add     esi, edx
.text:00404D7E rol     esi, 7
.text:00404D81 xor     ebx, esi
.text:00404D83 mov     esi, ebx
.text:00404D85 add     esi, eax
.text:00404D87 rol     esi, 9
.text:00404D8A xor     ecx, esi
.text:00404D8C mov     esi, ecx
.text:00404D8E add     esi, ebx
.text:00404D90 rol     esi, 0Dh
.text:00404D93 xor     edx, esi
.text:00404D95 mov     esi, edx
.text:00404D97 add     esi, ecx
.text:00404D99 rol     esi, 12h
.text:00404D9C xor     eax, esi
.text:00404D9E mov     [edi], eax
.text:00404DA0 mov     [edi+10h], ebx
.text:00404DA3 mov     [edi+20h], ecx
.text:00404DA6 mov     [edi+30h], edx
.text:00404DA9 mov     eax, [edi+14h]
.text:00404DAC mov     ebx, [edi+24h]
.text:00404DAF mov     ecx, [edi+34h]
.text:00404DB2 mov     edx, [edi+4]
.text:00404DB5 mov     esi, eax
.text:00404DB7 add     esi, edx
.text:00404DB9 rol     esi, 7
.text:00404DBC xor     ebx, esi
.text:00404DBE mov     esi, ebx
.text:00404DC0 add     esi, eax
.text:00404DC2 rol     esi, 9
