Retread Ransomware

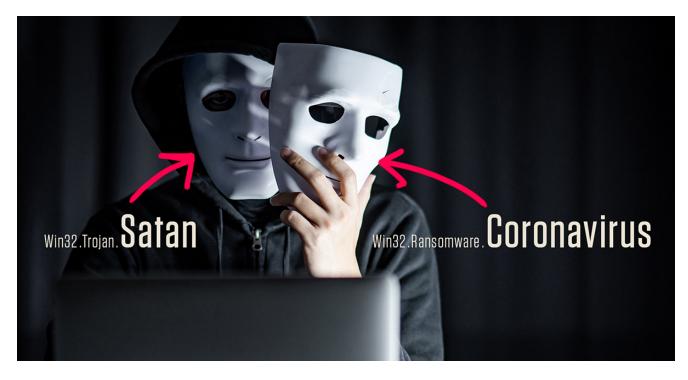
Blog.reversinglabs.com/blog/retread-ransomware

Threat Research | June 5, 2020



Blog Author

Robert Simmons, Independent malware researcher and threat researcher at ReversingLabs. <u>Read More...</u>



In March of 2020, MalwareHunterTeam <u>discovered a downloader</u> which installed both a KPot infostealer as well as a second payload which was a ransomware variant that used the string "CoronaVirus". This sample was leveraging ongoing current events and appears to be some form of cover for or distraction from the infostealer trojan that was installed alongside it. Via code analysis of this "CoronaVirus" sample, it is clear that it reuses a large amount of code from a four year old sample of ransomware detected as "Satana".

This older malware sample was first seen on June 29, 2016. This old sample and the recent one share two decoding algorithms that are used to hide strings, code, and a little, tiny PE file. This embedded file is run via a modification to the <u>BootExecute registry key</u>. The new file has some changes to the strings that are encoded along with the encoded data hiding the ransom note and the Bitcoin wallet addresses used to collect payment.

These are new addresses with a number of payments made during the same time that the new file was observed in the wild. The ransom amount demanded is quite small: \$50. All in all, this is a strange executable. What follows is an examination of the two decoding algorithms and a method for discovering the old samples based on YARA rules that focus on these algorithms.

String Decoding

Many malware families obfuscate the strings that are used during their execution as a way to foil static analysis and detections based on the <u>decoded forms of the strings</u>. In the samples examined here, the string encoding is very simple. It is a substitution cipher which shifts one character then subtracts the position index of the character in the encoded string to reveal the decoded character. This algorithm is highlighted in Figure 1.

		loc_401873:	
\rightarrow	0x00401873 8B55FC	mov	edx, dword [ebp+var_4]
	0x00401876 C6040200	mov	byte [edx+eax], 0x0
	0x0040187a C60000	mov	byte [eax], 0x0
	0x0040187d 8A1406	mov	dl, byte [esi+eax]
	0x00401880 2AD1	sub	dl, cl
	0x00401882 FECA	dec	dl
	0x00401884 881407	mov	byte [edi+eax], dl
	0x00401887 41	inc	есх
	0x00401888 40	inc	eax
	0x00401889 3B4DF8	cmp	<pre>ecx, dword [ebp+var_8]</pre>
	0x0040188c 76E5	jbe	loc_401873

Figure 1: String Decoding Algorithm

By focusing on the algorithm itself, which can be more stable than surrounding code when reused, one may locate other malware samples that could be related to the sample being analyzed. Figure 2 shows the same algorithm, but in a debugger where the first encoded string that is operated on in the sample can be seen in the dump at the bottom.

	00401873 00401873 00401876 0040187A 0040187D 00401880	26F0 8855 FC C60402 00 C600 00 8A1406 2AD1	mov edx, dword ptr ss:[ebp-4] mov byte ptr ds:[edx+eax],0 mov byte ptr ds:[eax],0 mov d],byte ptr ds:[esi+eax] sub d],c]	
	00401882 00401884 00401887	FECA 881407 41	<pre>dec dl mov byte ptr ds;[edi+eax].dl inc ecx</pre>	
0 0 0	00401888 00401889	41 40 3B4D F8 76 E5 5F	inc ecx inc eax cmp ecx,dword ptr ss:[ebp-8] jbe coronavirus_0ff1a.40187 pop edi	
0 0 0	0040188F 00401890 00401892 00401893	5E 8BC 3 5B 8BE5	pop esi mov eax,ebx pop ebx mov esp,ebp	
	00401895 00401896	5D C 3	pop ebp ret	
edx=FFFFF72 eax=00778B24 .text:00401865	Ciphe coronavirus	rhex Off1a.exe:\$1865 #C6	Cipherter	ct
🛢 Dump 1 🔮	Dump 2	Dump 3 🚯 Dump 4	Dump 5 🛢 Watch 1 🍳 Locals	s 🖌
Address Hex 00409000 44 71 00409010 75 67 00409020 76 72	77 71 00 00		ASCIT 4 67 77 79 Dqussg]q{.~. [gwy A 78 48 6C ugwqJuXw]XH]	

Figure 2: First Encoded String in Sample

This same algorithm can be easily translated into Python and the strings from the sample decoded without executing the sample. A small Python program for decoding these strings is shown in Figure 3. The first string in the sample, which decodes to "CoronaVirus", is shown here.

