

Petya - Taking Ransomware To The Low Level | Malwarebytes Labs

By Malwarebytes Labs

Published: 2016-03-31 · Archived: 2026-04-05 14:13:24 UTC

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

- [dfcced98585128312b62b42a2a250dd2](#) – zipped package containing the malicious executable
 - [af2379cc4d607a45ac44d62135fb7015](#) – the main executable
 - [7899d6090efae964024e11f6586a69ce](#) – Setup.dll
 - [d80fc07cc293bcd36e630d45a34aca11](#) – a dump of Petya bootloader + kernel

Main executable from another campaign (PDF icon)

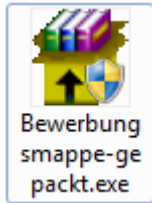
- [a92f13f3a1b3b39833d3cc336301b713](#)

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

Behavioral analysis

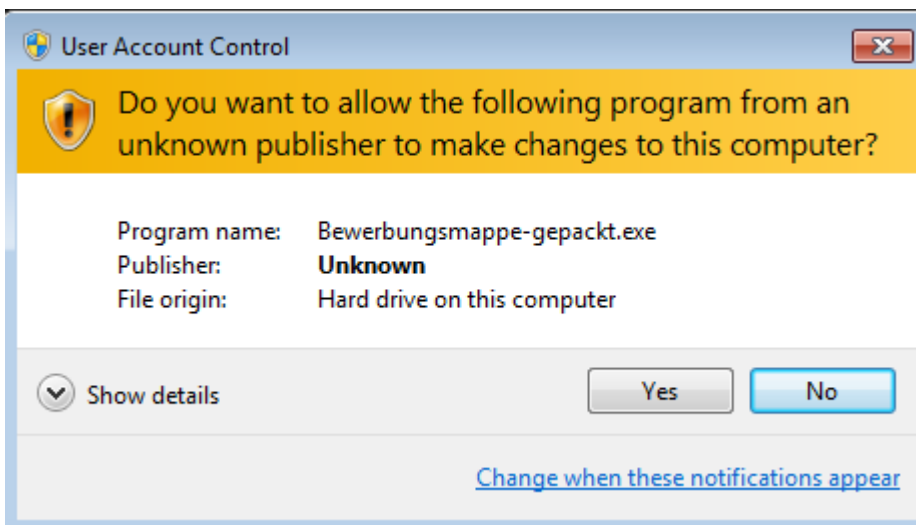
This ransomware is delivered via scam emails themed as a job application. E-mail comes with a Dropbox link, where the malicious ZIP is hosted. This initial ZIP contains two elements:

- 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):

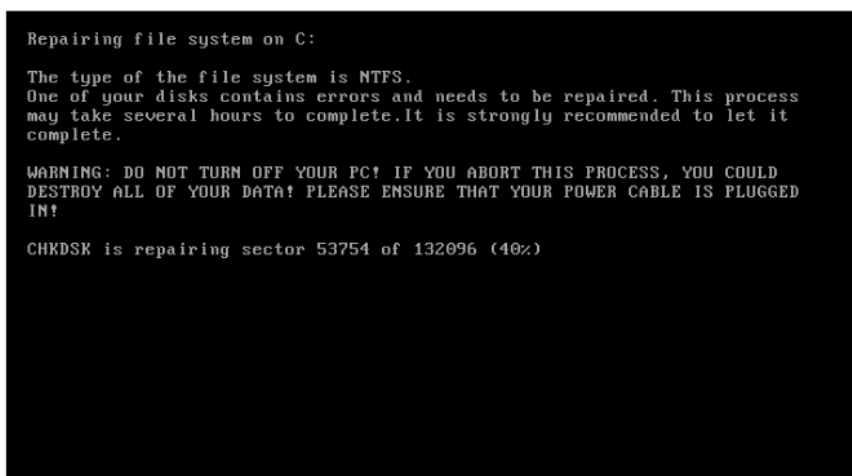


In order to execute its harmful features, it needs to run with Administrator privileges. However, it doesn't even try to deploy any [user account control \(UAC\)](#) bypass technique. It relies fully on social engineering.

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:



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



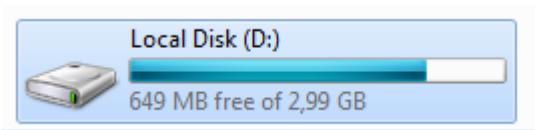
Pressing a key leads to the main screen with the ransom note and all information necessary to reach the Web panel and proceed with the payment:



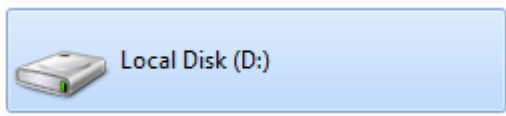
Infection stages

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.

