

Petya – Taking Ransomware To The Low Level

blog.malwarebytes.com/threat-analysis/2016/04/petya-ransomware/

Malwarebytes Labs

April 1, 2016

Petya is different from the other popular ransomware these days. Instead of encrypting files one by one, it denies access to the full system by attacking low-level structures on the disk. This ransomware's authors have not only created their own boot loader but also a tiny kernel, which is 32 sectors long.

Petya's dropper writes the malicious code at the beginning of the disk. The affected system's master boot record (MBR) is overwritten by the custom boot loader that loads a tiny malicious kernel. Then, this kernel proceeds with further encryption. Petya's ransom note states that it encrypts the full disk, but this is not true. Instead, it encrypts the master file table (MFT) so that the file system is not readable.

[UPDATE] READ ABOUT THE LATEST VERSION: GOLDENEYE

PREVENTION TIP: Petya is most dangerous in Stage 2 of the infection, which starts when the affected system is being rebooted after the BSOD caused by the dropper. In order to prevent your computer from going automatically to this stage, turn off *automatic restart after a system failure* ([see how to do this](#)).

If you detect Petya in Stage 1, your data can still be recovered. More information about it is [[here](#)] and in this article.

UPDATE: 8-th April 2016 Petya at Stage 2 has been cracked by leo-stone. Read more: <https://petya-pay-no-ransom.herokuapp.com/> and <https://github.com/leo-stone/hack-petya>. Tutorial helping in disk recovery is [here](#).

Analyzed samples

Main executable from another campaign (PDF icon)

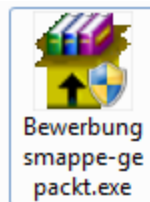
[a92f13f3a1b3b39833d3cc336301b713](#)

Special thanks to: [Florian Roth](#) – for sharing the samples, [Petr Beneš](#) – for a constructive [discussion](#) on [Twitter](#).

Behavioral analysis

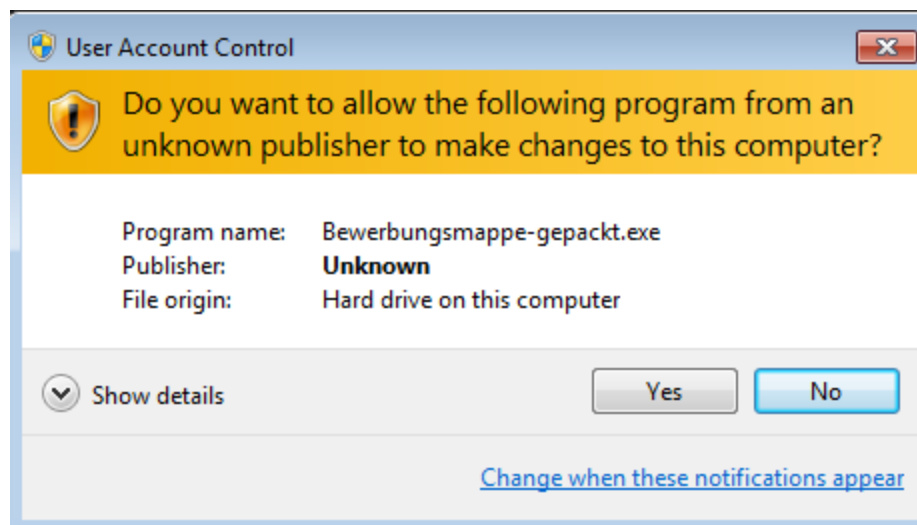
This ransomware is delivered via scam emails themed as a job application. E-mail comes with a Dropbox link, where the malicious ZIP is hosted. This initial ZIP contains two elements:

- a photo of a young man, purporting to be an applicant (in fact it is a publicly available stock image)
- an executable, pretending to be a CV in a self-extracting archive or in PDF (in fact it is a malicious dropper in the form of a 32bit PE file):



In order to execute its harmful features, it needs to run with Administrator privileges. However, it doesn't even try to deploy any user account control (UAC) bypass technique. It relies fully on social engineering.

When we try to run it, UAC pops up this alert:



After deploying the application, the system crashes. When it restarts, we see the following screen, which is an imitation of a CHKDSK scan:

```
Repairing file system on C:

The type of the file system is NTFS.
One of your disks contains errors and needs to be repaired. This process
may take several hours to complete. It is strongly recommended to let it
complete.

WARNING: DO NOT TURN OFF YOUR PC! IF YOU ABORT THIS PROCESS, YOU COULD
DESTROY ALL OF YOUR DATA! PLEASE ENSURE THAT YOUR POWER CABLE IS PLUGGED
IN!

CHKDSK is repairing sector 53754 of 132096 (40%)
```

In reality, the malicious kernel is already encrypting. When it finishes, the affected user encounters this blinking screen with an ASCII art:



Pressing a key leads to the main screen with the ransom note and all information necessary to reach the Web panel and proceed with the payment:

```
You became victim of the PETYA RANSOMWARE!

The harddisks of your computer have been encrypted with an military grade
encryption algorithm. There is no way to restore your data without a special
key. You can purchase this key on the darknet page shown in step 2.

To purchase your key and restore your data, please follow these three easy
steps:

1. Download the Tor Browser at "https://www.torproject.org/". If you need
help, please google for "access onion page".
2. Visit one of the following pages with the Tor Browser:

    http://petya37h5tbhyvki.onion/Mvnhqz
    http://petya5koahtsf7sv.onion/Mvnhqz

3. Enter your personal decryption code there:

    afMf5Z-C83M2q-Nv9uR1-g9GZXY-a4iU47-c5R4iT-xR1WZk-nX4HmW-rnc1Kg-HMekdy-
    W8WDRr-rXz6TZ-jo69HJ-pre5Ry-Myg9rt

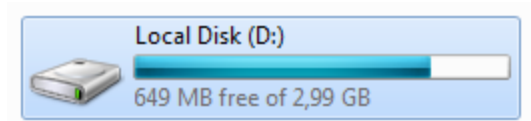
If you already purchased your key, please enter it below.

Key: _
```

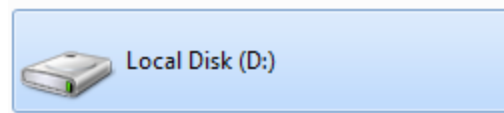
Infection stages

This ransomware have **two infection stages**.

The first is executed by the dropper (Windows executable file). It overwrites the beginning of the disk (including MBR) and makes an XOR encrypted backup of the original data. This stage ends with an intentional execution of **BSOD**. Saving data at this point is relatively easy, because only the beginning of the attacked disk is overwritten. The file system is not destroyed, and we can still mount this disk and use its content. That's why, if you suspect that you have this ransomware, the first thing we recommend is to not reboot the system. Instead, make a disk dump. Eventually you can, at this stage, mount this disk to another operating system and make the file backup. *See also: [Petya key decoder](#).*



The second stage is executed by the fake CHKDSK scan. After this, the file system is destroyed and cannot be read.

