# Third time (un)lucky – improved Petya is out

blog.malwarebytes.com/threat-analysis/2016/07/third-time-unlucky-improved-petya-is-out/

#### Malwarebytes Labs

July 18, 2016

5CE0 49 53 20 50 4C 55 4	47 47	45	44	0D	0A	20	20	49	4E	IS PLUGGED IN
05CF0 uu\$\$\$\$\$\$\$\$\$û0 0A 00 :	20 20							20		! CHKDSK i
05Duu\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$uu0 61 (	69 72		6E	67	20		65	63		s repairing sect
u\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$ u\$\$\$\$\$\$\$\$\$				20	72		62			orPlease reboo
u\$			6D				65	72	21	t your computer!
1u\$\$\$\$\$\$*00 *\$\$\$*20 *\$\$\$\$\$u63 '	72 79			69			20			Decrypting se
1*\$\$\$\$* 63 U\$U6F 72\$\$\$\$*00 (				20		6F		20	62	ctor You b
\$\$\$u u\$u u\$\$\$ \$\$\$u u\$\$\$u u\$\$\$					6D	20			20	ecame victim of
*\$\$\$\$uu\$\$\$ \$\$\$uu\$\$\$*										the PETYA RANSOM
05D*\$\$\$\$\$\$*11 *\$\$\$\$\$\$*1 OD (	00 A0		0A	20				20		WARE! The h
05D804\$\$\$\$\$\$u\$\$\$\$\$\$u 69 73 (		20	6F	66	20				72	arddisks of your
u\$*\$*\$*\$*\$*\$u \$\$u\$ \$ \$ \$u\$\$ uuu	74 65	72	20	68	61			20	62	computer have b
5DA0 \$\$\$\$\$u\$u\$u\$\$\$ 65 u\$\$						64	20		69	een encrypted wi
3000B0 *\$\$\$\$\$\$\$*61 600\$\$\$\$					61	72		20		th an military g
\$\$\$\$\$uu ***** uuuu\$\$\$\$\$\$			63	72				69	6F	rade encryptio
*\$\$\$\$\$\$\$\$\$\$uuu uu\$\$\$\$\$\$\$\$***\$ **\$\$\$\$\$\$\$\$\$uu **\$***	\$ <b>\$</b> *69	74		6D	2E	20	54			n algorithm. The
15Duuuu **\$\$\$\$\$\$\$\$		20		61		20			20	re is no way to
iu\$\$\$\$\$\$\$\$\$uu **\$\$\$\$\$\$\$\$\$\$	\$5 20		6F		72	20		61		restore your dat
*******			20	61	20				63	a without a spec Sect
\$\$\$*     \$\$\$\$ \$\$*    \$\$\$\$			2E	20		6F		20		ial key. You c
15E20 61 6E 20 70 75 72**	63 68	61	73	65	20	74	68	69	73	an purchase this

So far we dedicated several articles to the interesting, low-level ransomware called <u>Petya</u>, hijacking the boot sector. You can read about it here:

- <u>https://blog.malwarebytes.com/threat-analysis/2016/05/petya-and-mischa-ransomware-duet-p1/</u> Green Petya (version 2)
- <u>https://blog.malwarebytes.com/threat-analysis/2016/04/petya-ransomware/</u> Red Petya (version 1)

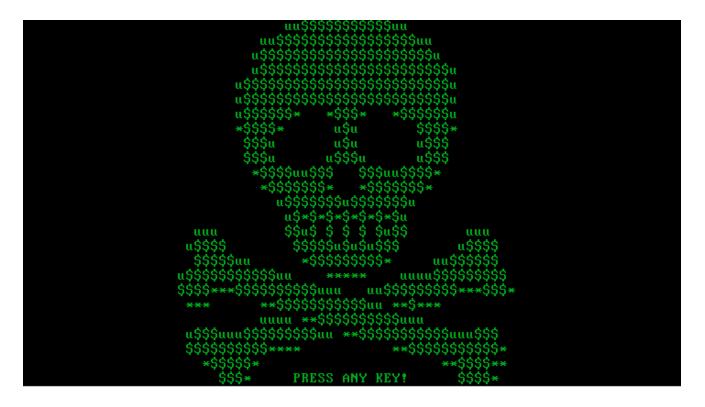
Each of those versions was using Salsa20 algorithm to encrypt Master File Table and make disk inaccessible. However, due to the implementation bugs the intended algorithm was weakened – giving a chance to recover data.