.text:00404DC5 xor     ecx, esi
.text:00404DC7 mov     esi, ecx
.text:00404DC9 add     esi, ebx
.text:00404DCB rol     esi, 0Dh
.text:00404DCE xor     edx, esi
.text:00404DD0 mov     esi, edx
.text:00404DD2 add     esi, ecx
.text:00404DD4 rol     esi, 12h
.text:00404DD7 xor     eax, esi
.text:00404DD9 mov     [edi+14h], eax
.text:00404DDC mov     [edi+24h], ebx
.text:00404DDF mov     [edi+34h], ecx
.text:00404DE2 mov     [edi+4], edx
```

Figure 138

The encrypted content is written to the initial file, followed by the encrypted Salsa20 matrix and the hash value, as displayed in the following figures:

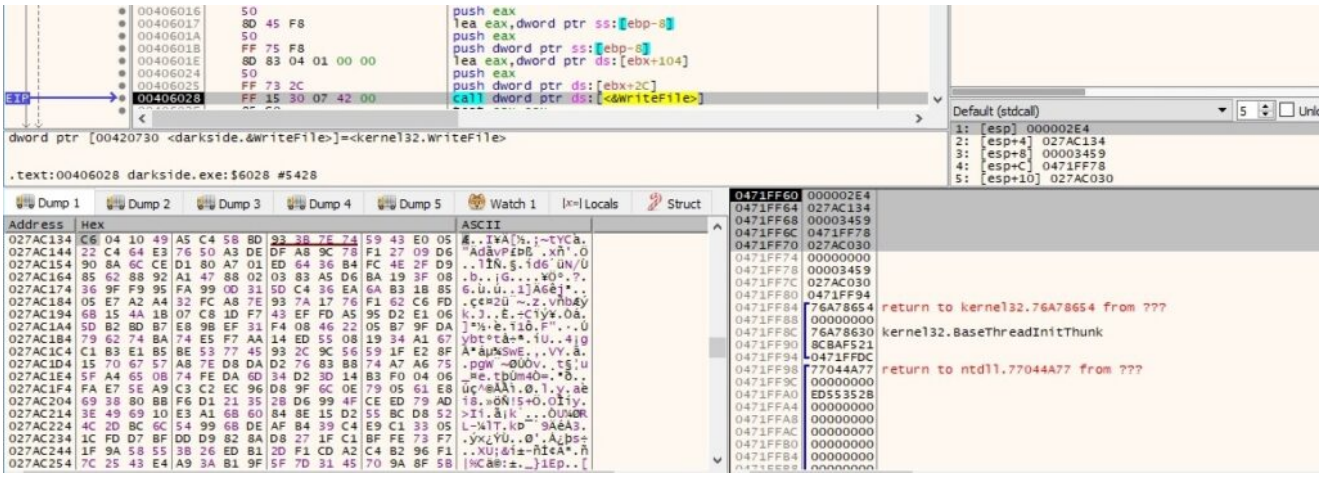


Figure 139

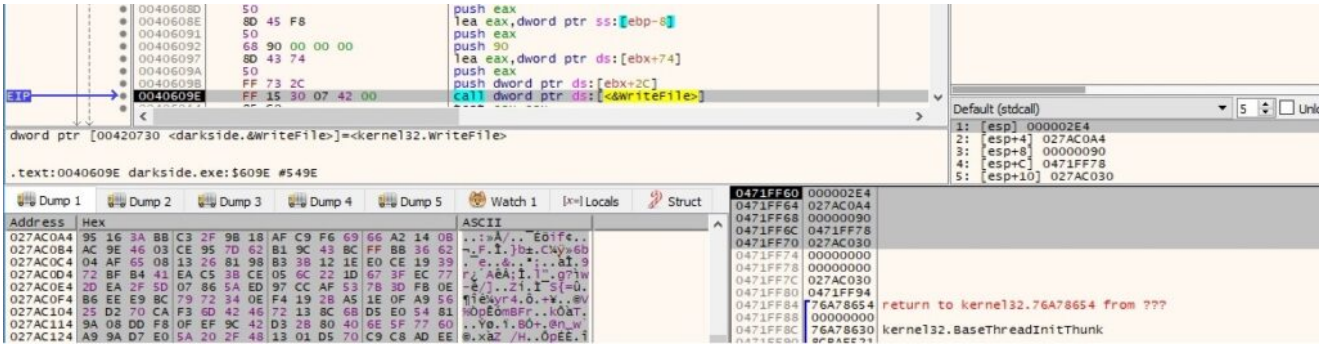


Figure 140

This last idea concludes our analysis of this thread. We continue to analyze the main thread.

If the current directory contains “backup”, then the malware deletes it:

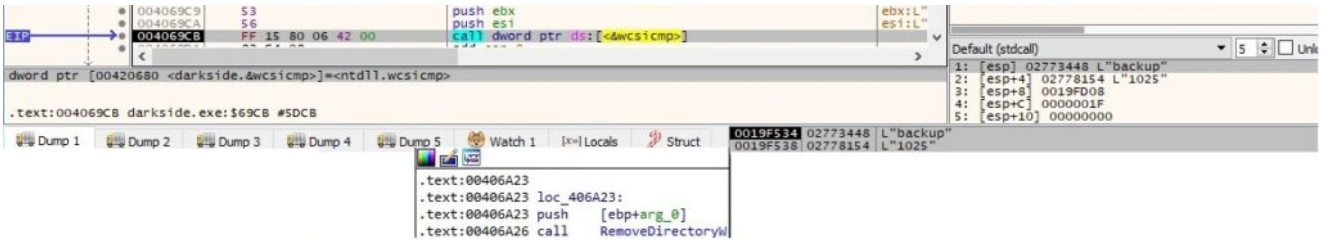


Figure 141