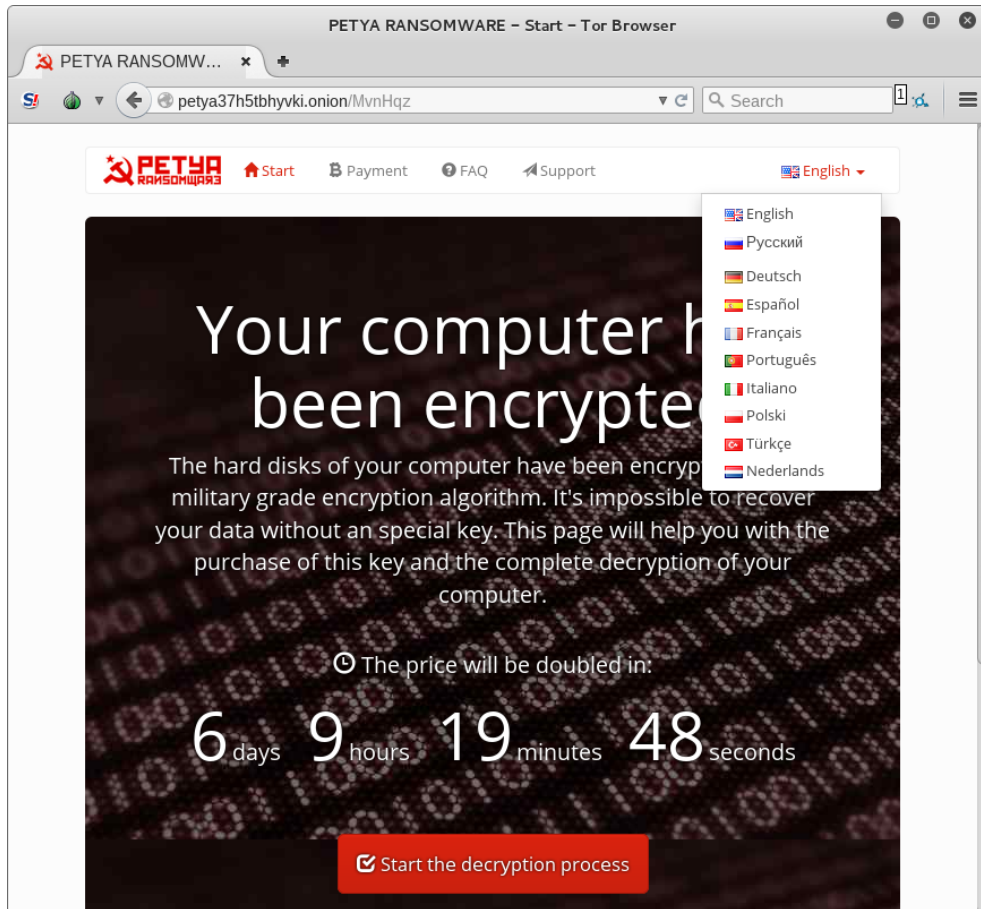


However, it is not true that the full disk is encrypted. If we view it by forensic tools, we can see many valid elements, including strings.

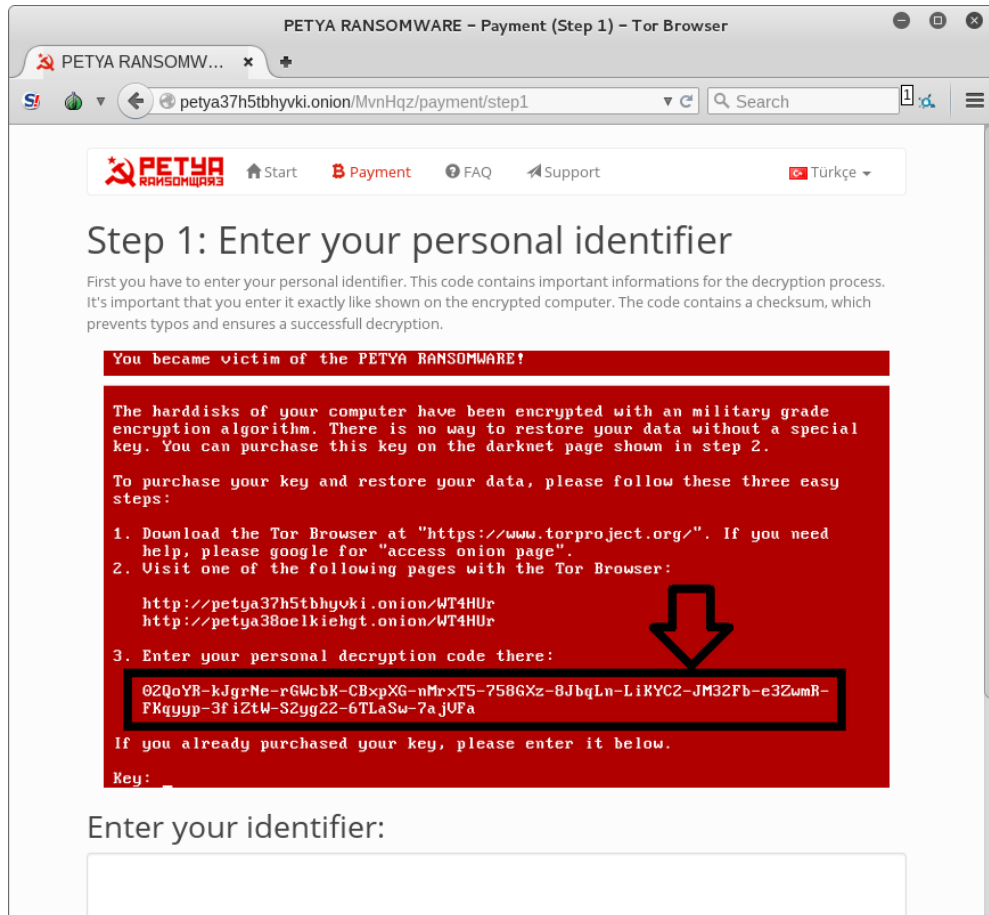
```
02BBAD30 04 00 00 00 01 00 00 00 7B 36 44 46 44 37 43 35 .....{6DFD7C5
02BBAD40 43 2D 32 34 35 31 2D 31 31 64 33 2D 41 32 39 39 C-2451-11d3-A299
02BBAD50 2D 30 30 43 30 34 46 38 45 46 36 41 46 7D 00 00 -00C04F8EF6AF)..
02BBAD60 F0 FF FF FF E0 02 7C 00 20 03 7C 00 88 05 7C 00 d' ' ' ' | . | . . | .
02BBAD70 D8 FF FF FF 76 6B 10 00 04 00 00 80 00 00 00 00 R' ' ' vk . . . . . € . . . .
02BBAD80 04 00 00 00 01 00 71 DB 53 68 75 74 64 6F 77 6E .....qũShutdown
02BBAD90 52 65 61 73 6F 6E 55 49 E0 FF FF FF 76 6B 08 00 ReasonUI€' ' ' vk..
02BBADA0 12 00 00 00 B8 03 7C 00 01 00 00 00 01 00 00 00 . . . . | . . . . .
02BBADB0 30 30 30 30 33 43 30 41 E8 FF FF FF 2A 00 32 00 00003C0Ač' ' ' *.2.
02BBADC0 35 00 30 00 2C 00 31 00 36 00 39 00 00 00 00 00 5.0.,.1.6.9.....
02BBADD0 E0 FF FF FF 76 6B 08 00 12 00 00 00 F0 03 7C 00 š' ' ' vk . . . . . d. | .
02BBADE0 01 00 00 00 01 00 01 00 30 30 30 30 34 30 30 31 .....00004001
02BBADF0 E8 FF FF FF 2A 00 32 00 35 00 30 00 2C 00 31 00 č' ' ' *.2.5.0.,.1.
02BBAE00 38 00 36 00 00 00 01 00 80 FE FF FF D8 D6 7B 00 8.6.....€ť' ' RÖ{. Sector 89559
```