Unfortunately, as always in such cases, it is just a matter of time when cybercriminals get their cryptography fixed. Petya's authors got it right at the third attempt. The currently launched wave of this <u>ransomware</u> finally seems to have the proper Salsa20.

#### sample: <u>c8623aaa00f82b941122edef3b1852e3</u>

### **Behavioral analysis**

Behavior of Petya didn't changed – we can see exactly the same UI like in the <u>previous</u> green edition:



### Inside

Let's take a look at differences in the code. Using BinDiff we can spot, that not many functions have changed. However, those that were giving weak points to the previous edition are modified.

0.95	0.99	G 0000844E	check_key	0000844E	check_key
0.66	0.95	-IE 00009822	s20_littleendian	0000984E	sub_984E_74
0.65	0.90	-IE 00008AAC	reboot_disk	00008ADA	sub_8ADA_51

### Salsa20

First of all, let's take a look the function **s20\_littleendian** that was causing the major bug in the last release. Due to it's invalid implementation, only 8 out of 16 characters of the key were meaningful and brutforcing the key was easier (working solution has been implemented by <u>procrash</u>). Detailed explanation of this bug you can find in the updated <u>post about the previous Petya</u> – under the section "New Petya, new bug".

On the left – you can see the implementation of the buggy function (from the previous edition). On the right – current, fixed implementation:

00009822	s20 lit	tleendia	in		
00009822	push	b2 bp			// s20 littleendian
00009823	mov	b2 bp	b2	sp	
00009825	push	b2 si			
00009826	mov	b2 si,	b2	ss:[si+arg_0]	
00009829	sub	bl al,	bl	al	
0000982B	mov	bl ah,	ы	ds:[sp+1]	
0000982E	mov	bl cl	bl	ds:[si]	
00009830	sub	bl ch	ь1	ch	
00009832	add	b2 ax,	b2	cx	
00009834	cwd				
00009835	pop leave	b2 si			

0000984E s	sub_984E					
0000984E 1	oush	b2	bp			
0000984F	nov	b2	bp,	b2	sp	
00009851	oush	b2	si			
00009852	nov	b2	si,	b2	ss:[si+arg_0]	
00009855	sub	b1	ah,	b1	ah	
00009857 :	nov	bl	al,	bl	ds:[sp+2]	
0000985A	shl	b2	ax,	ь1	0x10	
00009850	ewd					
0000985E :	nov	b2	CX,	b2	ax	
00009860	nov	b1	ah,	b1	ds:[sp+1]	
00009863	sub	b1	al,	b1	al	
00009865 1	nov	b2	bx,	b2	dx	
00009867	ewd					
00009868	add	b2	ax,	b2	CX	
0000986A a	adc	b2	dx,	b2	bx	
0000986C I	nov	b2	CX,	b2	ax	
0000986E I	nov	b1	ah,	b1	ds:[sp+3]	
00009871	shl	b1	ah,	b1	0x10	
00009874	sub	b1	al,	b1	al	
00009876	nov	b2	bx,	b2	dx	
00009878	ewd					
00009879	add	b2	ax,	b2	cx	
0000987B	adc	b2	dx,	b2	bx	
00009870	nov	b1	cl,	b1	ds:[si]	
0000987F	sub	b1	ch,	b1	ch	
00009881 a	add	b2	ax,	b2	CX	
00009883	adc	b2	dx,	b1	0	
00009886	oop	b2	si			
00009887	leave					
00009888	retn					

sub 984E 0000984E

Secondary

#### Explanation

00009822 s20\_littleendian

The old implementation was truncated – it didn't used 32 bit values as it should – only added a sign bit expansion to the 16 bit value:

Now, authors got the proper implementation, using 32 bits. So, the last bug in Salsa20 got finally fixed, making implementation complete.

