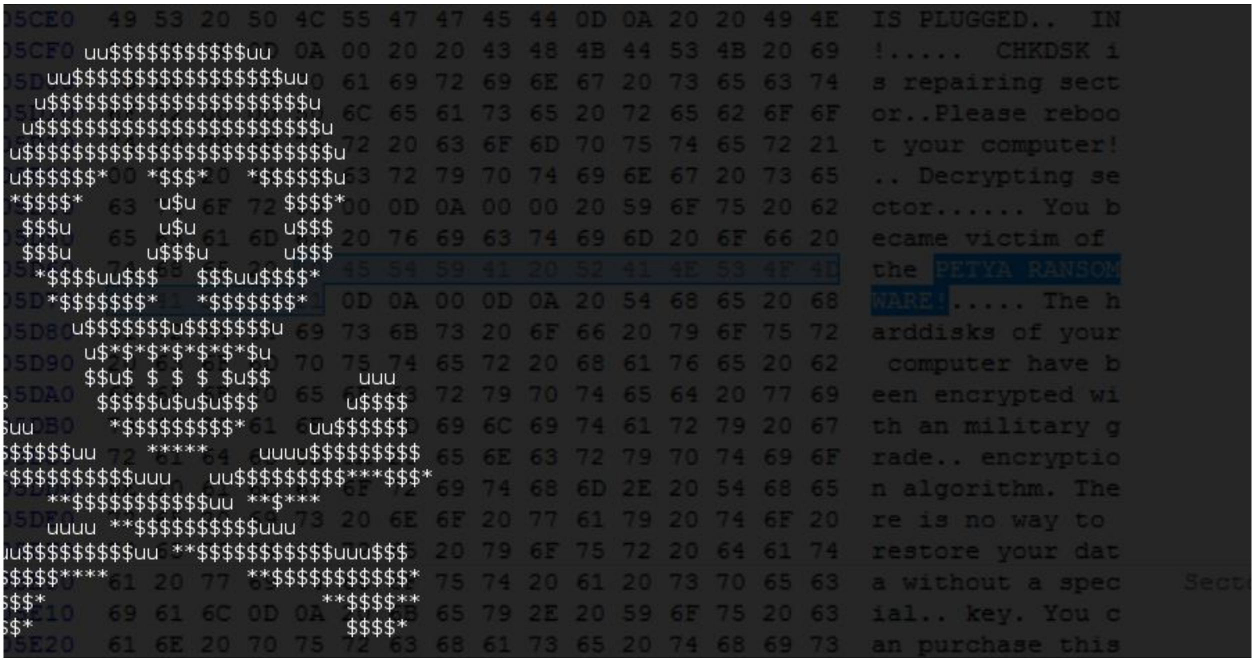


```
IS PLUGGED..  IN
!.....  CHKDSK i
s repairing sect
or..Please reboo
t your computer!
.. Decrypting se
ctor...... You b
ecame victim of
the PETYA RANSOM
WARE!..... The h
arddisks of your
 computer have b
een encrypted wi
th an military g
rade.. encryptio
n algorithm. The
re is no way to
restore your dat
a without a spec
ial.. key. You c
an purchase this
```