Website for the victim

We noted that the website for the victim is well prepared and very informative. The menu offers several language versions, but so far only English works:



It also provides a step-by-step process on how affected users can recover their data:



- Step1: Enter your personal identifier
- Step2: Purchase Bitcoins
- Step3: Do a Bitcoin transaction
- Step4: Wait for confirmation

We expect that cybercriminals release as little information about themselves as possible. But in this case, the authors and/or distributors are very open, sharing the team name—”Janus Cybercrime Solutions”—and the project release date—12th December 2015. Also, they offer a news feed with updates, including press references about them:

News

24.03.2015

WARNING

Do not restore the MBR with the Windows Recovery Tools. This could destroy your data completely!

There are a lot of wrong informations online. If you are looking for reliable informations, please visit <https://blog.gdata.de/2016/03/28222-ransomware-petya-verschlusset-die-festplatte>

16.12.2015

Petya launched

Today we launched the Petya Ransomware Project.

Copyright © 2016 Janus Cybercrime Solutions™

In case of questions or problems, it is also possible to contact them via a Web form.



Start

Payment

FAQ

Support

Inside

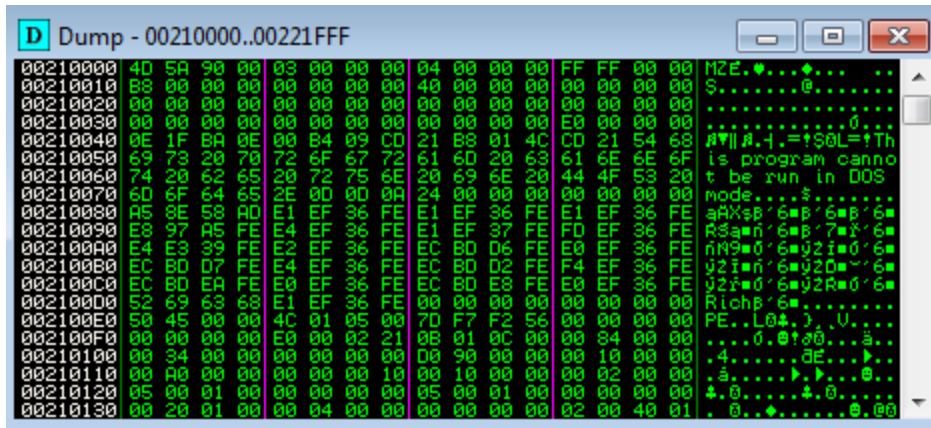
Stage 1

As we have stated earlier, the first stage of execution is in the Windows executable. It is packed in a good quality FUD/cryptor that's why we cannot see the malicious code at first. Executions starts in a layer that is harmless and used only for the purpose of deception and protecting the payload. The real malicious functionality is inside the payload dynamically unpacked to the memory.

Below you can see the memory of the running process. The code belonging to the original EXE is marked red. The unpacked malicious code is marked blue:

00200000	00003000				Priv RW RW
00210000	00012000				Priv RWE RWE
00230000	00003000				Priv RW RW
00240000	00011000				Map R R
00280000	00029000				Priv RW RW
00400000	00001000	Petya		PE header	Imag ??? Gua; RWE
00401000	00027000	Petya	.text	code	Imag ??? Gua; RWE
00420000	00000000	Petya	.rdata	imports	Imag ??? Gua; RWE
00430000	00005000	Petya	.data	data	Imag ??? Gua; RWE
00430000	00002000	Petya	.rsrc	resources	Imag ??? Gua; RWE
00430000	00003000	Petya	.reloc	relocations	Imag ??? Gua; RWE

The unpacked content is another PE file:



However, if we try to dump it, we don't get a valid executable. Its data directories are destroyed. The PE file have been processed by the cryptor in order to be loaded in a continuous space, not divided by sections. It lost the ability to run independently, without being loaded by the cryptor's stub. Addresses saved as RVA are in reality raw addresses.

I have remapped them using a custom tool, and it revealed more information, i.e. the name of this PE file is *Setup.dll*:

Offset	Name	Value	Meaning
A7E0	Characteristics	0	
A7E4	TimeDateStamp	56F2F77D	
A7E8	MajorVersion	0	
A7EA	MinorVersion	0	
A7EC	Name	FE12	Setup.dll
A7F0	Base	1	
A7F4	NumberOfFunctions	1	
A7F8	NumberOfNames	1	
A7FC	AddressOfFunctions	FE08	
A800	AddressOfNames	FE0C	
A804	AddressOfNameOrdinals	FE10	

Details				
Offset	Ordinal	Function RVA	Name RVA	Name
A808	1	1DC0	FE1C	._ZuWQdweafdsg345312@0

UPDATE: if we catch the process of unpacking in correct moment, we can dump the DLL before it is destroyed. The resulting payload is: 7899d6090efae964024e11f6586a69ce

As the name suggest, the role of the payload is to setup everything for the next stage. First, it generates a unique key that will be used for further encryption. This key must be also known to attackers. That's why it is encrypted by ECC and displayed to the victim as a personal

identifier, that must be send to attackers via personalized page.

Random values are retrieved by Windows Crypto API function: CryptGenRandom. Below, it gets 128 random bytes:

```

003D5F41  PUSH EDI
003D5F42  MOV EDI,DWORD PTR SS:[EBP+14]
003D5F45  LEA EAX,DWORD PTR SS:[EBP+14]
003D5F48  PUSH F0000000
003D5F4D  PUSH 1
003D5F4F  XOR EBX,EBX
003D5F51  PUSH EBX
003D5F52  PUSH EBX
003D5F53  PUSH EAX
003D5F54  MOV DWORD PTR DS:[EDI],EBX
003D5F56  CALL DWORD PTR DS:[3DA00C]          ADVAPI32.CryptAcquireContextA
003D5F5C  TEST EAX,EAX
003D5F5E  JNZ SHORT 003D5F65
003D5F60  PUSH -3C
003D5F62  POP EAX
003D5F63  JMP SHORT 003D5F8E
003D5F65  PUSH ESI
003D5F66  PUSH DWORD PTR SS:[EBP+C]
003D5F69  MOV ESI,DWORD PTR SS:[EBP+10]
003D5F6C  PUSH ESI
003D5F6D  PUSH DWORD PTR SS:[EBP+14]
003D5F70  CALL DWORD PTR DS:[3DA008]          ADVAPI32.CryptGenRandom
003D5F76  TEST EAX,EAX
003D5F78  JNZ SHORT 003D5F7F
003D5F7A  PUSH -3C
003D5F7C  POP EAX
003D5F7D  JMP SHORT 003D5F8D
003D5F7F  PUSH EBX
003D5F80  PUSH DWORD PTR SS:[EBP+14]
003D5F83  CALL DWORD PTR DS:[3DA010]          ADVAPI32.CryptReleaseContext
003D5F89  MOV DWORD PTR DS:[EDI],ESI
EAX=00000001

