

# Breaking Petya - Solving a Poor Implementation of Salsa

Peixian Wang  
May 10, 2016

## Abstract

Ransomware has become a relatively profitable development in recent years with the surge in popularity of Bitcoin and other untracable forms of money transactions. In this paper we detail Petya, a recent form of ransomware targeting Windows platforms and NTFS drives. We describe the construction of Petya and the underlying encryption algorithm, Salsa20, and also present one possible solution called Infestor, which utilizes Z3, an efficient satisfiability modulo theory solver, to defeat Petya.

## 1 Introduction

Online transactions through Tor (3) using anonymous cryptocurrencies allow for a certain level of privacy when it comes to payments, but also allow themselves to be used in a malicious fashion. Since Bitcoin, the most popular form of cryptocurrency, makes it extremely hard to track the transactions, Bitcoin has become the de facto standard for ransomware, malware which infects the victim's computer, holds files hostage through encryption, and extorts the victims for money in return for the files. Petya is one such ransomware, except rather than targeting the files of the victim, it targets the master boot record (MBR) and master file table (MFT) (6).

## 2 Petya

### 2.1 Overview

Petya is a relatively new ransomware variant, only starting to appear within the early months of 2016. In order to bypass the lengthy process of encrypting each file on the victim's hard drive, Petya simply seeks to write malicious code to the start of the disk. This code overwrites the MBR of the hard drive with a small kernel that then encrypts the MFT.

### 2.2 Behavioral Analysis

Petya is usually distributed through a zip file (2), containing two other files: 1. a photo of a young man, purporting to be an applicant and 2. an executable, disguised as a csv file, shown in figure 1. After opening the executable, Petya calls an undocumented API called `NtRaiseHardError`. The computer then promptly crashes and boots into a fake CHKDSK scan, shown in figure 2, which starts

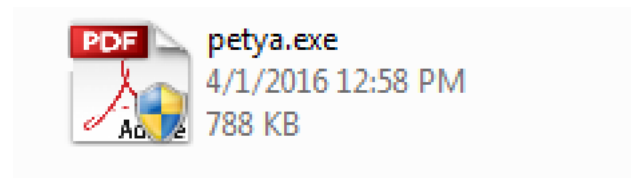


Figure 1: The petya executable disguised as a .pdf file.

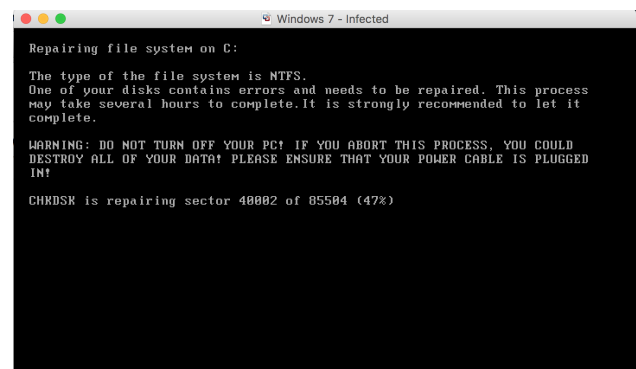


Figure 2: The fake CHKDSK scan made by Petya while it encrypts the MFT.

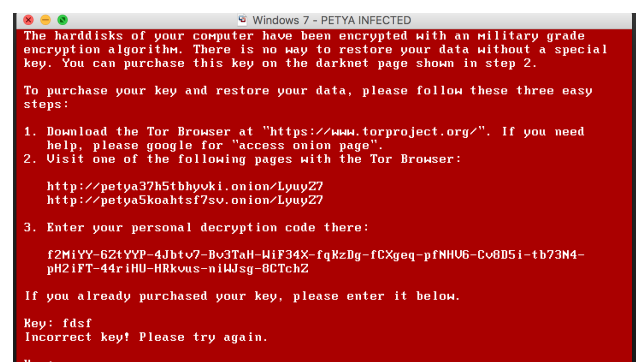


Figure 3: The ransom note by Petya.

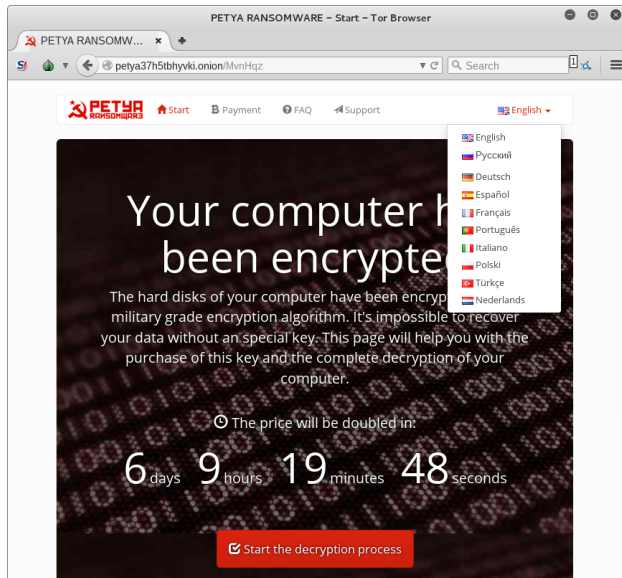


Figure 4: Accessibility is important, even for malware. From (2).

the encryption on the MFT. When the encryption completes, the user is shown a ransom note screen, shown in figure 3. After visiting the website, the user is presented with a relatively upscale website, featuring multiple languages (figure 4) and a tutorial on how victims can perform a bitcoin transaction (figure 5).