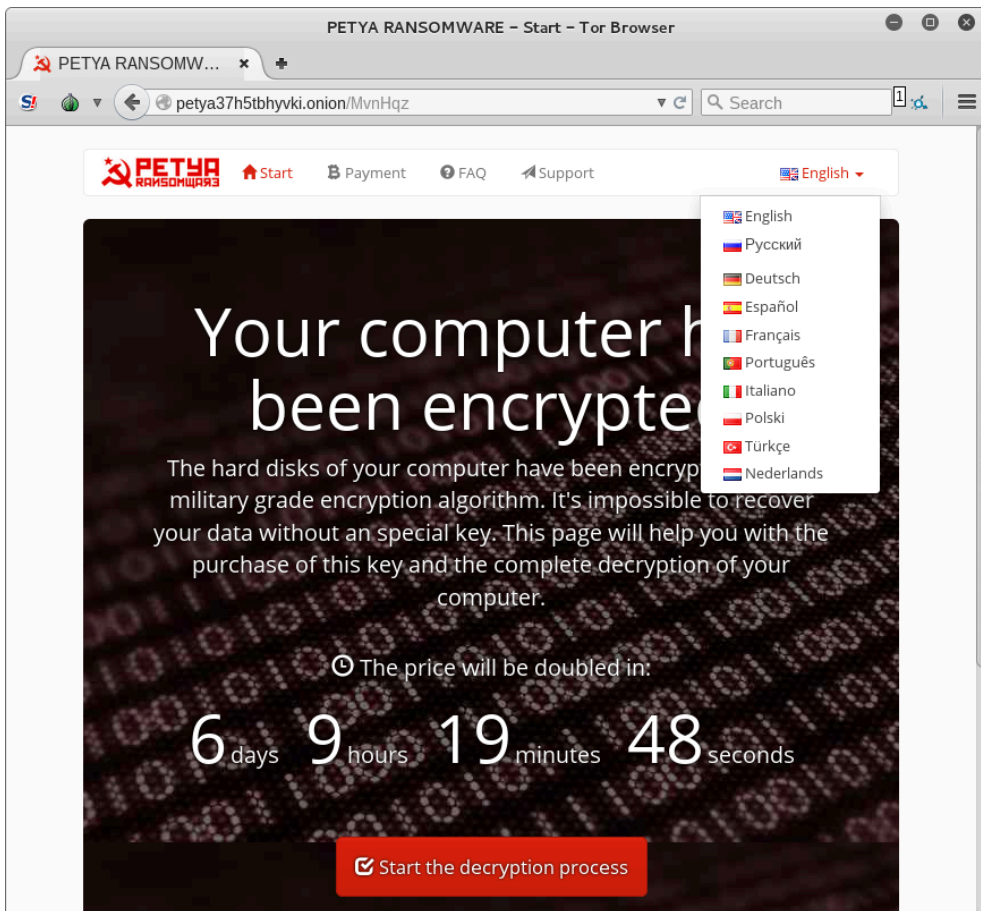
```

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'f.|.|...|.
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 .....qShutdown
02BBAD90 52 65 61 73 6F 6E 55 49 E0 FF FF FF 76 6B 08 00 ReasonU'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 00003C0A'*.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 f'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 8'*.2.5.0.,.1.
02BBAE00 38 00 36 00 00 00 01 00 80 FE FF FF D8 D6 7B 00 8.6.....e' RÖ{. Sector 89559

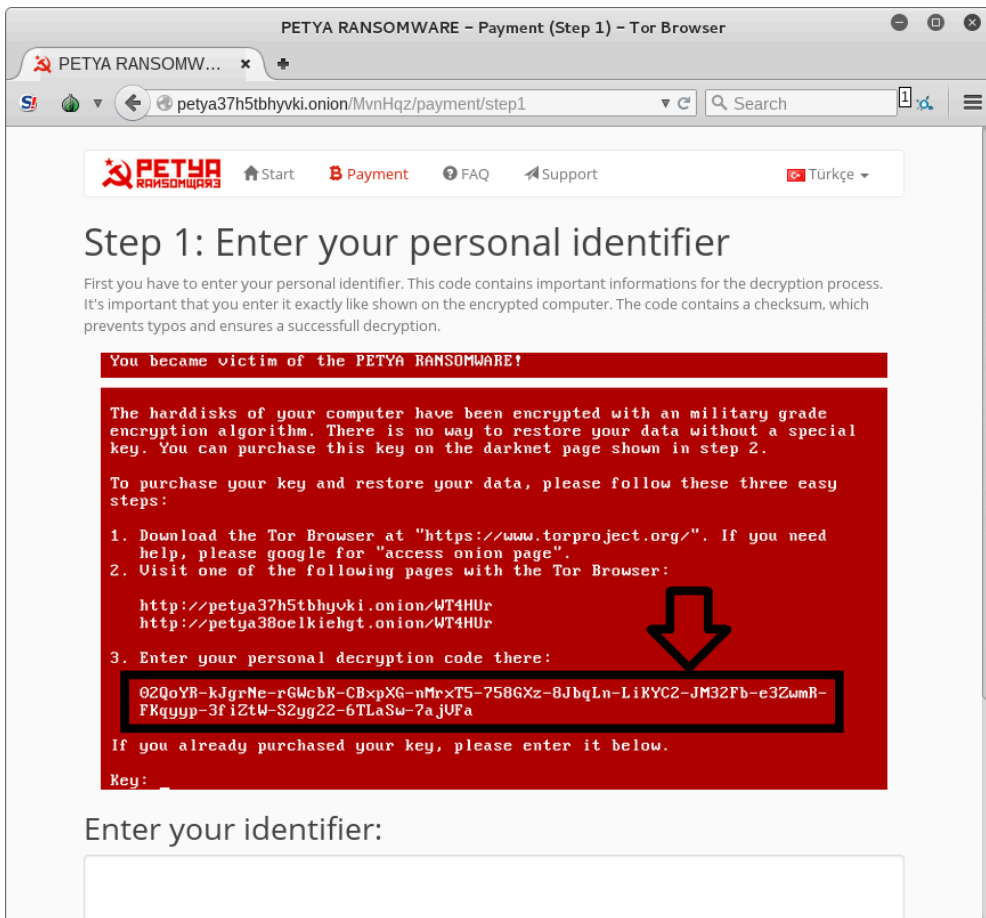
```