The main thread sends the buffer described above (which includes file content to be encrypted etc.) to the worker threads by calling the PostQueuedCompletionStatus routine:

Figure 142

We’ve also identified a function that we believe it’s used to propagate the malware to domain controllers (we didn’t have one in our environment). It calls functions such as DsGetDcNameW, DsGetDcOpenW and DsGetDcNextW:

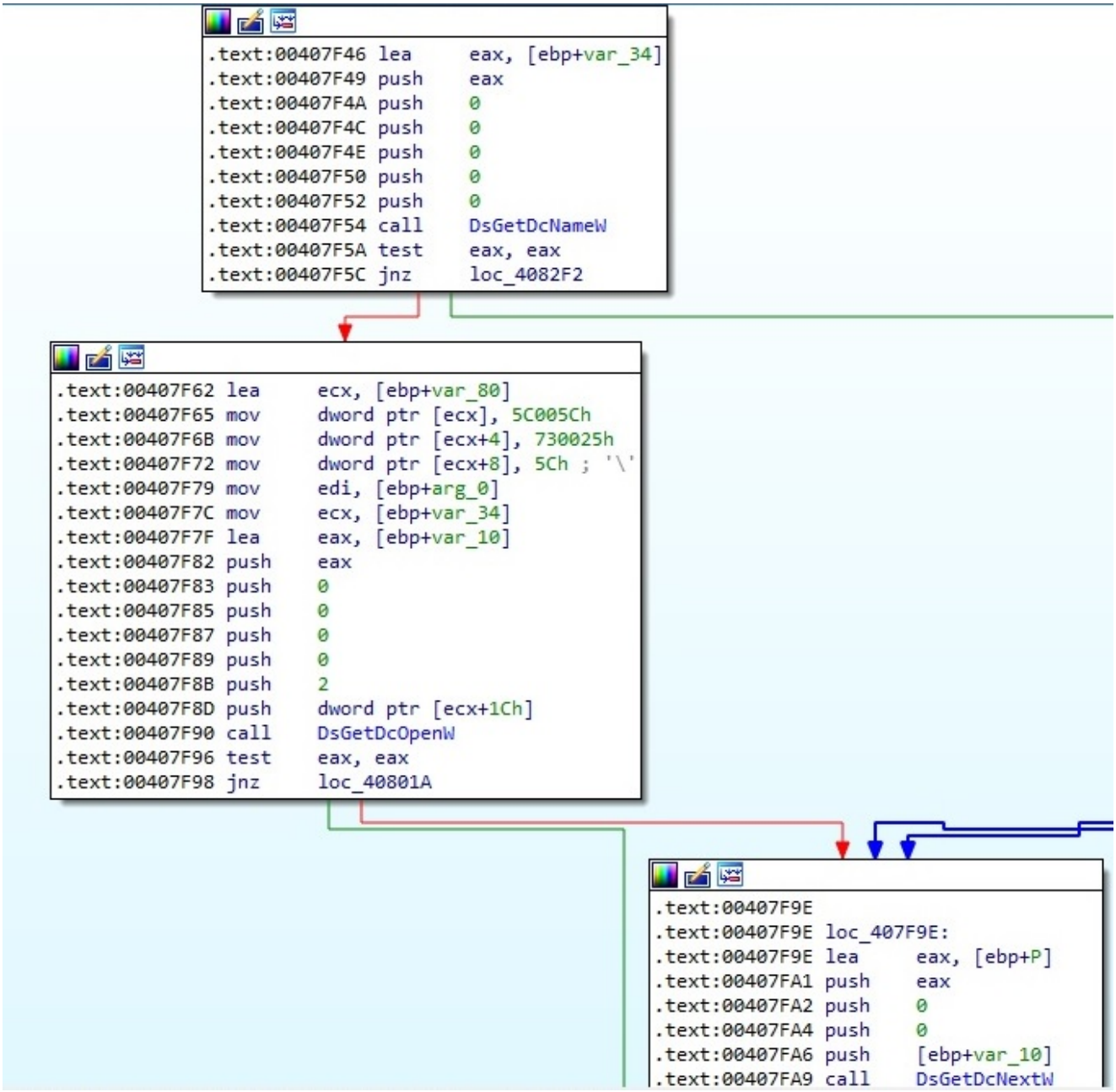
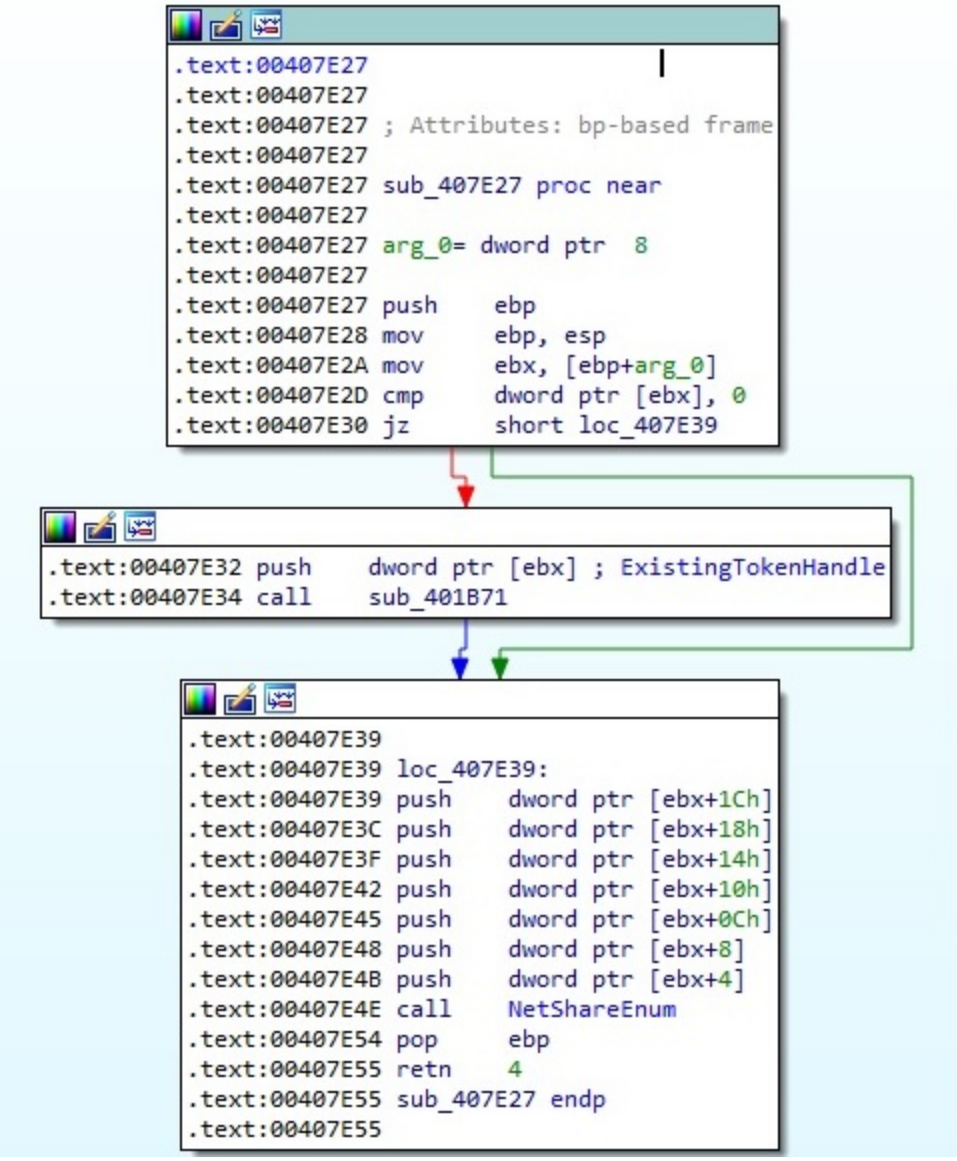


Figure 143

Darkside enumerates all network shares using the NetShareEnum API and encrypts each one of them by the main encryption routine described so far:



```
.text:00407E27
.text:00407E27
.text:00407E27 ; Attributes: bp-based frame
.text:00407E27
.text:00407E27 sub_407E27 proc near
.text:00407E27
.text:00407E27 arg_0= dword ptr  8
.text:00407E27
.text:00407E27 push    ebp
.text:00407E28 mov     ebp, esp
.text:00407E2A mov     ebx, [ebp+arg_0]
.text:00407E2D cmp     dword ptr [ebx], 0
.text:00407E30 jz      short loc_407E39
```

```