## 2.3 Code Analysis

The Petya attack can be detailed into 3 stages, the MBR attack stage, the MFT attack stage, and the ransom demand stage.

### 2.3.1 Stage 0 - MBR attack

In order to start analysis on Petya, the Windows executable must be examined. The executable generates a 8-byte initialization vector and a unique 16-byte random key that is used for further encryption, which also must be known to attackers to decrypt the files (2). This key is expanded into the 32-byte encryption key using figure 7. Both of these values are used later on in the encryption process (6).

Petya then encrypts the original MBR by XOR'ing the contents of the MBR with 0x37 (4). The encrypted value is then saved to sector 56 in the disk (5). Disk sector 1-34 is encrypted in the same method. Petya then proceeds to write its own kernel, creating an `ONION_SECTOR` structure and writing it to sector 54, shown in figure 6. This

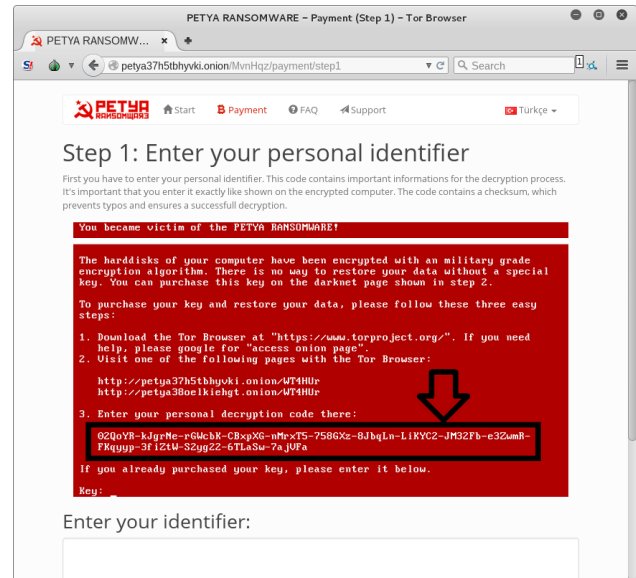


Figure 5: Petya Tutorial on purchasing bitcoins and performing a transaction. From (2).

```
00000000 ONION_SECTOR struct ; (sizeof=0x200)
00000000 eEncrypted db ? ; is drive encrypted flag, enum { NotEncrypted = 0,
; Encrypted = 1, Decrypted = 2 }
00000001 key db 32 dup(?) ; ENCRYPTION KEY
00000021 iv db 8 dup(?) ; IV (stays permanent in the system)
00000029 szURLs db 128 dup(?) ; onion links
000000A9 szPubKey db 343 dup(?) ; public key string, will be displayed at ransom page
00000200 ONION_SECTOR ends
```

Figure 6: The `ONION_SECTOR` struct, from (6).

data conveniently contains 8 byte `iv`, which is the initialization vector required later on to decrypt. Sector 55 is then filled with the byte 0x37, which is checked after the decryption process for validation purposes.

Finally, the `NtRaiseHardError` is called, crashing the victim's system.

### 2.3.2 Stage 1

The system drive is queried by the Petya kernel, and upon returning a successful value, Petya begins to overwrite the MFT while masquerading as a `CHKDSK` (5).

This stage can be divided into two steps: reading the key and deletion of the key, and encrypting sector 55.

In the first step, Petya reads sector 54 into memory, which currently contains the `ONION_SECTOR`. It copies the first 32 bytes of the `ONION_SECTOR` into memory, which is the encryption key, and then proceeds to zero the first 32 bytes of data. Afterwards, the 32-byte key is only present in memory.

The second step consists of Petya encrypting sec-

```

void key_expand(char key[16], char outKey[32])
{
    for (int i = 0; i < 16; ++i) {
        unsigned char uc = key[i];
        outKey[i * 2 + 0] = uc + 0x7A; // uc + "z"
    }
}

```

Figure 7: The key expansion method called by Petya. From (6).

tor 55, which is currently filled with all byte values of 0x37. Here Petya uses a flawed version of Salsa20 (1), passing in the 32-byte encryption key and the 8-byte initialization vector, to encrypt sector 55, which we will detail in section 3.