Third time (un)lucky – improved Petya is out

July 18, 2016 by Malwarebytes Labs

Last updated: September 8, 2016

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

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

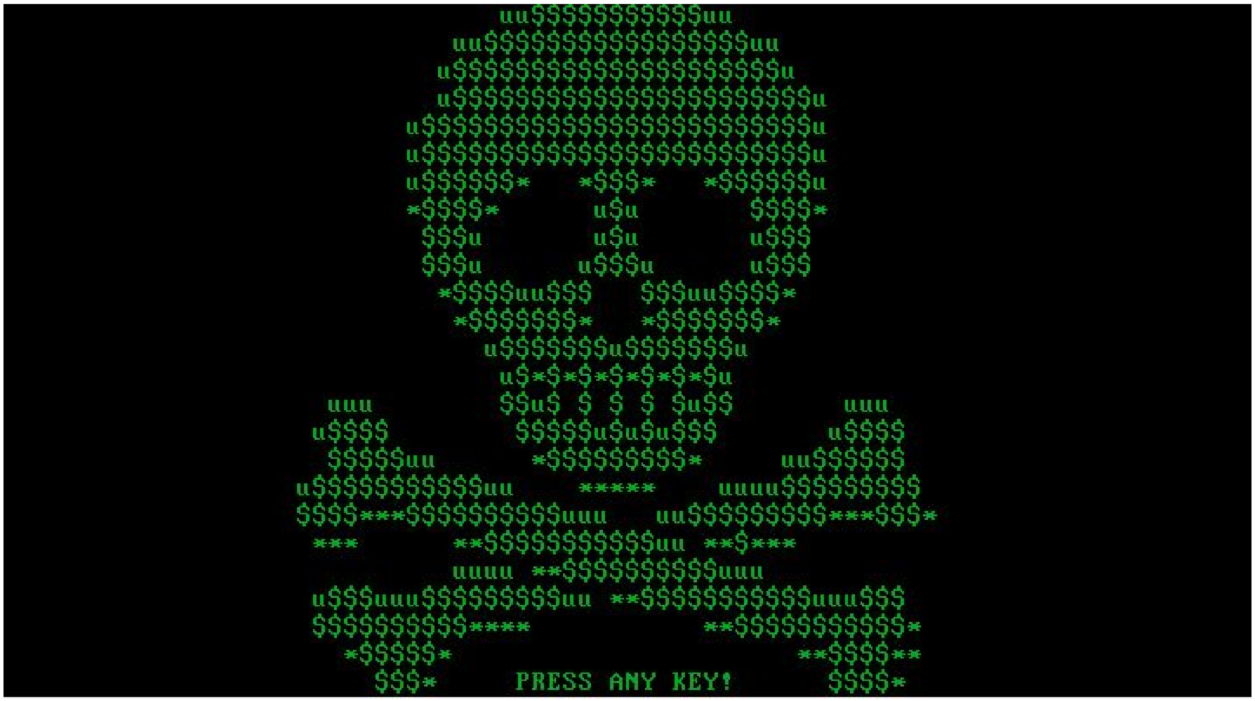
Each of those versions was using Salsa20 algorithm to encrypt Master File Table and make disk inaccessible. However, due to the implementation bugs the intended algorithm was weakened – giving a chance to recover data.

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 ransomware finally seems to have the proper Salsa20.

sample: c8623aaa00f82b941122edef3b1852e3

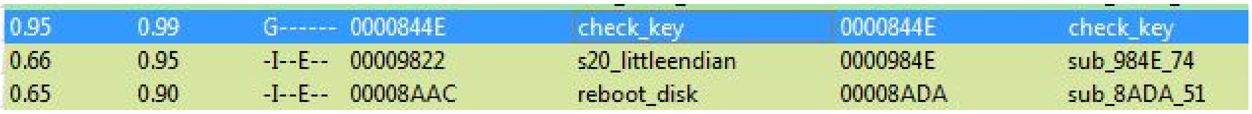
Behavioral analysis

Behavior of Petya didn’t changed – we can see exactly the same UI like in the previous 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.