```

Address	Hex dump	ASCII
0012E7D8	2A 41 F1 0A 18 0B 8A 54	*A".↑0T
0012E7E0	31 E2 B6 3D 62 A3 10 21	10A=bu↑
0012E7E8	F7 24 D4 A3 E9 E2 06 30	,sd'0000
0012E7F0	29 74 73 36 F6 52 FC EF)ts6+RA'
0012E7F8	C8 A1 0A BE 9E E6 27 8D	ti,2x3'2
0012E800	C8 20 B0 FF 3C 6A F0 55	ti <j-U
0012E808	8E 84 9C 4E 00 F5 FC 4C	AatN.SAL
0012E810	3F 6C 29 B6 9B B4 DF 45	?()AT-#E
0012E818	9A D1 D8 97 87 08 83 75	u0ésScu
0012E820	41 16 C3 06 9A 79 80 CB	A.↑uyCt
0012E828	15 E0 59 B0 40 BE 40 7D	3dy@e00
0012E830	49 26 BA A3 35 B7 4E 22	I ú5EN"
0012E838	18 51 45 5D DC B5 6E 59	↑0E]_AnY
0012E840	F7 5F 1A 6D 72 B8 FA 93	_→mS'0
0012E848	03 E6 C6 57 B0 29 53 09	↑SAM@ S.
0012E850	0F 59 04 EF 16 AD 8B EA	*V♦' _s0r
0012E858	D2 00 00 00 77 5E 3D 00	D...w"
0012E860	98 EA 12 00 28 ED 12 00	z8φ.(Yφ.

Making of onion addresses:

```

003D8400  PUSH 3DA760          ASCII "http://petya37h5tbhyuki.onion/"
003D840F  PUSH ESI
003D8410  CALL 003D91D0
003D8415  MOV EAX,DWORD PTR DS:[EDI]
003D8417  ADD ESP,0C
003D841A  MOV DWORD PTR DS:[ESI+1E],EAX
003D841D  MOV AX,WORD PTR DS:[EDI+4]
003D8421  MOV WORD PTR DS:[ESI+22],AX
003D8425  LEA EAX,DWORD PTR DS:[ESI+24]
003D8428  PUSH 24
003D842A  PUSH 3DA758          ASCII "http://petya5koahstsf7sv.onion/"
003D842F  PUSH EAX
003D8430  CALL 003D91D0
003D8435  MOV EAX,DWORD PTR DS:[EDI]
003D8437  ADD ESP,0C
003D843A  MOV DWORD PTR DS:[ESI+48],EAX
003D843D  MOV AX,WORD PTR DS:[EDI+4]
003D8441  MOV WORD PTR DS:[ESI+4C],AX
003D8445  MOV EAX,ESI
003D8447  POP EDI
003D8448  POP ESI
003D8449  MOV ESP,EBP
003D844B  POP EBP
003D844C  RETN
003D844D  SUB ESP,6D4
003D8453  PUSH EBX
003D8454  PUSH EBP
003D8455  XOR EBX,EBX
003D8457  POP EBP
003D8459  MOV DWORD PTR SS:[ESP+C],EBX
003D845D  PUSH ESI
003D845E  MOV ESI,EBX
003D8460  MOV DWORD PTR SS:[ESP+C],ESI
003D8464  PUSH EDI
003D8465  TEST EBP,EBP

```

Registers (MMX)

EAX 001B3468

ECX 00000000

EDX 001B3463

EBX 00000000

ESP 0012E8AC

EBP 0012E770

ESI 001B3465

EDI 001B3F48 ASCII "aeskisC8ps8"

EIP 003D840A

C 0 ES 0023 32bit 0FFFFFFFF

D 0

E 0 LastErr ERROR_SUCCESS (00000000)

EFL 00000212 (NO,NB,NE,A,NS,PO,GE,G)

MM0 0000 0000 0000 0000

MM1 0000 0000 0000 0000

MM2 0000 0000 0000 0000

MM3 0000 0000 0000 0000

MM4 0000 0000 0000 0000

MM5 0000 0000 0000 0000

MM6 0000 0000 0000 0000

MM7 0000 003B 0000 0000

Retrieving parameters of the disk using DeviceIoControl

```

C *G.P.U* - main thread, module Petya
0041280A . . . . . PUSH Petya.004318E0
0041280F . . . . . PUSH 0x103
00412814 . . . . . PUSH 0x104
00412819 . . . . . PUSH EAX
0041281A . . . . . CALL Petya.00427469
0041281F . . . . . ADD ESP,0x14
00412822 . . . . . LEA EAX,[LOCAL.134]
00412828 . . . . . PUSH 0x0
0041282A . . . . . PUSH 0x0
0041282C . . . . . PUSH 0x3
0041282E . . . . . PUSH 0x0
00412830 . . . . . PUSH 0x7
00412832 . . . . . PUSH 0x00000000
00412837 . . . . . PUSH EAX
00412838 . . . . . CALL DWORD PTR DS:[&KERNEL32.CreateFileW]
0041283E . . . . . MOV ESI,EAX
00412840 . . . . . MOV [LOCAL.144],ESI
00412846 . . . . . CMP ESI,-0x1
00412849 . . . . . JNZ SHORT Petya.00412855
0041284B . . . . . CALL Petya.00407E13
00412850 . . . . . JMP Petya.00412948
00412855 . . . . . PUSH 0x6
00412857 . . . . . POP ECX
00412858 . . . . . PUSH 0x0
0041285A . . . . . LEA EAX,[LOCAL.142]
00412860 . . . . . MOV [LOCAL.147],0x1010101
0041286A . . . . . PUSH EAX
0041286B . . . . . PUSH ECX
0041286C . . . . . LEA EAX,[LOCAL.3]
0041286F . . . . . MOV [LOCAL.142],ECX
00412875 . . . . . PUSH EAX
00412876 . . . . . PUSH 0x4
00412878 . . . . . LEA EAX,[LOCAL.147]
0041287E . . . . . PUSH EAX
0041287F . . . . . PUSH 0x170002
00412884 . . . . . PUSH ESI
00412885 . . . . . CALL DWORD PTR DS:[&KERNEL32.DeviceIoControl]
0041288B . . . . . TEST EAX,EAX
0041288D . . . . . JE Petya.0041293C

Arg4 = 004318E0
Arg3 = 00000103
Arg2 = 00000104
Arg1 = 0012FF80
Petya.00427469

hTemplateFile = NULL
Attributes = 0
Mode = OPEN_EXISTING
pSecurity = NULL
ShareMode = FILE_SHARE_READ|FILE_SHARE_WRITE|4
Access = GENERIC_READ
FileName = "???\x12?A?????\x12?????"
CreateFileW

kernel32.75823C45
pOverlapped = NULL

pBytesReturned = 0012FF80
OutBufferSize = A661F6CC (-1503529268.)

OutBuffer = 0012FF80
InBufferSize = 0x4

InBuffer = 0012FF80
IoControlCode = 0x170002
hDevice = NULL
DeviceIoControl