### 2.3.3 Stage 2

Finally, Petya simply reboots into the red flashing skull screen, displaying the ransom note and providing an entry for the decryption key. By running the infected disk through a debugger, we can follow the verification process of the key:

1. Read sector 54 to read the 8 byte initialization vector.
2. Read sector 55.
3. Read the user entered key, extract the first 16 bytes, and use the same expansion function as 7 in order to retrieve a 32-byte key.
4. Use the 32 byte key and the 8 byte initialization vector to decrypt sector 55, if all the values equal 0x37, then the user entered key is correct. Else throw an **Invalid Key!**.
5. If the user entered key is correct, proceed with the decryption process and ask user to restart their computer.

From this verification process, we can begin to construct a list of constraints the key must follow:

1. Only the first 16 bytes are extracted from the user entry, meaning the key must be at most 16 bytes long from the character set [a-zA-Z0-9], i.e. all alphanumeric characters.
2. The key must enable all data in sector 55 to be able to xor with 0x37.

3. The initialization vector must be used shuffled in somehow.

Thus, in order to break Petya, we must start by breaking its encryption algorithm: Salsa.

## 3 Salsa

### 3.1 Salsa20 Spec

Salsa20, at its core, is a function that takes a 64-byte string and produces a 64-byte string. The function follows an add-rotate-xor operation, whose operation can be found in exact detail in the Salsa20 spec (1), but the original implementation uses a 32-byte encryption key and an 8-byte initialization vector to produce a final 515-bit key-stream, shown in figure 8.

To create the output, the 64 byte input into Salsa is seen in little-endian form as 16 words, which are fed into 320 invertible modifications, where each modification changes one words. The resulting 16 words are added to the original input respectively mod  $2^{32}$ , which then produces the output in little-endian form. The exact modification is pre-defined in the Salsa spec and implementations (?), which is a series of 10 identical double-rounds, each consisting of 4 parallel quarter-rounds, with each quarter-round modifying 4 words (?).

## 4 Infestor

### 4.1 Code Walkthrough

## References

- [1] BERNSTEIN, D. J. Salsa20 specification. <https://cr.yp.to/snuffle/spec.pdf>.
- [2] HASHEREZADE. Petya – taking ransomware to the low level. <https://blog>.

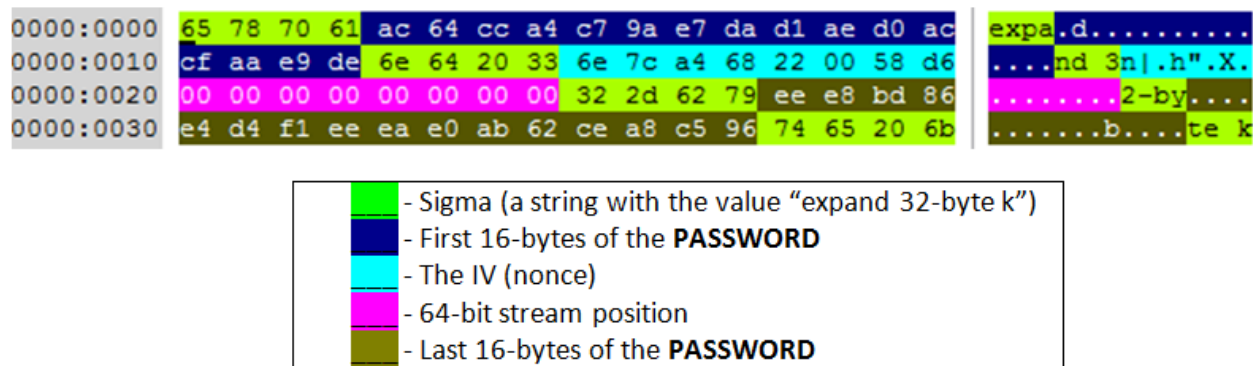


Figure 8: The original Salsa implementation. From (6).

malwarebytes.org/threat-analysis/2016/  
04/petya-ransomware/, 2016.

- [3] PROJECT, T. Tor project. <https://www.torproject.org/>.
- [4] TONELLO, G. Breaking petya ransomware! [http://www.tgsoft.it/english/news\\_archivio\\_eng.asp?id=718](http://www.tgsoft.it/english/news_archivio_eng.asp?id=718), 2016.
- [5] TONELLO, G. Petya ransomware x-rayed !!! [http://www.tgsoft.it/english/news\\_archivio\\_eng.asp?id=712718](http://www.tgsoft.it/english/news_archivio_eng.asp?id=712718), 2016.
- [6] TRAFIMCHUK, A. Decrypting the petya ransomware. <http://blog.checkpoint.com/2016/04/11/decrypting-the-petya-ransomware/>, 2016.