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:



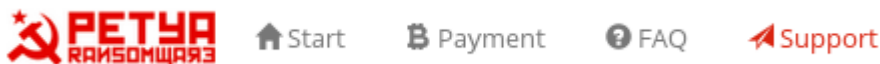
It also provides a step-by-step process on how affected users can recover their data:



- 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:

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



Inside

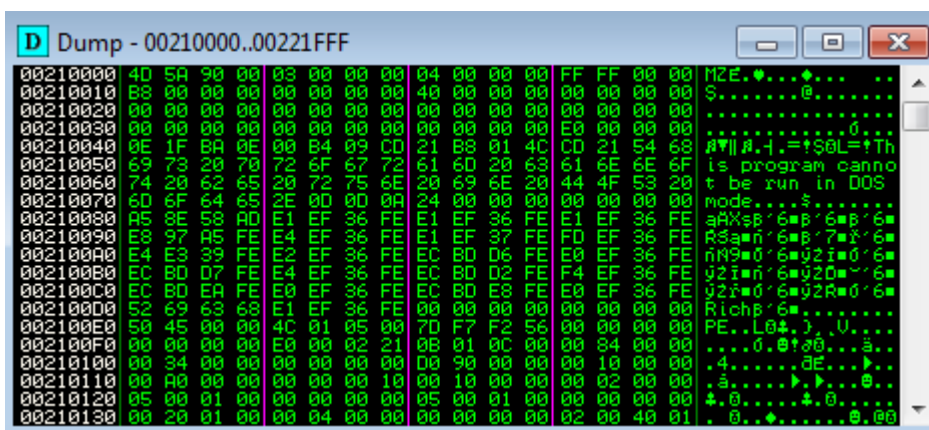
Stage 1

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	TimeStamp	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	

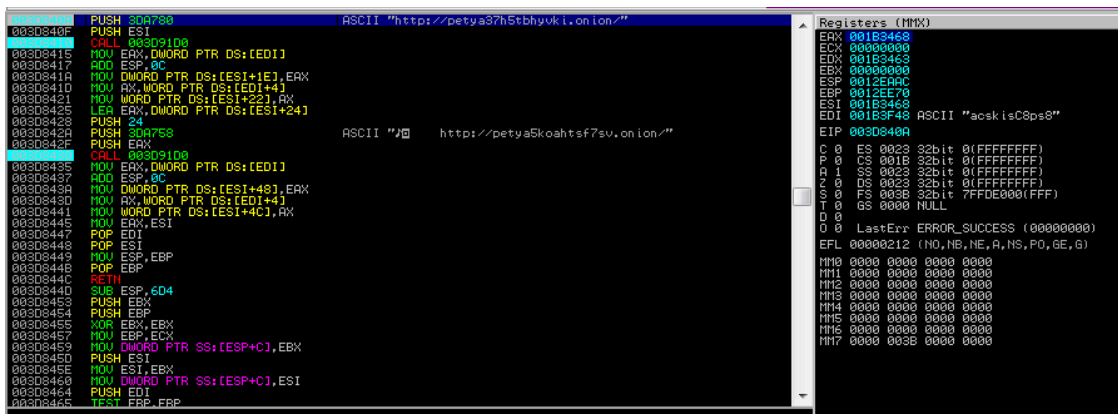
Details				
Offset	Ordinal	Function RVA	Name RVA	Name
A808	1	1DC0	FE1C	_ZuWQdweafdsg345312@0