```

Read/write to the disk:

```

00218964 . . . . . PUSH EBX
00218966 . . . . . PUSH ECX
00218967 . . . . . PUSH EBX
00218968 . . . . . PUSH ESI
00218969 . . . . . PUSH EDI
0021896A . . . . . XOR EBX,EBX
0021896C . . . . . MOV EDI,EDX
0021896E . . . . . PUSH EBX
0021896F . . . . . PUSH 30000000
00218974 . . . . . PUSH 3
00218976 . . . . . PUSH EBX
00218977 . . . . . PUSH 3
00218979 . . . . . PUSH C0000000
0021897E . . . . . PUSH ECX
0021897F . . . . . CALL DWORD PTR DS:[21A038] kernel32.CreateFileA
00218985 . . . . . MOV ESI,EAX
00218987 . . . . . CMP ESI,-1
0021898A . . . . . JNZ SHORT 00218997
0021898C . . . . . PUSH EAX
0021898D . . . . . CALL DWORD PTR DS:[21A034] kernel32.CloseHandle
00218993 . . . . . XOR EAX,EAX
00218995 . . . . . JMP SHORT 002189D5
00218997 . . . . . MOV ECX,DWORD PTR SS:[EBP+8]
0021899A . . . . . MOV EAX,DWORD PTR SS:[EBP+C]
0021899D . . . . . PUSH EBX
0021899E . . . . . SHLD EAX,ECX,9
002189A3 . . . . . MOV EAX,EBX

DS:[0021A038]=7581CEE8 (kernel32.CreateFileA)

address Hex dump ASCII
0012F488 37 37 37 37 37 37 37 37 77777777
0012F490 37 37 37 37 37 37 37 37 77777777
0012F498 37 37 37 37 37 37 37 37 77777777
0012F4A0 37 37 37 37 37 37 37 37 77777777
0012F4A8 37 37 37 37 37 37 37 37 77777777
0012F4B0 37 37 37 37 37 37 37 37 77777777
0012F4B8 37 37 37 37 37 37 37 37 77777777
0012F4C0 00000000
0012F204 0012F254 T. File Name = "\.PhysicalDrive0"
0012F208 C0000000 ... Access = GENERIC_READ|GENERIC_WRITE
0012F20C 00000003 ... ShareMode = FILE_SHARE_READ|FILE_SHARE_WRITE
0012F210 00000000 ... pSecurity = NULL
0012F214 00000003 ... Mode = OPEN_EXISTING
0012F218 30000000 ... Attributes = NO_BUFFERING|RANDOM_ACCESS
0012F21C 00000000 ... hTemplateFile = NULL
0012F220 00000001
0012F224 00000001

```

After overwriting the beginning of the disk, it intentionally crashes the system, using an undocumented function NtRaiseHardError:

```

00219012 MOV DWORD PTR SS:[EBP-18],1
00219019 PUSH EAX
0021901A PUSH ESI
0021901B PUSH DWORD PTR SS:[EBP-4]
0021901E MOV DWORD PTR SS:[EBP-C],2
0021902B CALL DWORD PTR DS:[21A014] ADVAPI32.AdjustTokenPrivileges
00219031 CALL DWORD PTR DS:[21A03C] kernel32.GetLastError
00219033 TEST EAX,EAX
00219035 JNZ SHORT 00218FF6
00219035 PUSH 21A7B4 ASCII "NtRaiseHardError"
0021903A PUSH 21A7C8 ASCII "NTDLL.DLL"
00219043 CALL DWORD PTR DS:[21A044] kernel32.GetModuleHandleA
00219045 PUSH EAX
0021904C CALL DWORD PTR DS:[21A040] kernel32.GetProcAddress
0021904F LEA ECX,DWORD PTR SS:[EBP-8]
00219050 PUSH ECX
00219050 PUSH 6 OptionShutdownSystem
00219052 PUSH ESI
00219053 PUSH ESI
00219054 PUSH ESI
00219055 PUSH C0000350
00219055 CALL EAX ntdll.ZwRaiseHardError
0021905C XOR EAX,EAX
0021905E ADD ESP,18

```

At this point, first stage of changes on the disk have been already made. Below you can see the MBR overwritten by the Petya's boot loader:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F		
00000000	FA	66	31	C0	8E	D0	8E	C0	8E	D8	BC	00	7C	FB	88	16	af1RZĐZŕZŕL. ũ..	Sector 0
00000010	93	7C	66	B8	20	00	00	00	66	BB	22	00	00	00	B9	00	" f, ...f>"...a.	
00000020	80	E8	14	00	66	48	66	83	F8	00	75	F5	66	A1	00	80	ěč..fHf.ř.uóř`.ē	
00000030	EA	00	80	00	00	F4	EB	FD	66	50	66	31	C0	52	56	57	ę.ē..ōëýfPf1ŕRVW	
00000040	66	50	66	53	89	E7	66	50	66	53	06	51	6A	01	6A	10	fPfS%çfPFS.Qj.j.	
00000050	89	E6	8A	16	93	7C	B4	42	CD	13	89	FC	66	5B	66	58	%óŠ." 'Bí.%uf[FX	
00000060	73	08	50	30	E4	CD	13	58	EB	D6	66	83	C3	01	66	83	s.P0aÍ.XeŌf.Ā.f.	
00000070	D0	00	81	C1	00	02	73	07	8C	C2	80	C6	10	8E	C2	5F	Đ..Ā..s.SĀeC.ŽĀ	
00000080	5E	5A	66	58	C3	60	B4	0E	AC	3C	00	74	04	CD	10	EB	^ZfXĀ`'.<.t.Í.ē	
00000090	F7	61	C3	00	00	00	00	00	00	00	00	00	00	00	00	00	±aĀ.....	
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00Ōio=..ē	
000001C0	21	00	07	DF	13	0C	00	08	00	00	00	20	03	00	00	DF	!..β.....β	
000001D0	14	0C	07	FE	FF	FF	00	28	03	00	00	D0	1C	03	00	00	...ť`.(...Đ....	
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AAUS	
00000200	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	7777777777777777	Sector 1
00000210	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	7777777777777777	
00000220	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	7777777777777777	
00000230	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	7777777777777777	
00000240	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	37	7777777777777777	

Next few sectors contains backup of original data XORed with '7'. After that we can find the copied Petya code (starting at 0x4400 – sector 34).