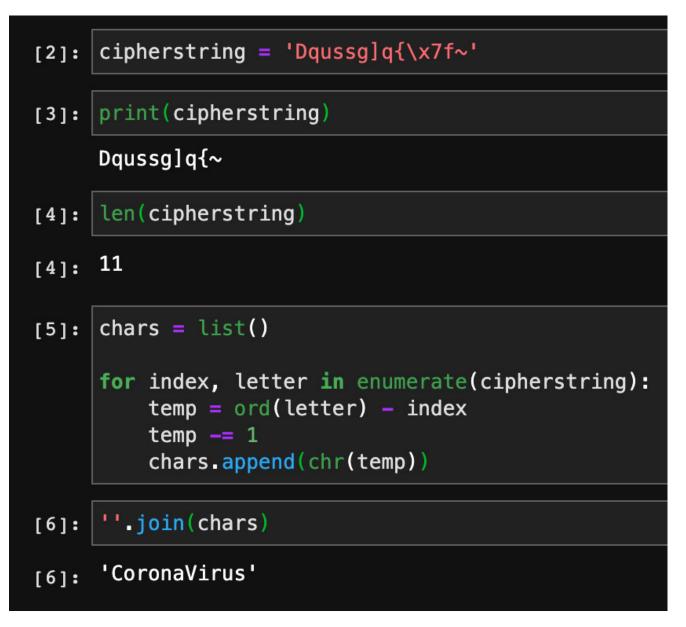


Figure 3: Python Implementation of String Decoder Algorithm

Taking only the bytes of the instructions that comprise this algorithm, a byte string for a YARA rule is identified.

```
$0p1 = { 8A 14 06 2A D1 FE CA 88 14 07 }
```

By then deploying this YARA rule as a retrohunt in the <u>Titanium Platform</u>, a number of additional samples are found. The results of this retrohunt are shown in Figure 4.

				File size		File	type					
					<1MB					PE/Exe		
-		Compl	05		<10MB				1	Unknov	vn	
2	210	フフ ^{Sampl}	es		<100MB				(other fo	ormats	
					<650MB							
					>=650MB							
Filter	ed by:				all shared	private	local	all	local	cloud	cloud	-retro
		Match Time	Threat	Name	Rule			<u>Format</u>	Fil	<u>es</u>	<u>Size</u>	
	•	2020-05-19 13:19 UTC	Win32.Trojan.Filecoder	b611ef99b6de4e12b66fc56708111f8f1	dfe9fCoronaviru	s_ROT_So	mething.	PE/Exe	1	. 1	116 KB	≡
	•	2020-05-19 13:19 UTC	Win32.Trojan.Satan	b38b4c1dcf6d6ecd1bbfc236b43c37c18	044c Coronaviru	s_ROT_So	mething.	PE/Exe	1	L	72 KB	≡
	•	2020-05-19 13:12 UTC	Win32.Trojan.Filecoder	a4be638c95a5f0f855285d8b43f53475	3db5bCoronaviru	s_ROT_So	mething.	PE/Exe	1	L 1	116 KB	≡
	•	2020-05-19 13:10 UTC	Win32.Ransomware.Cor.	.coronavirus_3299f.exe	Coronaviru	s_ROT_So	mething.	PE/Exe	1	[]	43 KB	≡
	•	2020-05-19 13:01 UTC	Win32.Trojan.Filecoder	7a4d98124c01a267aa47bc654a5ba46a	203b Coronaviru	s_ROT_So	mething.	PE/Exe	1	. 1	116 KB	≡

Figure 4: Retrohunt Results

Old Satana Sample

In the results of the retrohunt there are a number of samples from 2016. The sample with the earliest 'first seen' date is shown in Figure 5.

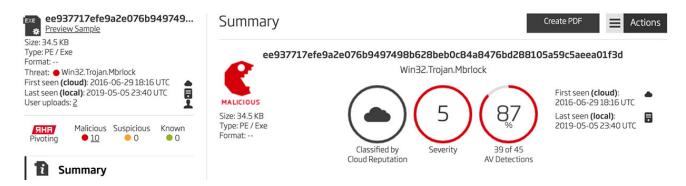


Figure 5: Earliest Related Satana Sample from 2016

This file is well detected as Satana via the <u>TitaniumCore</u> YARA classification as well as being classified as ransomware by the <u>TitaniumCore Machine Learning classification</u>. Both of these classifications are seen in Figure 6.