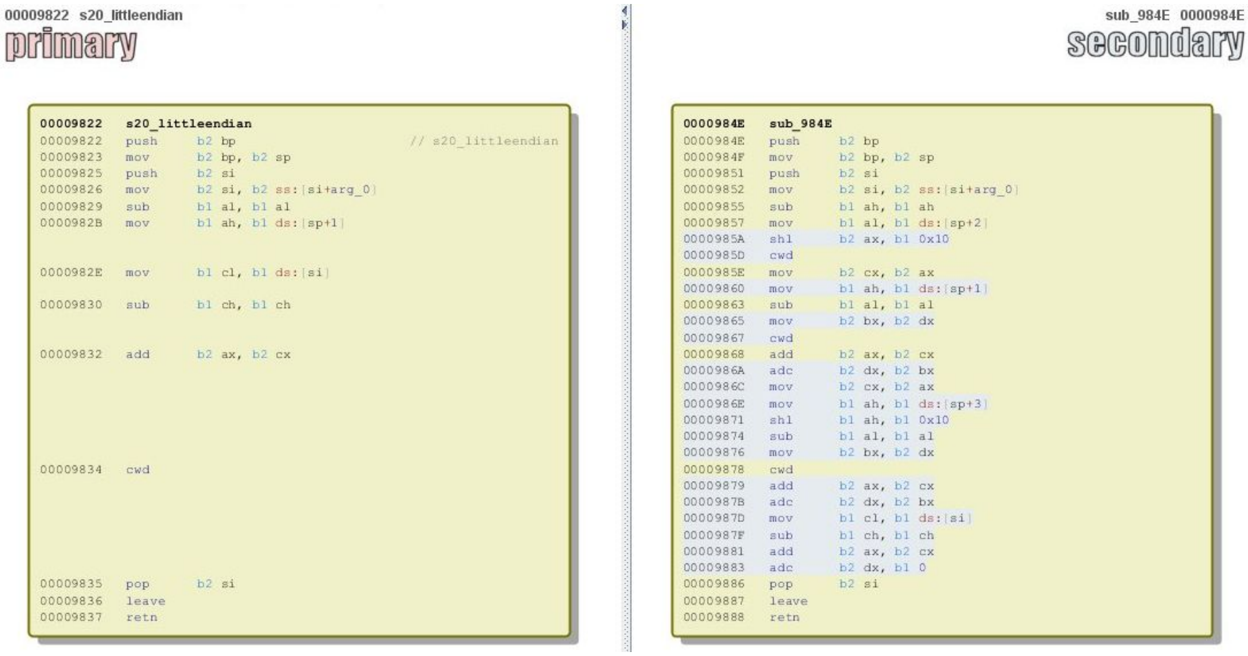


0.95	0.99	G------	0000844E	check_key	0000844E	check_key
0.66	0.95	-I--E--	00009822	s20_littleendian	0000984E	sub_984E_74
0.65	0.90	-I--E--	00008AAC	reboot_disk	00008ADA	sub_8ADA_51

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 procrash). Detailed explanation of this bug you can find in the updated post about the previous Petya – 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_littleendian
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     b1 al, b1 al
0000982B  mov     b1 ah, b1 ds:[sp+1]


0000982E  mov     b1 cl, b1 ds:[si]

00009830  sub     b1 ch, b1 ch


00009832  add     b2 ax, b2 cx






00009834  cwd






00009835  pop     b2 si
00009836  leave
00009837  retn
```

```
0000984E  sub_984E
0000984E  push    b2 bp
0000984F  mov     b2 bp, b2 sp
00009851  push    b2 si
00009852  mov     b2 si, b2 ss:[si+arg_0]
00009855  sub     b1 ah, b1 ah
00009857  mov     b1 al, b1 ds:[sp+2]
0000985A  shl     b2 ax, b1 0x10
0000985D  cwd
0000985E  mov     b2 cx, b2 ax
00009860  mov     b1 ah, b1 ds:[sp+1]
00009863  sub     b1 al, b1 al
00009865  mov     b2 bx, b2 dx
00009867  cwd
00009868  add     b2 ax, b2 cx
0000986A  adc     b2 dx, b2 bx
0000986C  mov     b2 cx, b2 ax
0000986E  mov     b1 ah, b1 ds:[sp+3]
00009871  shl     b1 ah, b1 0x10
00009874  sub     b1 al, b1 al
00009876  mov     b2 bx, b2 dx
00009878  cwd
00009879  add     b2 ax, b2 cx
0000987B  adc     b2 dx, b2 bx
0000987D  mov     b1 cl, b1 ds:[si]
0000987F  sub     b1 ch, b1 ch
00009881  add     b2 ax, b2 cx
00009883  adc     b2 dx, b1 0
00009886  pop     b2 si
00009887  leave
00009888  retn
```

Explanation

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:

```
static int16_t s20_littleendian(uint8_t *b)
{
  return b[0] +
         (b[1] << 8);
}
```

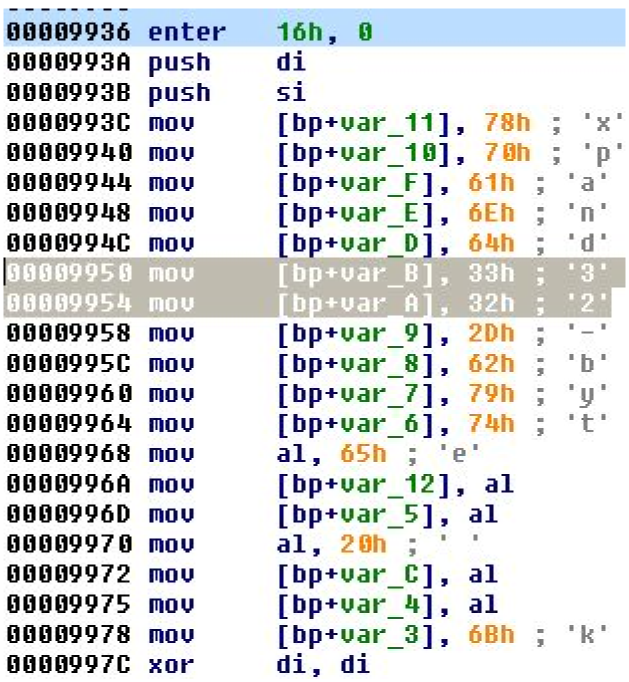
Now, authors got the proper implementation, using 32 bits. So, the last bug in Salsa20 got finally fixed, making implementation complete.

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.

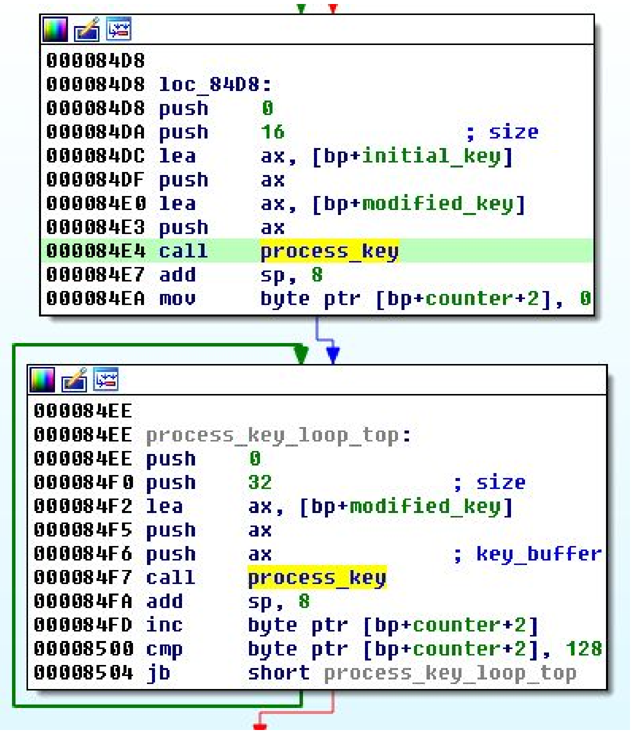
In the second – green edition, they gave up this idea and applied the original 16 byte long key, without any modification.

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
0000993B push     si
0000993C mov      [bp+var_11], 78h ; 'x'
00009940 mov      [bp+var_10], 70h ; 'p'
00009944 mov      [bp+var_F], 61h ; 'a'
00009948 mov      [bp+var_E], 6Eh ; 'n'
0000994C mov      [bp+var_D], 64h ; 'd'
00009950 mov      [bp+var_B], 33h ; '3'
00009954 mov      [bp+var_A], 32h ; '2'
00009958 mov      [bp+var_9], 2Dh ; '-'
0000995C mov      [bp+var_8], 62h ; 'b'
00009960 mov      [bp+var_7], 79h ; 'y'
00009964 mov      [bp+var_6], 74h ; 't'
00009968 mov      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):

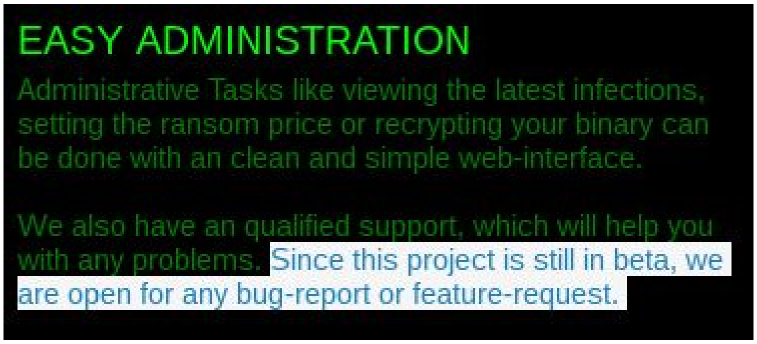


```
000084D8
000084D8 loc_84D8:
000084D8 push    0
000084DA push    16              ; size
000084DC lea     ax, [bp+initial_key]
000084DF push    ax
000084E0 lea     ax, [bp+modified_key]
000084E3 push    ax
000084E4 call    process_key
000084E7 add     sp, 8
000084EA mov     byte ptr [bp+counter+2], 0
```

```
000084EE
000084EE process_key_loop_top:
000084EE push    0
000084F0 push    32              ; size
000084F2 lea     ax, [bp+modified_key]
000084F5 push    ax
000084F6 push    ax              ; key_buffer
000084F7 call    process_key
000084FA add     sp, 8
000084FD inc     byte ptr [bp+counter+2]
00008500 cmp     byte ptr [bp+counter+2], 128
00008504 jb      short process_key_loop_top
```

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

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 @hasherezade and her personal blog: https://hshrzd.wordpress.com.