.text:00407E32 push    dword ptr [ebx] ; ExistingTokenHandle
.text:00407E34 call    sub_401B71
```

```
.text:00407E39
.text:00407E39 loc_407E39:
.text:00407E39 push    dword ptr [ebx+1Ch]
.text:00407E3C push    dword ptr [ebx+18h]
.text:00407E3F push    dword ptr [ebx+14h]
.text:00407E42 push    dword ptr [ebx+10h]
.text:00407E45 push    dword ptr [ebx+0Ch]
.text:00407E48 push    dword ptr [ebx+8]
.text:00407E4B push    dword ptr [ebx+4]
.text:00407E4E call    NetShareEnum
.text:00407E54 pop     ebp
.text:00407E55 retn    4
.text:00407E55 sub_407E27 endp
.text:00407E55
```

Figure 144

Thread activity – sub_4096A4

The following JSON is decrypted by the thread:

Address	Hex	ASCII
026C79A0	7B 0D 0A 22 69 64 22 3A 22 25 73 22 2C 0D 0A 22	{.."id":"%s",.."
026C79B0	75 69 64 22 3A 22 25 73 22 2C 0D 0A 22 65 6E 63	uid":"%s",.."enc
026C79C0	2D 6E 75 6D 22 3A 22 25 75 22 2C 0D 0A 22 65 6E	-num":"%u",.."en
026C79D0	63 2D 73 69 7A 65 22 3A 22 25 73 22 2C 0D 0A 22	c-size":"%s",.."