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:

Making of onion addresses:



Retrieving parameters of the disk using [DeviceIoControl](#)

```

C *G.P.U* - main thread, module Petya
0041280A . PUSH Petya.004318E0 Arg4 = 004318E0
0041280F . PUSH 0x103 Arg3 = 00000103
00412814 . PUSH 0x104 Arg2 = 00000104
00412819 . PUSH ECX Arg1 = 0012FF80
0041281A . CALL Petya.00427469 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 0x80000000
00412837 . PUSH ECX
00412838 . CALL DWORD PTR DS:[<&KERNEL32.CreateFileW] 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 0x5
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] DeviceIoControl
0041288B . TEST EAX,EAX
0041288D . JE Petya.0041298C
    
```

Read/write to the disk:

```

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
00218985 . CALL DWORD PTR DS:[21A030] kernel32.CreateFileA
00218988 . MOV ESI,EAX
00218989 . 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]
00218999 . MOV EAX,DWORD PTR SS:[EBP+C]
0021899D . PUSH EBX
0021899E . SHLD EAX,ECX,9
0021899F . SHLD EAX,ECX,9
    
```

Address	Hex	dump	ASCII
0012F488	37 37 37 37 37 37 37	77777777	
0012F490	37 37 37 37 37 37 37	77777777	
0012F498	37 37 37 37 37 37 37	77777777	
0012F4A0	37 37 37 37 37 37 37	77777777	
0012F4A8	37 37 37 37 37 37 37	77777777	
0012F4B0	37 37 37 37 37 37 37	77777777	
0012F4B8	37 37 37 37 37 37 37	77777777	
0012F4C0	37 37 37 37 37 37 37	77777777	
0012F4C8	37 37 37 37 37 37 37	77777777	
0012F4D0	37 37 37 37 37 37 37	77777777	
0012F4D8	37 37 37 37 37 37 37	77777777	
0012F4E0	37 37 37 37 37 37 37	77777777	
0012F4E8	37 37 37 37 37 37 37	77777777	
0012F4F0	37 37 37 37 37 37 37	77777777	
0012F4F8	37 37 37 37 37 37 37	77777777	
0012F500	37 37 37 37 37 37 37	77777777	
0012F508	37 37 37 37 37 37 37	77777777	
0012F510	37 37 37 37 37 37 37	77777777	
0012F518	37 37 37 37 37 37 37	77777777	
0012F520	37 37 37 37 37 37 37	77777777	
0012F528	37 37 37 37 37 37 37	77777777	
0012F530	37 37 37 37 37 37 37	77777777	
0012F538	37 37 37 37 37 37 37	77777777	
0012F540	37 37 37 37 37 37 37	77777777	
0012F548	37 37 37 37 37 37 37	77777777	
0012F550	37 37 37 37 37 37 37	77777777	
0012F558	37 37 37 37 37 37 37	77777777	
0012F560	37 37 37 37 37 37 37	77777777	
0012F568	37 37 37 37 37 37 37	77777777	
0012F570	37 37 37 37 37 37 37	77777777	
0012F578	37 37 37 37 37 37 37	77777777	
0012F580	37 37 37 37 37 37 37	77777777	
0012F588	37 37 37 37 37 37 37	77777777	
0012F590	37 37 37 37 37 37 37	77777777	
0012F598	37 37 37 37 37 37 37	77777777	
0012F5A0	37 37 37 37 37 37 37	77777777	
0012F5A8	37 37 37 37 37 37 37	77777777	
0012F5B0	37 37 37 37 37 37 37	77777777	
0012F5B8	37 37 37 37 37 37 37	77777777	
0012F5C0	37 37 37 37 37 37 37	77777777	
0012F5C8	37 37 37 37 37 37 37	77777777	
0012F5D0	37 37 37 37 37 37 37	77777777	
0012F5D8	37 37 37 37 37 37 37	77777777	
0012F5E0	37 37 37 37 37 37 37	77777777	
0012F5E8	37 37 37 37 37 37 37	77777777	
0012F5F0	37 37 37 37 37 37 37	77777777	
0012F5F8	37 37 37 37 37 37 37	77777777	
0012F600	37 37 37 37 37 37 37	77777777	
0012F608	37 37 37 37 37 37 37	77777777	
0012F610	37 37 37 37 37 37 37	77777777	
0012F618	37 37 37 37 37 37 37	77777777	
0012F620	37 37 37 37 37 37 37	77777777	
0012F628	37 37 37 37 37 37 37	77777777	
0012F630	37 37 37 37 37 37 37	77777777	
0012F638	37 37 37 37 37 37 37	77777777	
0012F640	37 37 37 37 37 37 37	77777777	
0012F648	37 37 37 37 37 37 37	77777777	
0012F650	37 37 37 37 37 37 37	77777777	
0012F658	37 37 37 37 37 37 37	77777777	
0012F660	37 37 37 37 37 37 37	77777777	
0012F668	37 37 37 37 37 37 37	77777777	
0012F670	37 37 37 37 37 37 37	77777777	
0012F678	37 37 37 37 37 37 37	77777777	
0012F680	37 37 37 37 37 37 37	77777777	
0012F688	37 37 37 37 37 37 37	77777777	
0012F690	37 37 37 37 37 37 37	77777777	
0012F698	37 37 37 37 37 37 37	77777777	
0012F6A0	37 37 37 37 37 37 37	77777777	
0012F6A8	37 37 37 37 37 37 37	77777777	