We can also see the strings that are displayed in the ransom note, copied to the the disk:

```

000005D60 74 68 65 20 50 45 54 59 41 20 52 41 4E 53 4F 4D the PETYA RANSOM
000005D70 57 41 52 45 21 0D 0A 00 0D 0A 20 54 68 65 20 68 WARE!..... The h
000005D80 61 72 64 64 69 73 6B 73 20 6F 66 20 79 6F 75 72 arddisks of your
000005D90 20 63 6F 6D 70 75 74 65 72 20 68 61 76 65 20 62 computer have b
000005DA0 65 65 6E 20 65 6E 63 72 79 70 74 65 64 20 77 69 een encrypted wi
000005DB0 74 68 20 61 6E 20 6D 69 6C 69 74 61 72 79 20 67 th an military g
000005DC0 72 61 64 65 0D 0A 20 65 6E 63 72 79 70 74 69 6F rade.. encryptio
000005DD0 6E 20 61 6C 67 6F 72 69 74 68 6D 2E 20 54 68 65 n algorithm. The
000005DE0 72 65 20 69 73 20 6E 6F 20 77 61 79 20 74 6F 20 re is no way to
000005DF0 72 65 73 74 6F 72 65 20 79 6F 75 72 20 64 61 74 restore your dat
000005E00 61 20 77 69 74 68 6F 75 74 20 61 20 73 70 65 63 a without a spec Sector 47

```

Stage 2

Stage 2 is inside the code written to the disk's beginning. This code uses 16 bit architecture.

Execution starts with a boot loader, that loads into memory the tiny malicious kernel. Below we can see execution of the loading function. Kernel starts at sector 34 and it is 32 sectors long (including saved data):

```

seg000:0012      mov     eax, 32      ; sectors_number
seg000:0018      mov     ebx, 34      ; start_sector
seg000:001E      mov     cx, 8000h    ; output_address
seg000:0021      loc_21:
seg000:0021      call   read_sector   ; CODE XREF: seg000:002A↓j
seg000:0024      dec     eax
seg000:0026      cmp     eax, 0
seg000:002A      jnz    short loc_21
seg000:002C      mov     eax, ds:8000h
seg000:0030      jmp    far ptr 0:8000h ; jump to the copied code

```

Beginning of the kernel:

```

seg000:8000 loc_8000:
seg000:8000
seg000:8000      jmp    loc_8640

```

Checking if the data is already encrypted is performed using one byte flag that is saved at the beginning of sector 54. If this flag is unset, program proceeds to the fake CHKDSK scan. Otherwise, it displays the main red screen.

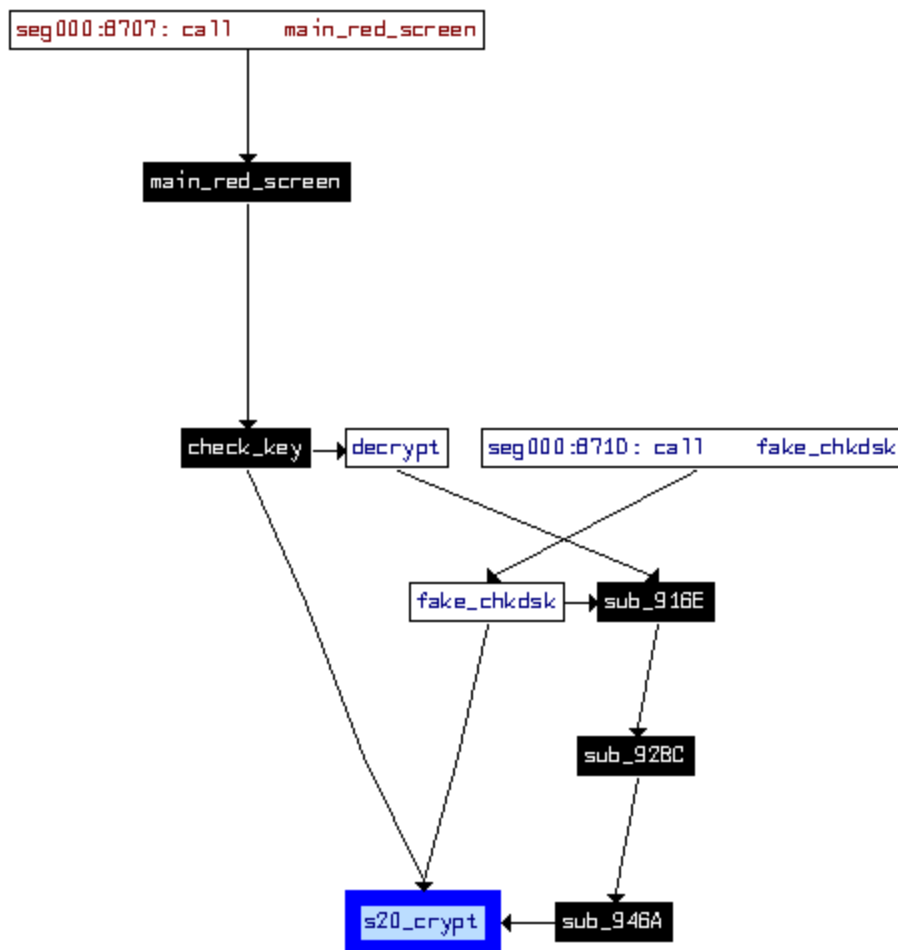
```

seg000:86D9      push    0                ; read
seg000:86DB      push    1
seg000:86DD      push    0
seg000:86DF      push    54               ; sector
seg000:86E1      lea    ax, [bp-286h]    ; out_buf
seg000:86E5      push    ax
seg000:86E6      mov    al, [bp-2]
seg000:86E9      push    ax
seg000:86EA      call   disk_read_or_write
seg000:86ED      add    sp, 0Ch
seg000:86F0      or     al, al
seg000:86F2      jz     short loc_86F7   ; is data encrypted?
seg000:86F4      error2:
seg000:86F4      jmp    error            ; CODE XREF: seg000:86D7↑j
seg000:86F7      ; -----
seg000:86F7      loc_86F7:
seg000:86F7      cmp    byte ptr [bp-286h], 1 ; is data encrypted?
seg000:86F7      jb    short to_fake_chkds
seg000:86FC

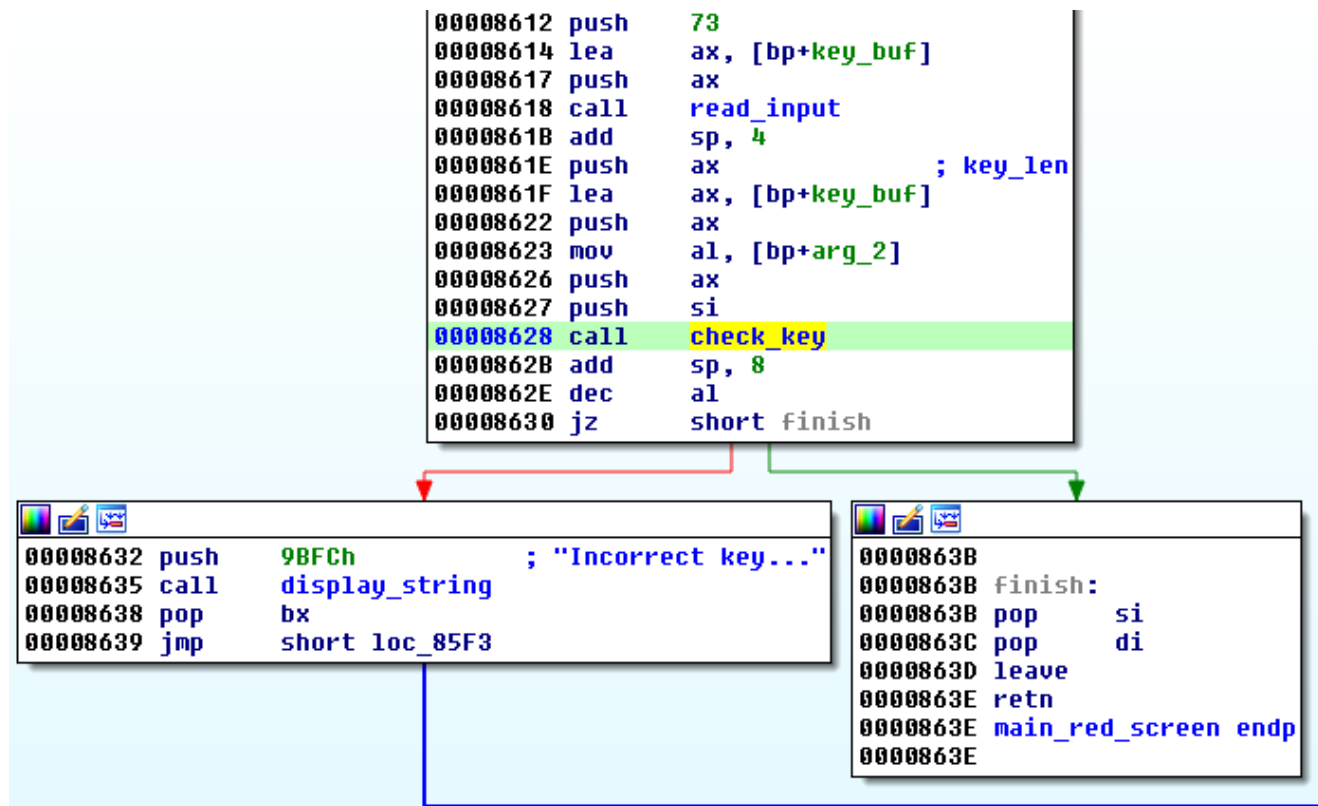
```