026C79E0	73 68 69 70 2D 6E 75 6D 22 3A 22 25 75 22 2C 0D	skip-num":"%u",.
026C79F0	0A 22 65 6C 61 70 73 65 64 2D 74 69 6D 65 22 3A	."elapsed-time":
026C7A00	22 25 75 2E 25 75 22 0D 0A 7D 00 AB AB AB AB AB	"%u.%u"..}.««««««

Figure 145

The file opens the following registry key by calling RegCreateKeyExW:

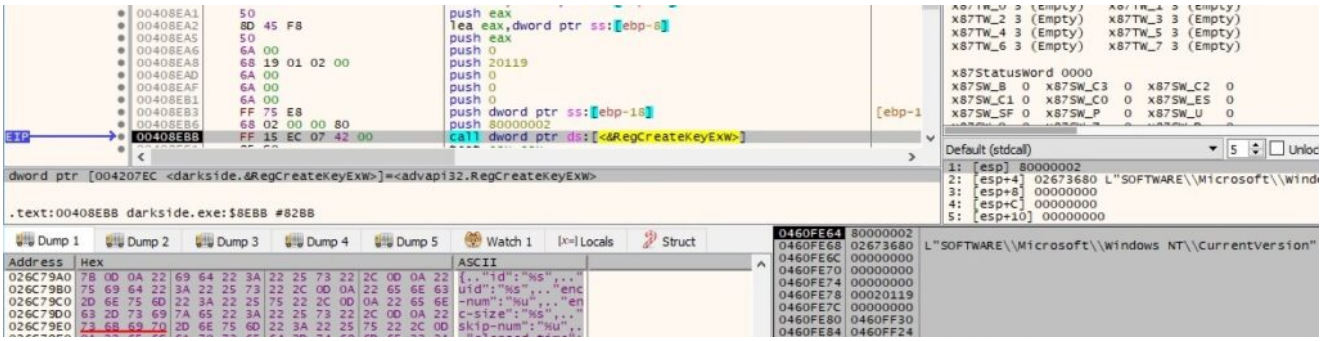


Figure 146

The Product ID is retrieved again by calling the RegQueryValueExW function:

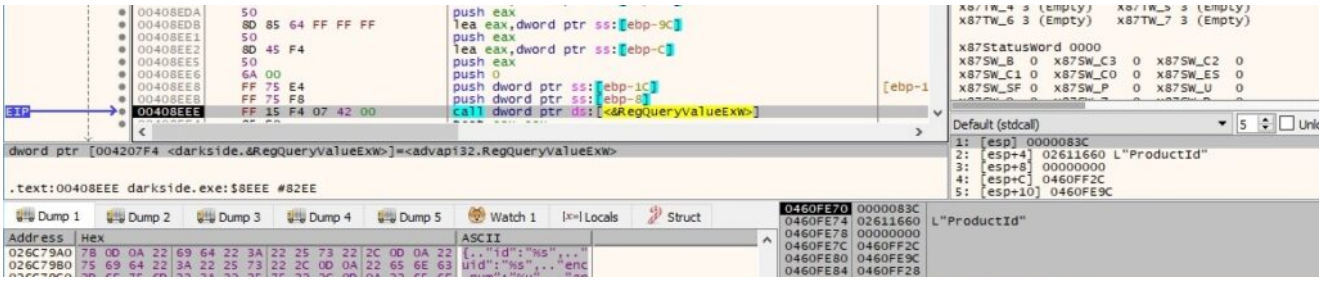


Figure 147

The machine GUID is extracted from the registry and represents a unique identifier for the machine:

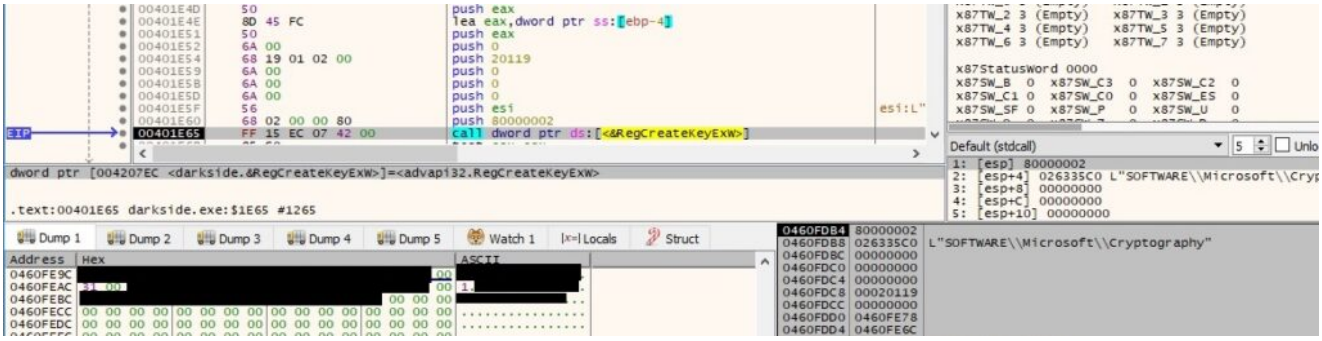


Figure 148

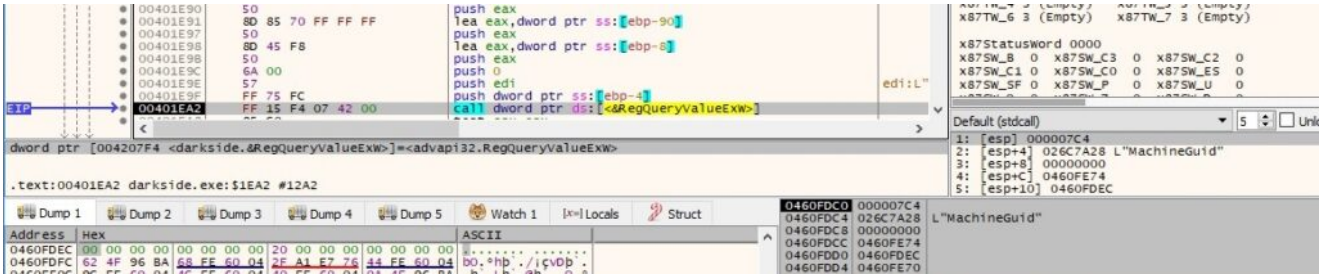


Figure 149

After the encryption finishes, the malware sends encryption statistics to the C2 server, such as: victim ID, uid, number of encrypted files, size of encrypted files, number of skipped files and elapsed time. The final JSON structure looks like the following:

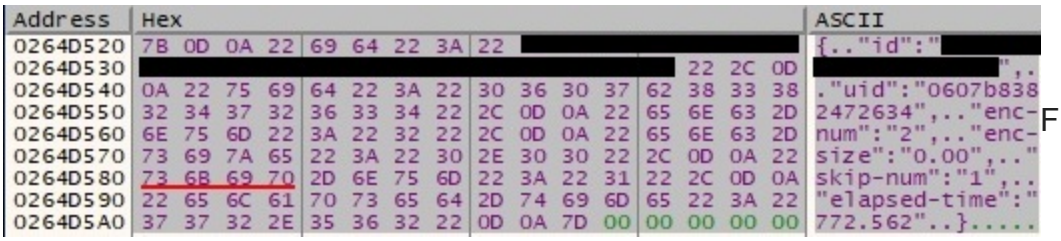


Figure 150