	•	Titar	iumCore	YARA
--	---	-------	---------	------

Туре	Engine Classification
Version	3.9.4.0
Result	Win32.Ransomware.Satana

• TitaniumCore Machine Learning

Туре	Engine Classification
Version	3.9.4.0
Result	Win32.Ransomware.Heuristic

Figure 6: Detection and Classification

In addition to this first sample, there are nine other samples that have been identified as related to the first based on the <u>ReversingLabs Hash Algorithm (RHA)</u>. These other samples are seen in Figure 7.

ee937717efe9a2e076b949749 Preview Sample	Malicious	\sim	All Local 1	FiCloud				
Size: 34.5 KB Type: PE / Exe Format:		<u>Time</u>	Threat	Name	<u>Format</u>	<u>Files</u>	<u>Size</u>	
Threat: • Win32.Trojan.Mbrlock First seen (cloud): 2016-06-2918:16 UTC •		2020-05-19 12:52 UTC	Win32.Trojan.Mbrlock	ee937717efe9a2e076b9497498b628beb0c84a8	PE/Exe	1	34.5 KB	≡
Last seen (local): 2019-05-05 23:40 UTC User uploads: 2		2020-05-19 12:52 UTC	Win32.Trojan.Satan	3faab8447d3cf878d9383dd064d37efc89a7b5a4	PE/Exe	1	34.5 KB	≡
SHE Malicious Suspicious Known Pivoting ● 10 ● 0 ● 0	1 1	2020-05-19 11:26 UTC	Win32.Trojan.Mbrlock	ea2c169529e782994be5296c81ff4668dba2b77a	. PE/Exe	1	34.5 KB	≡
i Summary		2020-05-19 12:52 UTC	Win32.Trojan.Mbrlock	307992fed12fa6b1bedbf277532140250f3c171e	PE/Exe	1	34.5 KB	≡
•		2020-05-19 12:52 UTC	Win32.Trojan.Satan	385371bfe457fa2e06e3b94963eff6428a355d93	PE/Exe	1	34.5 KB	≡
TitaniumCore		2020-05-19 12:52 UTC	Win32.Trojan.Mbrlock	23714f5a88fbf948040794004ed2cca0859a1952	PE/Exe	1	72 KB	≡
TitaniumCloud		2020-05-19 12:52 UTC	Win32.Trojan.Satana	00d44472aad92a037df1b280dad50ed25017af7a	PE/Exe	1	27 KB	≡
Extracted Files (0)		2020-05-19 12:52 UTC	Win32.Trojan.Satan	a0720d5d65adc0e6cb5e1e083a31c0e76cc3c1df	PE/Exe	2	36.5 KB	≡
File Visualization		2020-05-19 12:52 UTC	Win32.Trojan.Mbrlock	f15bcae0a5b0c7205aa2fa3414d171c02a876589	PE/Exe	1	72 KB	≡
Sources (2)		2020-05-19 13:19 UTC	Win32.Trojan.Satan	b38b4c1dcf6d6ecd1bbfc236b43c37c18044c2f42f.	. PE/Exe	1	72 KB	≡

Figure 7: Cluster of Ten Files Related via RHA

The earliest files in the cluster, each of which are also first seen in 2016, are very minor changes to the earliest file. These may be researchers testing or other types of analysis artifacts. One such minor change is highlighted in Figure 8 which shows a side-by-side

comparison of the two files using <u>HexFiend</u>.

	😑 ea2c	:169529e	782994b	e5296c8	1ff4668d	lba2b77a8	05bd0	57b53e59	952c65aa	af72 vs ee	937717e	fe9a2e07	6b9497
0	4D5A9000	03000000	04000000	FFFF0000	B8000000	00000000	0000	4D5A9000	03000000	04000000	FFFF0000	B8000000	00000000
24	40000000	00000000	00000000	00000000	00000000	00000000	0018	40000000	00000000	00000000	00000000	00000000	00000000
48	00000000	00000000	00000000	E8000000	ØE1FBAØE	00B409CD	0030	00000000	00000000	00000000	E8000000	ØE1FBAØE	00B409CD
72	21B8014C	CD215468	69732070	726F6772	616D2063	616E6E6F	0048	21B8014C	CD215468	69732070	726F6772	616D2063	616E6E6F
96	74206265	2072756E	20696E20	444F5320	6D6F6465	2E0D0D0A	0060	74206265	2072756E	20696E20	444F5320	6D6F6465	2E0D0D0A
120	24000000	00000000	78F9BD97	3C98D3C4	3C98D3C4	3C98D3C4	0078	24000000	00000000	78F9BD97	3C98D3C4	3C98D3C4	3C98D3C4
144	35E040C4	3098D3C4	5387D7C4	3E98D3C4	5387D9C4	3D98D3C4	0090	35E040C4	3098D3C4	5387D7C4	3E98D3C4	5387D9C4	3D98D3C4
168	BF84DDC4	3D98D3C4	3C98D2C4	4098D3C4	C6BBCAC4	3F98D3C4	00A8	BF84DDC4	3D98D3C4	3C98D2C4	4098D3C4	C6BBCAC4	3F98D3C4
192	3C98D3C4	3E98D3C4	27057CC4	3E98D3C4	27054EC4	3D98D3C4	0000	3C98D3C4	3E98D3C4	27057CC4	3E98D3C4	27054EC4	3D98D3C4
216	52696368	3C98D3C4	00000000	00000000	50450000	4C010400	00D8	52696368	3C98D3C4	00000000	00000000	50450000	4C010400
240	5E556E57	00000000	00000000	E0002201	0B010A00	00580000	00F0	5E556E57	00000000	00000000	E0002201	0B010A00	00580000
264	00280100	00000000	20540000	00100000	00700000	00004000	0108	00280100	00000000	20540000	00100000	00700000	00004000
288	00100000	00020000	05000100	00000000	05000100	00000000	0120	00100000	00020000	05000100	00000000	05000100	00000000
312	00B00100	00040000	4AEB0000	02000081	00001000	00100000	0138	00AA0100	00040000	4AEB0000	02000081	00001000	00100000
336	00001000	00100000	00000000	10000000	00000000	00000000	0150	00001000	00100000	00000000	10000000	00000000	00000000
1: Repl	ace 1 byte	at offset	0x139 wi	th 1 byte									