0012F6B0	37 37 37 37 37 37 37	77777777	
0012F6B8	37 37 37 37 37 37 37	77777777	
0012F6C0	37 37 37 37 37 37 37	77777777	
0012F6C8	37 37 37 37 37 37 37	77777777	
0012F6D0	37 37 37 37 37 37 37	77777777	
0012F6D8	37 37 37 37 37 37 37	77777777	
0012F6E0	37 37 37 37 37 37 37	77777777	
0012F6E8	37 37 37 37 37 37 37	77777777	
0012F6F0	37 37 37 37 37 37 37	77777777	
0012F6F8	37 37 37 37 37 37 37	77777777	
0012F700	37 37 37 37 37 37 37	77777777	
0012F708	37 37 37 37 37 37 37	77777777	
0012F710	37 37 37 37 37 37 37	77777777	
0012F718	37 37 37 37 37 37 37	77777777	
0012F720	37 37 37 37 37 37 37	77777777	
0012F728	37 37 37 37 37 37 37	77777777	
0012F730	37 37 37 37 37 37 37	77777777	
0012F738	37 37 37 37 37 37 37	77777777	
0012F740	37 37 37 37 37 37 37	77777777	
0012F748	37 37 37 37 37 37 37	77777777	
0012F750	37 37 37 37 37 37 37	77777777	
0012F758	37 37 37 37 37 37 37	77777777	
0012F760	37 37 37 37 37 37 37	77777777	
0012F768	37 37 37 37 37 37 37	77777777	
0012F770	37 37 37 37 37 37 37	77777777	
0012F778	37 37 37 37 37 37 37	77777777	
0012F780	37 37 37 37 37 37 37	77777777	
0012F788	37 37 37 37 37 37 37	77777777	
0012F790	37 37 37 37 37 37 37	77777777	
0012F798	37 37 37 37 37 37 37	77777777	
0012F7A0	37 37 37 37 37 37 37	77777777	
0012F7A8	37 37 37 37 37 37 37	77777777	
0012F7B0	37 37 37 37 37 37 37	77777777	
0012F7B8	37 37 37 37 37 37 37	77777777	
0012F7C0	37 37 37 37 37 37 37	77777777	
0012F7C8	37 37 37 37 37 37 37	77777777	
0012F7D0	37 37 37 37 37 37 37	77777777	
0012F7D8	37 37 37 37 37 37 37	77777777	
0012F7E0	37 37 37 37 37 37 37	77777777	
0012F7E8	37 37 37 37 37 37 37	77777777	
0012F7F0	37 37 37 37 37 37 37	77777777	
0012F7F8	37 37 37 37 37 37 37	77777777	
0012F800	37 37 37 37 37 37 37	77777777	
0012F808	37 37 37 37 37 37 37	77777777	
0012F810	37 37 37 37 37 37 37	77777777	
0012F818	37 37 37 37 37 37 37	77777777	
0012F820	37 37 37 37 37 37 37	77777777	
0012F828	37 37 37 37 37 37 37	77777777	
0012F830	37 37 37 37 37 37 37	77777777	
0012F838	37 37 37 37 37 37 37	77777777	
0012F840	37 37 37 37 37 37 37	77777777	
0012F848	37 37 37 37 37 37 37	77777777	
0012F850	37 37 37 37 37 37 37	77777777	
0012F858	37 37 37 37 37 37 37	77777777	
0012F860	37 37 37 37 37 37 37	77777777	
0012F868	37 37 37 37 37 37 37	77777777	
0012F870	37 37 37 37 37 37 37	77777777	
0012F878	37 37 37 37 37 37 37	77777777	
0012F880	37 37 37 37 37 37 37	77777777	
0012F888	37 37 37 37 37 37 37	77777777	
0012F890	37 37 37 37 37 37 37	77777777	
0012F898	37 37 37 37 37 37 37	77777777	
0012F8A0	37 37 37 37 37 37 37	77777777	
0012F8A8	37 37 37 37 37 37 37	77777777	
0012F8B0	37 37 37 37 37 37 37	77777777	
0012F8B8	37 37 37 37 37 37 37	77777777	
0012F8C0	37 37 37 37 37 37 37	77777777	
0012F8C8	37 37 37 37 37 37 37	77777777	
0012F8D0	37 37 37 37 37 37 37	77777777	
0012F8D8	37 37 37 37 37 37 37	77777777	
0012F8E0	37 37 37 37 37 37 37	77777777	
0012F8E8	37 37 37 37 37 37 37	77777777	
0012F8F0	37 37 37 37 37 37 37	77777777	
0012F8F8	37 37 37 37 37 37 37	77777777	
0012F900	37 37 37 37 37 37 37	77777777	
0012F908	37 37 37 37 37 37 37	77777777	
0012F910	37 37 37 37 37 37 37	77777777	
0012F918	37 37 37 37 37 37 37	77777777	
0012F920	37 37 37 37 37 37 37	77777777	
0012F928	37 37 37 37 37 37 37	77777777	
0012F930	37 37 37 37 37 37 37	77777777	
0012F938	37 37 37 37 37 37 37	77777777	
0012F940	37 37 37 37 37 37 37	77777777	
0012F948	37 37 37 37 37 37 37	77777777	
0012F950	37 37 37 37 37 37 37	77777777	
0012F958	37 37 37 37 37 37 37	77777777	
0012F960	37 37 37 37 37 37 37	77777777	
0012F968	37 37 37 37 37 37 37	77777777	
0012F970	37 37 37 37 37 37 37	77777777	
0012F978	37 37 37 37 37 37 37	77777777	
0012F980	37 37 37 37 37 37 37	77777777	
0012F988	37 37 37 37 37 37 37	77777777	
0012F990	37 37 37 37 37 37 37	77777777	
0012F998	37 37 37 37 37 37 37	77777777	
0012FA00	37 37 37 37 37 37 37	77777777	
0012FA08	37 37 37 37 37 37 37	77777777	
0012FA10	37 37 37 37 37 37		