As already described so far regarding the C2 communication, the buffer is encrypted with a custom algorithm and base64-encoded. The request sent to the C2 server is presented in the next picture:

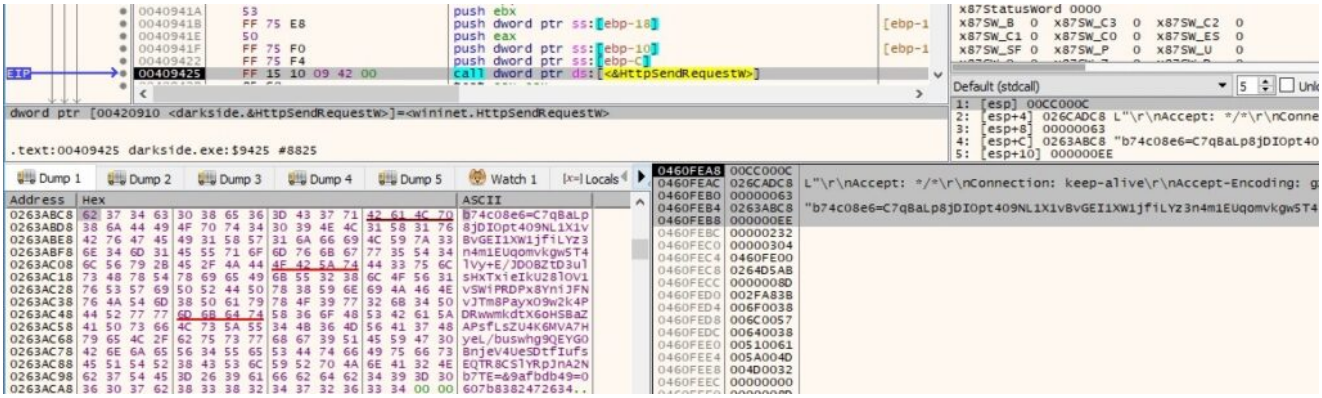


Figure 151

If the self deletion feature would be enabled, Darkside would delete itself using ShellExecuteW:

Figure 152

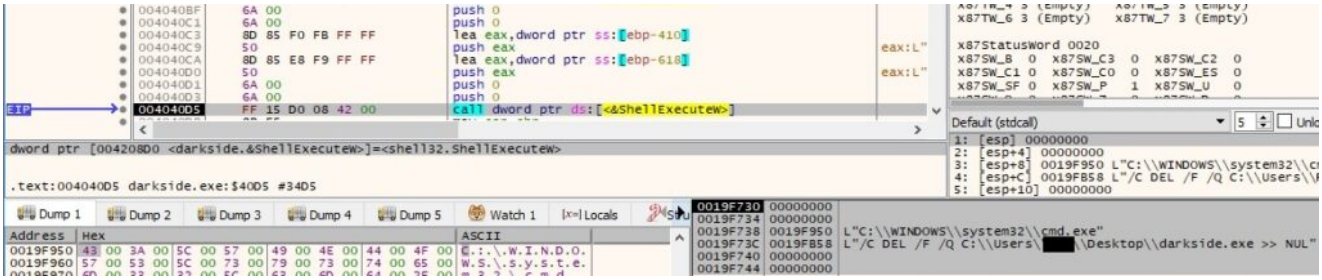


Figure 153

As we specified at the beginning of the analysis, the binary can run with different parameters:

- 1 parameter: filename – only this file will be encrypted
- 2 parameters: “-path” directory – only this directory will be encrypted
- 3 parameters: “-work” worker0 job0-<Process Id> – this is spawned by the initial process, already described

A particular case is handled by the ransomware differently when it deals with a shortcut file (.lnk file). Basically, the binary wants to extract the full path to the file from this link. It calls the CoCreateInstance API with the CLSID of {000214F9-0000-0000-C000-000000000046} (IShellLinkW interface):

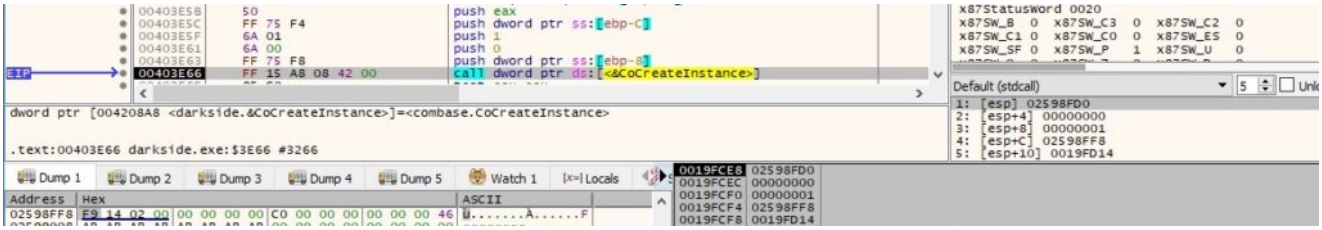


Figure 154