Figure 8: Side-by-side Comparison of Satana Samples

More recently observed files from this same cluster are clearly analysis artifacts from researchers or analysis tools. Many of them simply have the encoded strings decoded in the sample and no other changes. Knowing that all the files in the cluster are essentially the same file, only the earliest sample

(ee937717efe9a2e076b9497498b628beb0c84a8476bd288105a59c5aeea01f3d) is used for comparison with the CoronaVirus sample.

Data Decoding

Another common method used to encode data to avoid detection and identification is the "exclusive or" operation or <u>XOR</u>. The sample analyzed here uses this operation to hide large blobs of data. It specifically encodes the ransom note as well as a very small PE file which is used to display a variation of the ransom note on reboot. The location where this XOR operation is carried out is shown in Figure 9.

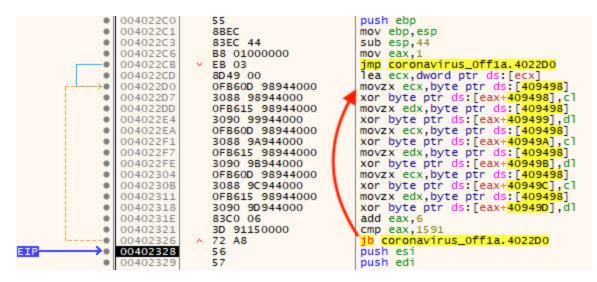


Figure 9: XOR Decoding Process

This is a loop which uses the first byte of the data as the key and then operates using that key on the rest of the bytes in the data. The encoded data before this operation has started is shown in Figure 10. The XOR key is the very first byte, 0x58.

Ump 1	1		Dur	np 2			Dum	р 3			Dump	94	ų	D	ump	5	💮 Watch 1	[x=] L
Address	He	¢							ni Natara								ASCII	
00409498	58	5F	58	58	64	59	58	58	6E	5D	58	58	67	5E	58	58	X_XXdYXXn]	XXg^XX
004094A8	OD	5A	58	58	02	5A	58	58	8C	5 B	58	58	79	58	58	58	.ZXX.ZXX.	XXYXXX
004094B8	76	-9	58	58	58	54	58	58	58	48	58	58	7B	5B	58	58	VYXXXTXXXH	XX{[XX
004094C8	7B	5 B	58	58	76		58	58	43	58	58	58	61	E9	58	3E	{[XXV\XXCX	xxaéx>
004094D8	38	A	E6	58	24	E7	58	58	5E	E1	58	5A	AB	FC	E3	74	8¤æX\$çXX^a	
004094E8	58	-	D9	B 3	58	24	D9	9B	58	58	5E	D1	81	D2	5 F	6A	X\$U*X\$U,XX	
004094F8	5E		58	5F	DO	5F	1B	D9	A3	A3	5F	58	2D	A9	A7	B9	^¤X_ĐÛ££	
00409508	2F	7D	01	5 B	58	58	95	48	BO	99	58	D1	86	58	BO	CB	/}a[XX.H°.	
00409518	58	BO	37	58			58	95	42	DB	99	59	D1	93	C8	58	X°7XiXX.BO	
00409528	95	42	6X	81	RÞ.	A:	F		58	58	B 3	14	E2	D8	58	EO	.Ba;°ZXX	
00409538	59	58	5A	E1	5F	58	E3	F 8	26	95	50	4B	E1	5A	59	42	YXZá_XãX&.	
00409548	E3	58	24	95	7A	4B	58	F8	DF	58	5A	5A	F8	58	26	88	āX\$.zKXøßX	
00409558	2D	A9	E1	59	58	6A	5 B	5A	41	58	74	54	EO	5D	58	7C	-@áYXj[ZAX	
00409568	5 B	77	EC	5 B	95	4B	58	3E	39	B2	58	24	58	58	A2	58	[W1[.KX>9*	
00409578	E8	D7	BE	28	E8	58	BE	29	58	E8	A6	BE	3C	AC	69	91	ex%(eX%)Xe	
00409588	0E	58	F4	64	58	2C	51	D9	A1	A7	58	5F	2C	5 B	19	B 3	.xôdx,QÙį§	×_,[.*
00409598	AA	06	EO	58	59	4B	69	8A	E3	57	58	D1	18	AD	EB	5D	axyki.äw	
004095A8	95	48	9B	58	22	E1	48	50	58	EO	50	58	12	48	95	4B	.H.X"áHPXa	
004095B8	58	D1	85	D2	1F	4A	64	26	2C	58	59	9B	EC	49	E8	58	XN.O.Jd&,X	
004095C8	E1	A7	D8	58	E2	58	58	EF	48	EB	58	9A	18	9B	E1	5 B	á§ØXâXXïHë	
004095D8	58	EO	5C	58	7D	78	58	95	4B	9B	3E	38	E9	-	CA	58	Xa\X}xX.K.	
004095E8	D1	88	8B	BO	7C	57	5C	68	58	64	61	2E	5A	5C	5F	BO	Nº W\hXd	
004095F8	50	58	58	D8	B1	5C	2B	B2	3E	39	59	D8	56	E3	53	58	PXXر\+=>9	
00409608	EC	56	95	48	59	D8	5D	9B	6B	ED	58	15	02	C8	58	5 B	iV.HYØ].ki	
00409618	58	58	58	DA	SC.	58	68	A7	A7	58	58	EO	58	60	75	59	XXXU\Xh§§X	XaX uY
00409628	58	18	5C	60	41	58	88	58	54	56	47	58	E2	56	58	EC	X.\ AX.XTV	
00409638	51	95	79	EO	58	59	14	95	79	0C	30	31	2B	58	78	28	Q.yaxyy.	
00409648	2A	37	3F	2A	39	35	58	78	3B	39	36	36	37		78	58	*7?*95XX;9	
00409658	3A	3D	78	2A	2D	36	78	31	58	36	78	10	17	OB	78	35	:=x*-6x1X6	
00409668	37	D8	3C	3D	76	55	55	52	7C	5C	DE	58	41	F1	84	98	70<=VUUR \	
00409678	05	90	EA	CB	49	5F	5B	EB	CB		58	5F	FF	B3	F3	5C	êEI_[ëE.	
00409688	CB	06	58	5F	1E	0D	45	CB	04	DD	5A	5F	77	5A	5F	0A	E.XEE.Y	Z_WZ