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 7f1RžDžRžRL|ä..
00000001 93 7C 66 B8 20 00 00 00 66 BB 22 00 00 00 B9 00 "|f, ...f»"...a.
00000002 80 E8 14 00 66 48 66 83 F8 00 75 F5 66 A1 00 80 eč..fHf.ž.uóf`.€
00000003 EA 00 80 00 00 F4 EB FD 66 50 66 31 C0 52 56 57 e.€.đéýfPfiRRVW
00000004 66 50 66 53 89 E7 66 50 66 53 06 51 6A 01 6A 10 fPfSžçfPfS.Qj.j.
00000005 89 E6 8A 16 93 7C B4 42 CD 13 89 FC 66 5B 66 58 žčš."|'BÍ.šuf[fX
00000006 73 08 50 30 E4 CD 13 58 EB D6 66 83 C3 01 66 83 s.P0aí.XeÓf.Ā.f.
00000007 D0 00 81 C1 00 02 73 07 8C C2 80 C6 10 8E C2 5F D..Ā..s.SĀeĆ.ZĀ
00000008 5E 5A 66 58 C3 60 B4 0E AC 3C 00 74 04 CD 10 EB ^žfXĀ`.-<.t.í.ě
00000009 F7 61 C3 00 00 00 00 00 00 00 00 00 00 00 00 -aĀ.....
0000000A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000000B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000000C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000000D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000000E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000000F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000011 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000012 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000013 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000014 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000015 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000016 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000017 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000018 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000019 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000001A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000001B 00 00 00 00 00 00 00 00 00 D5 EE 6F 3D 00 00 80 20 .....Óio=.€
0000001C 21 00 07 DF 13 0C 00 08 00 00 00 20 03 00 00 DF !..B.....B
0000001D 14 0C 07 FE FF FF 00 28 03 00 00 D0 1C 03 00 00 ...t`.(...Đ...
0000001E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000001F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000001F0 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA .....UŠ
00000020 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 7777777777777777 Sector 1
000000210 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 7777777777777777
000000220 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 7777777777777777
000000230 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:           ; CODE XREF: seg000:002A↓j
seg000:0021      call   read_sector
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.