Unfortunately, Scylla didn't help us here and it couldn't provide us the methods. We've found that the next 2 function calls are used to extract the path of the file/directory:

Figure 155

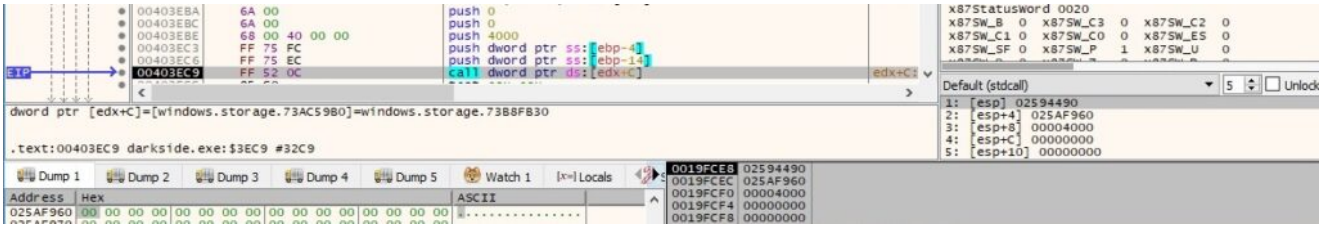
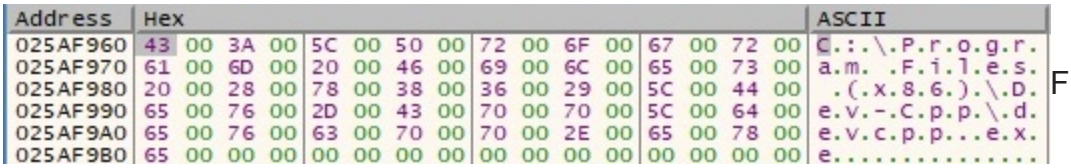


Figure 156

The file extracted above is encrypted as usual:



Address	Hex	ASCII
025AF960	43 00 3A 00 5C 00 50 00 72 00 6F 00 67 00 72 00	C.:.\.P.r.o.g.r.
025AF970	61 00 6D 00 20 00 46 00 69 00 6C 00 65 00 73 00	a.m. .F.i.l.e.s.
025AF980	20 00 28 00 78 00 38 00 36 00 29 00 5C 00 44 00	.(.x.8.6.).\.D.
025AF990	65 00 76 00 2D 00 43 00 70 00 70 00 5C 00 64 00	e.v.-.C.p.p.\.d.
025AF9A0	65 00 76 00 63 00 70 00 70 00 2E 00 65 00 78 00	e.v.c.p.p...e.x.
025AF9B0	65 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	e...............

Figure 157

References

MSDN: https://docs.microsoft.com/en-us/windows/win32/api/

Fakenet: https://github.com/fireeye/flare-fakenet-ng

Any.run: https://any.run/report/0a0c225f0e5ee941a79f2b7701f1285e4975a2859eb4d025d96d9e366e81abb9/e7a712f5-961a-45b4-a7e5-a0f7196113a5

VirusTotal: https://www.virustotal.com/gui/file/0a0c225f0e5ee941a79f2b7701f1285e4975a2859eb4d025d96d9e366e81abb9/detection