Figure 10: First Byte of Encoded Data is XOR Key

After this decoding process is complete, the embedded PE file begins to appear. This data is still compressed, but the file magic "MZ" and the DOS stub string are clear. The decoded version of the data shown above is seen in Figure 11.

Dump 1 Dump 2					8		Dum	р 3	🚛 Dump 4				💭 Dump 5				🥘 Watch 1 🛛 [x=] L
Address	He	ĸ															ASCII
00409498	58	07	00	00	3C	01	00	00	36	05	00	00	3F	06	00	00	X<6?
004094A8	55	02	00	00	5A	02	00	00	D4	03	00	00	21	00	00	00	UZÔ!
004094B8	2E	01	00	00	00	0C	00	00	00	10	00	00	23	03	00	00	#
004094C8	23	03	00	00	2E	04	00	00	1B	00	00	00	39	B1	00	66	#9±.f
004094D8	60	FC	BE	00	7C	BF	00	00	06	B9	00	02	F3	A4	BB	2C	`ü¾. ¿'Ó¤»,
004094E8	00	7C	81	EB	00	7C	81	C3	00	00	06	89	D9	8A	07	32	. .ë. .ÅÙ2
004094F8	06	FC	00	07	88	07	43	81	FB	FB	07	00	75	F1	FF	E1	.üC.ûûuñÿá
00409508	77	25	B 8	03	00	00	CD	10	E8	C1	00	89	DE	00	E8	93	w%1.èAÞ.è.
00409518	00	E8	6F	00	B4	00	00	CD	1A	83	C1	01	89	CB	90	00	.eo 1AE
00409528	CD	1A	39	D9	75	F9	E8	02	00	00	EB	4C	BA	80	00	B 8	1.9ÙuùèëL°
00409538	01	00	02	B9	07	00	BB	00	7E	CD	08	13	39	02	01	1A	'».~1'
00409548	BB	00	7C	CD	22	13	00	A0	87	00	02	02	A	00	7E	DO	». 1"
00409558	75	F1	B9	01	00	32	03	02	19	00	2C	0C	B8	05	00	24	uñ'\$
00409568	03	2F	B4	03	CD	13	00	66	61	EA	00	7C	00	10	FA	00	./ .1faê. ú.
00409578	BO	8F	E6	70	BO	00	E6	71	00	BO	FE	E6	64	F	31	C9	°.æp°.æq.°þædô1É
00409588	56	00	AC	3C	00	74	09	81	F9	FF	00	07	74	03		EB	V.¬<.tùÿt.Aë
00409598	F2	5E	B 8	00	01	13	31	D2	BB	OF	00	89	40	F5	13	05	0^10»@0".
004095A8	CD		C3	and the other distances in the local distance	7A	B9	10	08	00	B 8	08	00	4A	10	CL	13	1.Å.z'
004095B8	00	89		8A	47	12	3C	7E	74	00	01	C3	B4	11	BO	00	Ý.G. <~tÅ . °.
004095C8	B9	FF	80	00	BA	00	00	B7	10	B 3	00	C2		C3	B9	3	'ÿº
004095D8	00	_	04	00	25	20	00	CD	13	C3	66	60	B1	0C	92	00	% .1.Af`±
004095E8	89	DO		E8	24	OF	04	30	00	_	39	_	02	04	07	E.	.DÓè\$0.<9vè
004095F8	08	00	00	80	E9	04	73	EA	66	61	01	80	0E	BB	OB	00	
00409608	B4	OE	CD	10	01	80	05	C3	33	B5	00	4D	5A	90	00	03	.1A3µ.MZ
00409618	00	00	00	82	04	00	30	FF	FF	00	00	B 8	00	38	2D	01	·····8-,
00409628	00	40	04	38	19		D0	00	0C	OE	1F	00	BA	0E	00	Β4	.@.8D°
00409638	09	CD	21	B8	00	01	4C	CD	21	54	68	69	73	00	20	70	.1!L1!This. p
00409648	72	6F	67	72	61	GD	00	20	63	61	6E	6E	6F	74	20	00	rogram. cannot .
00409658	62	65	20	72	75	6E	20	69	00	6E	20	44	4F	53	20	GD	be run i n DOS m
00409668	6F	80	64	65	2E	OD	OD	0A	24	04	86	00	19	A9	DC	CO	o.de\$@UA
00409678	5D	C8	B2	93	11	07	03	B 3	93	56	00	07	A7	EB	AB	04	
00409688	93	5E	00	07	46	55	1D	93	5C	85	02	07	2F	02	07	52	.^FU\/R