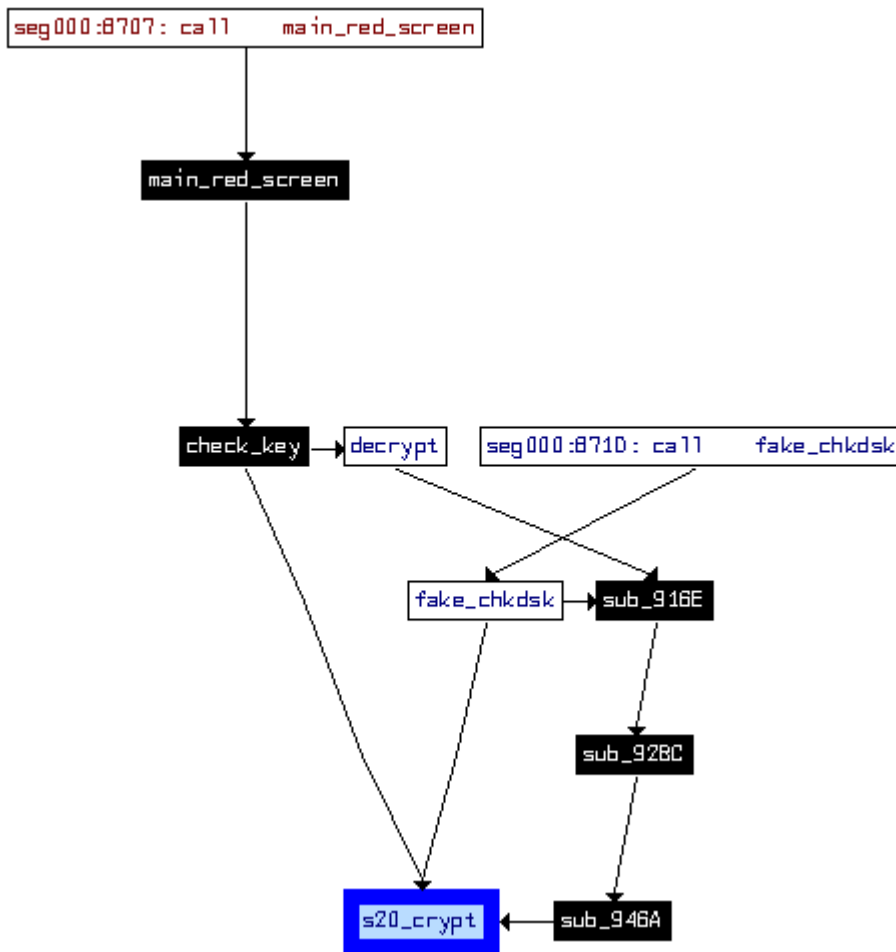
```

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:           ; CODE XREF: seg000:86D7↑j
seg000:86F4      jmp   error
seg000:86F7      ; -----
seg000:86F7      loc_86F7:         ; CODE XREF: seg000:86F2↑j
seg000:86F7      cmp   byte ptr [bp-286h], 1 ; is data encrypted?
seg000:86FC      jb    short to_fake_chkdsk

```

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.

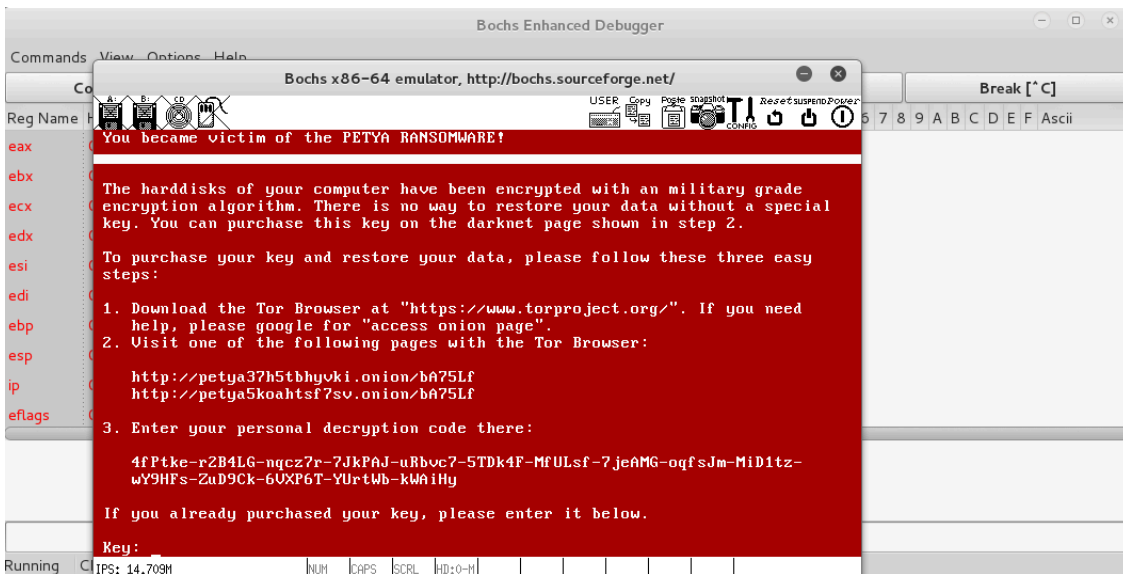
<pre> 00008612 push 73 00008614 lea ax, [bp+key_buf] 00008617 push ax 00008618 call read_input 0000861B add sp, 4 0000861E push ax ; key_len 0000861F lea ax, [bp+key_buf] 00008622 push ax 00008623 mov al, [bp+arg_2] 00008626 push ax 00008627 push si 00008628 call check_key 0000862B add sp, 8 0000862E dec al 00008630 jz short finish </pre>	<pre> 00008632 push 9BFCh ; "Incorrect key..." 00008635 call display_string 00008638 pop bx 00008639 jmp short loc_85F3 </pre>	<pre> 0000863B finish: 0000863B pop si 0000863C pop di 0000863D leave 0000863E retn 0000863E main_red_screen endp 0000863E </pre>
---	--	---