Analysis of Darkside Ransomware v1.8.6.2: https://chuongdong.com/reverse%20engineering/2021/05/06/DarksideRansomware/

Fireeye report: https://www.fireeye.com/blog/threat-research/2021/05/shining-a-light-on-darkside-ransomware-operations.html

https://gist.github.com/api0cradle/d4aaef39db0d845627d819b2b6b30512

https://forum.powerbasic.com/forum/user-to-user-discussions/source-code/25222-wmi-wrapper-functions

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wmi/3485541f-6950-4e6d-98cb-1ed4bb143441

INDICATORS OF COMPROMISE

C2 domains: baroquetees[.]com, rumahsia[.]com

SHA256: 0A0C225F0E5EE941A79F2B7701F1285E4975A2859EB4D025D96D9E366E81ABB9

Created files: README<RansomPseudoValue>.TXT, %PROGRAMDATA%\<RansomPseudoValue>.BMP, %PROGRAMDATA%\<RansomPseudoValue>.ico

Service Name: <RansomPseudoValue>, Service display name: <RansomPseudoValue>

Registry key: HKCR\<RansomPseudoValue>\DefaultIcon=%PROGRAMDATA%\<RansomPseudoValue>.ico

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:79.0) Gecko/20100101 Firefox/80.0 (prone to False Positives)