The fake CHKDSK encrypts MFT using Salsa20 algorithm. The used key is 32 byte long, read from the address 0x6C01. After that, the key gets erased.

Salsa20 is used in several places in Petya's code – for encryption, decryption and key verification. See the diagram below:



Inside the same function that displays the red screen, the `Key` checking routine is called. First, user is prompted to supply the key. The maximal input length is 73 bytes, the minimal is 16 bytes.

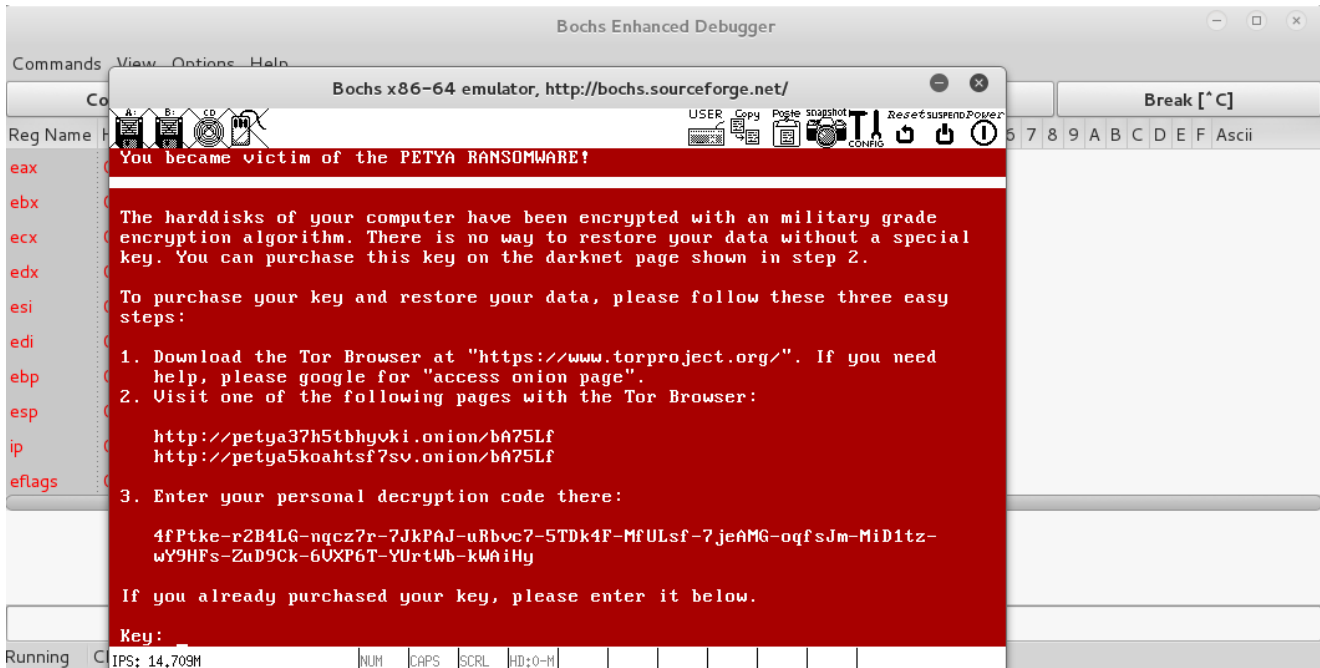


Debugging

Of course, we cannot debug this stage of Petya via typical userland debuggers that are the casual tools in analyzing malware. We need to go to the low level. The simplest way (in my opinion) is to use Bochs internal debugger. We need to make a full dump of the infected disk. Then, we can load it under Bochs.

I used the following Bochs configuration ('infected.dsk' is my disk dump): [bochsrc.txt](#)

This is how it looks running under Bochs:



Key verification

Key verification is performed in the following steps:

1. Input from the user is read.
 - o Accepted
charset: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZJKL MNPQRSTUVWXYZ – if the character outside of this charset occurred, it is skipped.
 - o Only first **16 bytes** are stored
2. The **supplied key** is encoded by a custom algorithm. **Encoded key** is 32 bytes long.
3. Data from sector 55 (512 bytes) is read into memory // *it will be denoted as verification buffer*
4. The value stored at physical address 0x6c21 (just before the Tor address) is read into memory. It is an 8 byte long array, unique for a specific infection. // *it will be denoted as nonce*
5. The **verification buffer** is encrypted by 256 bit Salsa20 with **encoded key** and the **nonce**
6. If, as the result of applied procedure, **verification buffer** is fully filled with '7' – it means the **supplied key** is correct.