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.
 - Accepted
charset: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ – if the character outside of this charset occurred, it is skipped.
 - 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

- [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.h12
0x00007700	33	34	35	36	37	38	34	33	31	32	32	33	34	10	00		34568784312234.

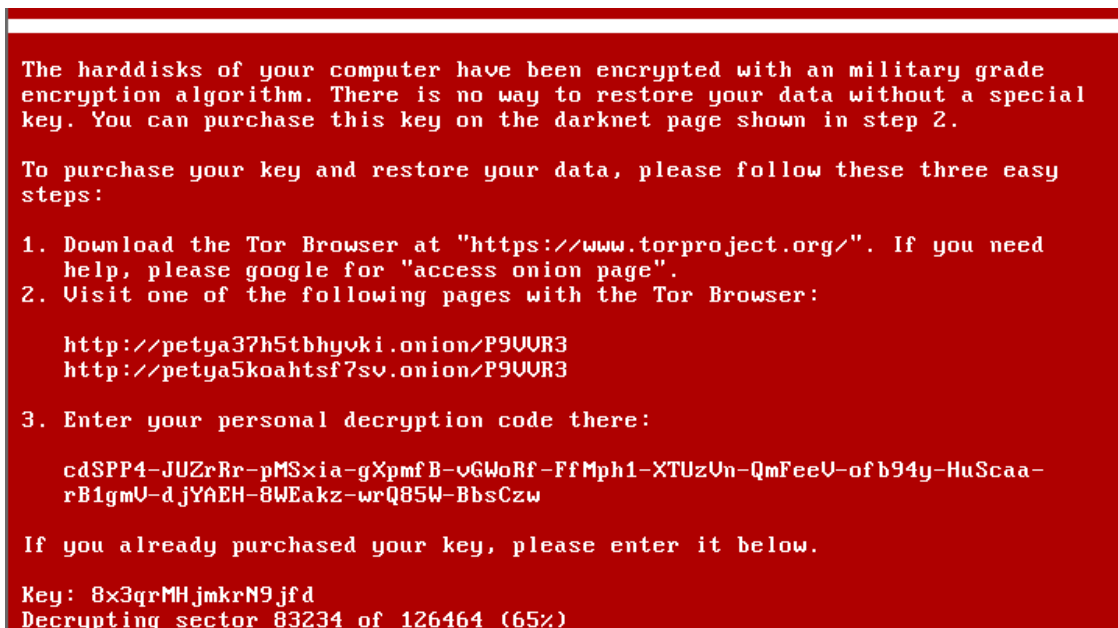
[/CGATAGCLOSE>

The algorithm for encoding the supplied key is very simple:

<https://gist.github.com/hasherezade/785f7da52dfd91fe9e59ae283df2e898>

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.



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:

- <https://blog.gdatasoftware.com/2016/03/28213-ransomware-petya-encrypts-hard-drives> – first article about Petya ransomware (by GData)
- <https://blog.gdatasoftware.com/2016/03/28226-ransomware-petya-a-technical-review> – technical analysis by GData

- <http://blog.trendmicro.com/trendlabs-security-intelligence/petya-crypto-ransomware-overwrites-mbr-lock-users-computers/>
- <http://www.bleepingcomputer.com/news/security/petya-ransomware-skips-the-files-and-encrypts-your-hard-drive-instead/>

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://hshrd.wordpress.com>.

Source: <https://blog.malwarebytes.com/threat-analysis/2016/04/petya-ransomware/>