#### Key

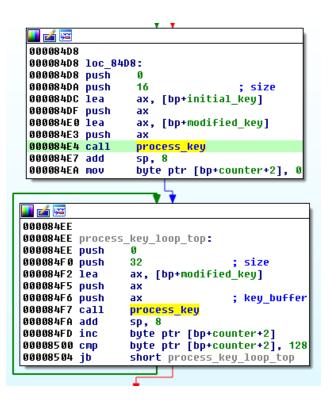
In the first (red) version of Petya authors used 32 byte long Salsa key – that was, however, generated from the 16 byte long key, using a custom function to pre-process it and extend.

In the second – green edition, they gave up this idea and applied the original 16 byte long key, without any modification.

This time, they changed mind and went back to the first solution of using 32 byte long key, yet with some improvements. Again we can see **expand32** in the code (instead of **expand16** known from the previous edition):

00009936 enter	16h, 0
0000993A push	di
00009938 push	si
0000993C mov	[bp+var_11], 78h ; 'x'
00009940 mov	[bp+var_10], 70h ; 'p'
00009944 mov	[ <b>bp+var_F], 61h</b> ; 'a'
00009948 <mark>mov</mark>	<pre>[bp+var_E], 6Eh ; 'n'</pre>
0000994C mov	[ <b>bp+var_D], 64h</b> ; 'd'
00009950 mov	[bp+var_B], 33h ; '3'_
00009954 mov	[bp+var_A], 32h ; '2'
00009958 mov	[bp+var_9], 2Dh ; '-'
0000995C mov	[ <b>bp+var_8], 62h</b> ; 'b'
00009960 mov	[ <b>bp+var_7], 79h</b> ; 'y'
00009964 <mark>mov</mark>	[bp+var_6], 74h ; 't'
00009968 <mark>mov</mark>	al, 65h ; 'e'
0000996A mov	[bp+var_12], al
0000996D mov	[bp+var_5], al
00009970 mov	al, 20h ; ' '
00009972 mov	[bp+var_C], al
00009975 mov	[bp+var_4], al
00009978 mov	[bp+var_3], 6Bh ; 'k'
0000997C xor	di, di

When the victim insert the key for the verification, before using it as a Salsa20 key, it is preprocessed by a new algorithm (more complex than in case of Red Petya):



### Conclusion

New edition shows that the project is reaching maturity – however, as we can read on the associated onion page – it is still a beta version and we can expect that it will keep evolving. Below – fragment of Petya's RaaS website:

## EASY ADMINISTRATION

Administrative Tasks like viewing the latest infections, setting the ransom price or recrypting your binary can be done with an clean and simple web-interface.

We also have an qualified support, which will help you with any problems. Since this project is still in beta, we are open for any bug-report or feature-request.

We are not yet sure about the distribution method, but probability is high, that also this time it is spam with a link leading to cloud storage. We strongly advise to be extra vigilant for the job applications coming this days – it proven to be a common cover for Petya/Mischa dropper. More information about it you can find in our previous articles about Petya.

# Appendix

Petya and Mischa – Ransomware Duet (Part 1)

This video cannot be displayed because your *Functional Cookies* are currently disabled. To enable them, please visit our *privacy policy* and search for the Cookies section. Select *"Click Here"* to open the Privacy Preference Center and select *"Functional Cookies"* in the menu. You can switch the tab back to *"Active"* or disable by moving the tab to *"Inactive."* Click *"Save Settings."* 

This was a guest post written by Hasherezade, an independent researcher and programmer with a strong interest in InfoSec. She loves going in details about malware and sharing threat information with the community. Check her out on Twitter @<u>hasherezade</u> and her personal blog: <u>https://hshrzd.wordpress.com</u>.