Figure 11: Decoded Data with PE File Starting to Appear

The next step after decoding the data is to decompress it. The instructions that perform this action utilize the library function <u>"RtlDecompressBuffer"</u>. The function name is loaded dynamically by decoding a string using the string decoding capability examined earlier. As puzzling as it sounds, something like a comic book supervillain telling you their next step before doing something evil, the meaning of these instructions is stated clearly by the additional string, "DeCompressor", which is not encoded or hidden in any way. These instructions as seen in the debugger are shown in Figure 12.

• • •	00401900 00401901 00401903 00401904 00401908 00401912	55 8BEC 51 833D 345F4100 00 C745 FC 00000000 V 75 2D	<pre>push ebp mov ebp,esp push ecx cmp dword ptr ds:[<&RtlDecompressBuffer>],0 mov dword ptr ss:[ebp-4],0 ine coronavirus Offla.401941</pre>	Decode
•	00401914 00401919 0040191E 0040191E	B8 6C914000 E8 F2FEFFFF 50 A1 48834100	<pre>mov eax,coronavirus_Off1a.40916C call <coronavirus_off1a.decode_string> push eax mov eax,dword ptr ds:[418348]</coronavirus_off1a.decode_string></pre>	RtlDecompressBuffer
•	00401924 00401925 00401928	50 FF15 F0604000 A3 345F4100 85C0	<pre>push eax call dword ptr ds:[<&GetProcAddress>] mov dword ptr ds:[<&RtlDecompressBuffer>],eax</pre>	
	00401930 00401932 00401934 00401939	75 0D 68 D4614000 E8 A2FEFFFF	test eax,eax jne coronavirus_Off1a.401941 push coronavirus_Off1a.4061D4 call <coronavirus_off1a.super_nop></coronavirus_off1a.super_nop>	4061D4:"DeCompressor"
	0040193E 00401941 00401942 00401943	83C4 04 57 56 E8_98FEFFFF	add esp,4 push edi push esi call <coronavirus_offla.super_nop></coronavirus_offla.super_nop>	1
• • •	00401948 0040194B 0040194E 00401951	8855 08 8845 0C 83C4 08 8D4D FC	<pre>mov edx,dword ptr ss:[ebp+8] mov eax,dword ptr ss:[ebp+C] add esp,8 lea ecx,dword ptr ss:[ebp-4]</pre>	Why?
	00401954 00401955 00401956 00401957	51 57 52 56	push ecx push edi push edx oush esi	
•	00401958 00401959 00401958	50 6A 02 FF15 345F4100	push eax push 2 [call dword ptr ds:[k&RtlDecompressBuffer>]]	Execute
0	00401961 00401963	× 79 03	test eax,eax jns coronavirus_0ff1a.401968	

Figure 12: Disassembled Decompression Instructions

Aside from the tiny PE file and the ransom note template, the decoded list of Bitcoin wallet addresses is provided. One of these addresses is selected to be used with the ransom note template to generate the note that is dropped by the ransomware. These decoded and decompressed Bitcoin wallet addresses are shown in Figure 13.