Example: **encoded key** versus **supplied key**:

P.Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
0x000076D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	AB	62b
0x000076E0	AC	64	AD	66	AE	68	AF	6A	B0	6C	B2	70	B1	6E	B2	70	.d.f.h.j.l.p.n.p
0x000076F0	AE	68	AD	66	AB	62	AC	64	AC	64	AD	66	AE	68	31	32	.h.f.b.d.d.f.h
0x00007700	33	34	35	36	38	37	38	34	33	31	32	32	33	34	10	00	34568784312234

The algorithm for encoding the supplied key is very simple:

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters.

[Learn more about bidirectional Unicode characters](#)

[Show hidden characters](#)

```
bool encode(char* key, BYTE *encoded)
```

```
{
```

```
if (!key || !encoded) {
```

```
printf("Invalid buffer\n");
```

```
return false;
```

```
}
```

```
size_t len = strlen(key);
```

```
if (len < 16) {
```

```
printf("Invalid key\n");
```

```
return false;
```

```
}
```

```
if (len > 16) len = 16;
```

```
int i, j;
```

```
i = j = 0;
```

```
for (i = 0, j = 0; i < len; i++, j += 2) {
```

```
char k = key[i];
```

```
encoded[j] = k + 'z';
```

```
encoded[j+1] = k * 2;
```

```
}
```

```
encoded[j] = 0;
```

```
encoded[j+1] = 0;
```

```
return true;
```

```
}
```

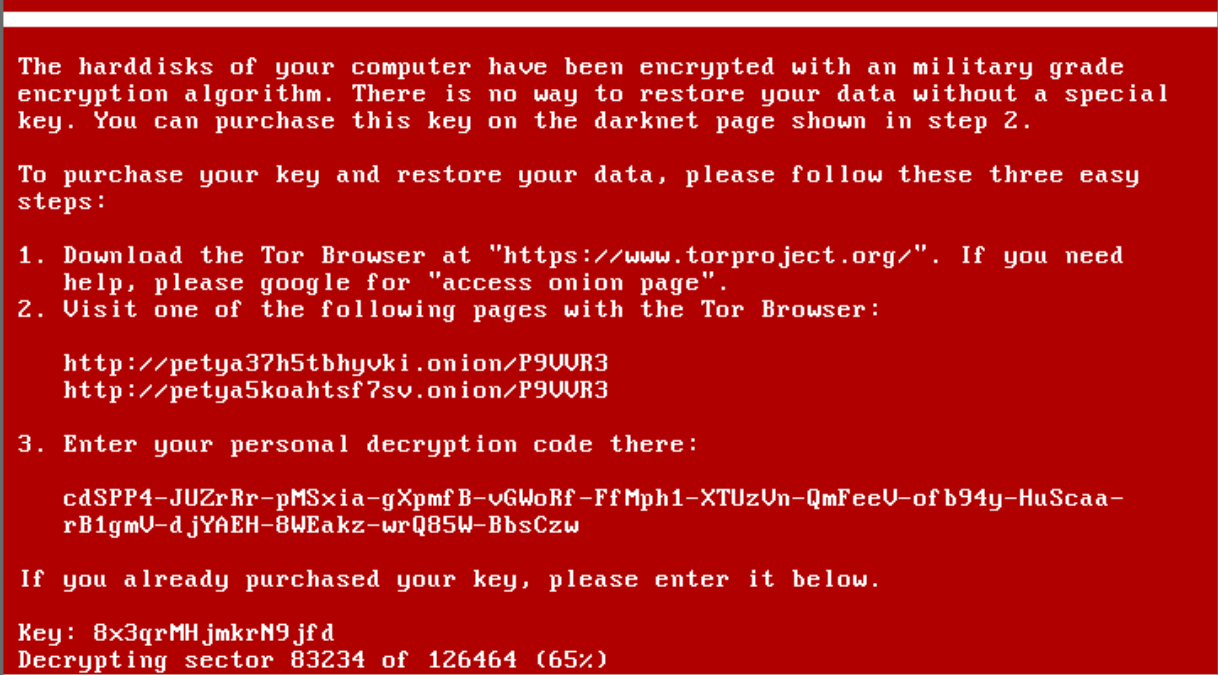
[view raw](#)

[petya_encoder.cpp](#)

hosted with ❤ by [GitHub](#)

Valid key is important for the process of decryption. If we supply a bogus key and try to pass it as valid by modifying jump conditions, Petya will recover the original MBR but other data will not be decrypted properly and the operating system will not run.

When the key passed the check, Petya shows the message “Decrypting sectors” with a progress.



The haddisks of your computer have been encrypted with an military grade encryption algorithm. There is no way to restore your data without a special key. You can purchase this key on the darknet page shown in step 2.

To purchase your key and restore your data, please follow these three easy steps:

1. Download the Tor Browser at "<https://www.torproject.org/>". If you need help, please google for "access onion page".
2. Visit one of the following pages with the Tor Browser:

```
http://petya37h5tbhyvki.onion/P9UUR3  
http://petya5koahtsf7sv.onion/P9UUR3
```

3. Enter your personal decryption code there:

```
cdSPP4-JUZrRr-pMSxia-gXpmfB-vgWoRf-FfMph1-XTUzUn-QmFeeU-ofb94y-HuScaa-  
rB1gmU-djYAEH-8WEakz-wrQ85W-BbsCzw
```

If you already purchased your key, please enter it below.

```
Key: 8x3qrMHjmkR9jfd  
Decrypting sector 83234 of 126464 (65%)
```

After it finishes, it asks to reboot the computer. Below is Petya’s last screen, showing that user finally got rid of this ransomware:

Please reboot your computer!



Conclusion

In terms of architecture, Petya is very advanced and atypical. Good quality FUD, well obfuscated dropper – and the heart of the ransomware – a little kernel – depicts that authors are highly skilled. However, the chosen low-level architecture enforced some limitations, i.e.: small size of code and inability to use API calls. It makes cryptography difficult. That's why the key was generated by the higher layer – the windows executable. This solution works well, but introduces a weakness that allowed to restore the key (if we manage to catch Petya at Stage 1, before the key is erased). Moreover, authors tried to use a ready made [Salsa20](#) implementation and make slight changes in order to adopt it to 16-bit architecture. But they didn't realized, that changing size of variables triggers serious vulnerabilities (detailed description you can find in [CheckPoint's article](#)).

Most of the ransomware authors take care of the user experience, so that even a non technical person will have easy way to make a payment. In this case, user experience is very bad. First – denying access to the full system is not only harmful to a user, but also for the ransomware distributor, because it makes much harder for the victim to pay the ransom. Second – the individual identifier is very long and it cannot be copied from the screen. Typing it without mistake is almost impossible.

Overall, authors of Petya ransomware wrote a good quality code, that, however – missed the goals. Ransomware running in userland can be equally or more dangerous.

Appendix

About Petya by other vendors:

Read also:

- <http://www.invoke-ir.com/2015/05/ontheforensictrail-part2.html> – Master Boot Record
 - <http://sysforensics.org/2012/06/mbr-malware-analysis/> – MBR malware analysis
 - <https://socprime.com/en/blog/dismantling-killdisk-reverse-of-the-blackenergy-destructive-component/> – Dismantling KillDisk: reverse of the BlackEnergy destructive component (another malware attacking hard disk)
-

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://hshzrd.wordpress.com>.