Dump 1			Dump 2			🚛 Dump 3			1	🚛 Dump 4			🚛 Dump 5			5	👹 Watch 1	[x=] L
Address	He	ĸ														- 23	ASCII	
000358A8	62	63	31	71	74	36	79	70	7A	66	76	32	35	68	77	76	bc1qt6ypzfv	25 hwv
000358B8	37	30	7A	73	38	6E	76	72	64	68	6A	6C	39	36	32	6A	70zs8nvrdhj	1962j
000358C8	7A	79	79	6D	6B	6C	37	79	39	64	OD	0A	62	63	31	71	zyymk17y9d.	.bc1q
000358D8	7A	77	77	30	6B	6A	74	65	75	35	32	77	35	30	6D	33	zww0kjteu52	w5 Om3
000358E8	64	32	75	63	70	37	32	7A	68	35	64	77	63	68	70	68	d2ucp72zh5d	wchph
000358F8	39	76	71	6A	36	6B	OD	0A	62	63	31	71	33	76	36	66	9vqj6kbc1	q3v6f
00035908	61	72	38	35	67	74	64	73	72	6B	34	7A	75	34	66	75	ar85gtdsrk4	zu4fu
00035918	68	70	68	68	65	79	71	70	72	77	6D	75	76	36	32	6E	hphheyqprwm	uv62n
00035928	39	32	OD	0A	62	63	31	71	39	64	64	35	6E	6B	71	72	92bc1q9dd	15 nkqr

Figure 13: Decoded and Decompressed Bitcoin Wallet Addresses

These addresses are listed in Table 1. The list of wallet addresses in the sample have two duplicates, so the total set of addresses has 22 members each of which is of type <u>Bech32</u> <u>SegregatedWitness address format</u>.

bc1qc9axh3fq2ypgcd92j582v9khfrn52strql7ztn

bc1qzww0kjteu52w50m3d2ucp72zh5dwchph9vqj6k

bc1q5e8pwyk9rqtq400agngmq5h23cuz42x0wlqw3q

bc1qrkp9cx6svxguxupx9p0z5ss4nmyr4fwhvgkasg

bc1qjl0ufmwct84ww69zwyxe99gext7za6qkyhx200

bc1q6ryyex33jxgr946u3jyre66uey07e2xy3v2cah

bc1qlmu9xk8wdnydnlcvy9uvcepzklcv7kxyhk8ymy

bc1qftwqsaw57v6cstwrdvclmkz63plvf5q3vqvw4k

bc1qegps92ddvgv8t45lfcn02afsjlyf7mynuqvpmm

bc1 quwc6 yqgdcgm6 z2663 vpjm9 cgtfwf7 mhk2 n7 gtn

bc1q9dd5nkqrxsny93r9u09jwq8agvkf04afxh67jg

bc1qt6ypzfv25hwv70zs8nvrdhjl962jzyymkl7y9d

bc1qgd3nj0486k35ra42a550ntyafdr7s5lmyzjn29

bc1q8r42fm7kwg68dts3w70qah79n5emt5m76rus5u

bc1q3v6far85gtdsrk4zu4fuhphheyqprwmuv62n92

bc1qpvguajy4rxr7743hzuwfmz32krzzfcjl9rf0qx

bc1qe9gj2sj3an73dq37vpe34xflc83yjp4u3pzfgz

bc1qpaksevt2w6cqdeqjvm8dapvz66y3hs3jjy4x66

bc1qt3uf3wx569z5z9wdeanuj4rwq7m06grhxt96v3

bc1qzv6h2zaedjgduc6xmyn86hdsu0skuunt9lhkxn

bc1q2x9h8wlh2cuxjd9rv94v6syz7lpxxk2wwrndmv

bc1qxsjfw0jftfr9n5urdyh7xmz4cspls8qefuyetr

Table 1: Bitcoin Wallet Addresses

The ransom note template has four locations where string replacement is used to generate the final note text that is dropped to the ransom note file. These locations in the template are highlighted in Figure 14.

CORONAVIRUS is there All your file are crypted. Your computer is temporarily blocked on several levels. Applying strong military secret encryption algorithm. To assist in decrypting your files, you must do the following: 1. Pay %s btc to Bitcoin wallet %s or purchase the receipt Bitcoin; 2. Contact us by e-mail: %s and tell us this your unique ID: %s and send the link to Bitcoin transaction generated or Bitcoin check number. After all this, you get in your email the following: 1. Instructions and software to unlock your computer 2. Program - decryptor of your files. Donations to the US presidential elections are accepted around the clock. Desine sperare qui hic intras! [Wait to payment timeout 25 - 40 min]

Figure 14: Ransom Note Template

The email address, coronaVi2022[@]protonmail[.]ch, along with the amount demanded in Bitcoin (BTC), 0.008, are both hard coded. The unique ID is generated by concatenating the drive serial number collected from PhysicalDrive0 and the HwProfileGuid and then generating an MD5 from this string. The Bitcoin wallet address is chosen from the list decoded and decompressed as documented above. These four values just before the call to sprintf can be seen in Figure 15.



Figure 15: String Replacement into Ransom Note Template

The payment demand of 0.008 BTC, interestingly, does not line up with any of the seven payments made to the set of wallet addresses from the binary. Each is a single payment. Some of the payments are larger than the demand and some are smaller than the demand. It is unknown whether these payments are in fact connected with the ransomware.

Old Satana Sample Differences

The majority of the instructions that handle the generation of randomness and moving that random data around in memory are all identical between the Satana and the Coronavirus samples. Library and compiler added code along with these identical functions are shown in Figure 16 as analyzed using Relyze's binary difference feature.

Everything	Equal Modifie	d Removed	Added		
Difference 🔻	It	tem Type	Diff Type	Item A	Item B
0.00%	Fu	inction	Equal	_allrem1	_allrem1
0.00%	Fu	inction	Equal	_seh_longjmp_unwind	_seh_longjmp_unwind
0.00%	Fu	inction	Equal	anonymous_library_func1	unwind_handler
0.00%	Fu	inction	Equal	func_0x405B60	func_0x405D8C
0.00%	Fu	inction	Equal	func_0x402794	func_0x4019E4
0.00%	Fu	inction	Equal	memsetl	memset1
0.00%	Fu	inction	Equal	matherr	matherr
0.00%	Fu	inction	Equal	memcpy1	memcpy1
0.00%	Fu	inction	Equal	func_0x401E80	func_0x401170
0.00%	Fu	inction	Equal	func_0x402290	func_0x401580
0.00%	Fu	inction	Equal	func_0x402060	func_0x401350
0.00%	Fu	inction	Equal	anonymous_library_func	abnormal_termination
0.00%	Fu	inction	Equal	func_0x401DB0	func_0x401000
0.00%	Fu	inction	Equal	_chkstk1	_chkstk1
0.00%	Fu	inction	Equal	func_0x402EA0	func_0x401FA0
0.00%	Fu	inction	Equal	func_0x402BF0	func_0x401E40
0.00%	Fu	inction	Equal	func_0x402740	func_0x401990
0.00%	Fu	inction	Equal	func_0x402780	func_0x4019D0
0.00%	Fu	inction	Equal	func_0x405C38	func_0x405E64
0.00%	Function		Equal	RtlUnwind1	RtlUnwind1
0.00%	Fu	inction	Equal	func_0x405C7A	func_0x405EA6

Figure 16: Binary Difference Showing Identical Functions

There are a number of functions that have some amount of change according to binary difference, however, most of these instruction changes do not have a significant effect on the malware sample's behavior. The strings that are operated on are different and one significant location of change is the BTC demand amount which was 0.5 BTC in the older sample and 0.008 BTC in the newest. This change in price in the disassembly is shown in Figure 17.

<pre>mov byte ptr [edx+ex], cl inc eax inc eax isst cts cl, cl inz book byte ptr [edx+ex], cl inc eax isst cts cl, cl inz book byte ptr [edx+ex], cl inz bo</pre>	<pre>code_0x402300: mov Al, byte ntr [ecx] mov bitp thr [eck], al inc ecx inc.edx test al, al jur code_0x402830 call func_0x402720 ; inline voidcdecl(void) mov ecx, duord ptr [dsta_0x419424] pub vs_ptr dsta_0x419523 pub vs_ptr dsta_0x419523 pub vs_ptr dsta_0x419523 pub vs_ptr dsta_0x419523 pub vs_ptr dsta_0x419523</pre>	
<pre>push ws_ptr string_08 [P457] push ws_ptr data_041214 push ws_ptr data_0412178 push ws_ptr data_0412178 push ws_ptr data_041218 push ws_ptr data_041218 push ws_ptr data_041218 push ws_ptr data_041218 push ws_ptr data_041218 push ext sev dword ptr [data_04184A0], eax call dword ptr [data_04184A0] mov ext, dwx add emp; [j] les data, [data]]</pre>	<pre>puble vs_ptr_string_0000</pre>	

Figure 17: Difference in Hard-Coded BTC Demand

Conclusion

As can be seen through comparison between the older Satana sample and the new Coronavirus sample, these two are very closely related. This old sample, or the source code for it, has been repurposed and redeployed as "CoronaVirus" ransomware. However, due to the distribution alongside the KPot sample, the very low demand amount of approximately \$70, and the payments that are oddly larger or smaller than the demand: this may be a faux ransomware campaign. Totaling all the payments received, the set of wallet addresses only collected 0.10417322 BTC or about \$930, a strangely low amount.

YARA Rule

```
private rule WindowsPE
 {
      condition:
          uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550
  }
  rule SubstCipher_MinusIndex
  {
      meta:
          author = "Malware Utkonos"
          date = "2020-04-13"
          exemplar =
"ee937717efe9a2e076b9497498b628beb0c84a8476bd288105a59c5aeea01f3d"
      strings:
          $0p1 = { 8A 14 06 2A D1 FE CA 88 14 07 }
      condition:
          WindowsPE and all of them
  }
```

MORE BLOG ARTICLES