Petya Ransomware Goes Low Level

Petya Ransomware Analysis and Full Details on Bitdefender's Vaccine Preventing File Encryption

Bitdefender

Contents

Authors

Razvan Benchea - Team Leader Malware Research
Vatamanu Cristina - Malware Researcher
Maximciuc Alexandru - Malware Researcher
Liviu Arsene - Senior E-Threat Analyst

Executive Summary

The new Petya ransomware seems to have been built with speed in mind, as to expedite the encryption process. While traditional ransomware encrypts files one by one, Petya encrypts the location containing all information about disk files, such as size, permissions, and data content, essentially preventing users from accessing all their data.

While this low level approach allows for the same file-restricting outcome as traditional ransomware, it's a lot faster in terms of encryption time, as the ransomware no longer has to encrypt each file at a time. Instead, it only encrypts the NTFS MFT (Master File Table). However, this approach makes cryptography extremely difficult to implement, increasing the probability of making mistakes in the cryptographic functions.

Bitdefender was able to analyze the Petya ransomware and offer potential victims a tool that intercepts the encryption process and offers the decryption key, free of charge. Most importantly, the tool needs to be installed prior to being infected - not afterwards – in order to perform its function correctly.

Key Findings

- Potentially same developers as the ones behind Chimera and Rokku ransomware families;
- Works faster - Petya doesn't encrypt files; it encrypts the NTFS Master File Table (MFT);
- Features its own bootloader and Kernel – few ransomware families have that;
- Reverse engineered by Bitdefender to offer a free tool that assists in decryption of NFTS MFT (third-party tools have become available, but they're more difficult to use).

What's the big deal with Ransomware?

Ransomware incidents inflicted an estimated $24.1 million[1] in financial losses in 2015, according to 2,453 reported hackings. With 13 million U.S. users encountering some form of ransomware, our own studies have shown that 50 percent[2] of them would actually pay to regain access to their data.

Difficult to detect and even more difficult to crack it's encryption on users' files, ransomware has made its way beyond the Windows operating system, hitting Linux, Mac and even Android[3]. While its behavior varies widely – from simply displaying a lock screen to encrypting all stored files – the ultimate goal is to force users to pay ransom, in bitcoins.

Motivated by financial gains and with advanced technical skills, cybercriminals have constantly "innovated" in terms of new ways of encryption, delivery mechanisms, and even locking out users without encrypting the actual files on their disks, as you'll soon learn from this paper detailing how the new Petya ransomware works.

Ransomware is one of the nastiest forms of malware to date, not only because it generates immense revenue streams for cybercriminals, but also because the encryption it uses to deny users access to their own files is usually impossible to crack. Although some variants have exhibited weaknesses[4] in the encryption process, sometimes even allowing security experts to develop vaccines[5] for particular infections, ransomware remains

1 http://www.usnews.com/news/business/articles/2016-04-05/5-ways-to-become-a-smaller-target-for-ransomware-hackers
2 http://download.bitdefender.com/resources/files/News/CaseStudies/study/59/Bitdefender-Ransomware-A-Victim-Perspective.pdf
3 http://download.bitdefender.com/resources/files/News/CaseStudies/study/85/Android-Malware-Threat-Report-H2-2015.pdf
4 https://labs.bitdefender.com/2016/01/third-iteration-of-linux-ransomware-still-not-ready-for-prime-time/
5 https://labs.bitdefender.com/2016/03/combination-crypto-ransomware-vaccine-released/

particularly difficult to thwart.

The usual modus operandi for ransomware remains constantly the same. The infection vector is usually comprised either of malicious ads disseminated trough legitimate websites or infected attachments promising urgent invoices or even hiring CVs – as was the case with Petya.

To this end, ransomware relies on some form of user interaction with the dropper (a.k.a. the user-presented file), socially engineering users into single handedly executing the infected file and getting the ransomware. A carefully crafted spearfishing email will have a significantly high success rate, guaranteeing sufficient victims to make it worthwhile for cybercriminals.

The following steps are usually the same from one ransomware variant or family to another, and usually revolve around something like this:

- The dropper downloads a payload that's executed locally;
- Connects to a C&C and uses a public key to encrypt local data (traditionally);
- Displays a message asking the victim to pay a ransom;
- Once payment is successful, contacts the C&C server and gets the decryption key;

While the entire process seems pretty straightforward, there's **no actual guarantee that the victim will ever get the decryption key** – although the cybercriminals have been honest in most cases – nor that they're actually telling the truth about encrypting the data.

In some instances, cybercriminals only displayed a message – granted, difficult to remove – on screen, trying to scare victims into paying and leverage past ransomware experiences that have plagued the internet. While no actual data is encrypted, some users do end up paying cybercriminals anyway.

A major issue with ransomware is that it's extremely easy to get your hands on – for a mere $3,000 you can get your own ransomware kit[6] - and it's extremely profitable, with an estimated[7] r**eturn on investment of 1,425%.** Put that on the dark web and you'll find that dozens of non-tech-savvy individuals – but with an entrepreneurial sense of making money – will be willing to get their hands on it and start making money.

How Petya Ransomware Works

Petya usually arrives via email containing a Dropbox URL or an attachment, and the executable usually differs from one dropper to another. It mostly uses clean executables - ***java update checker 2.8.73.2; jucheck.exe, picasa 3.9.141.259, google crash handles 1.3.29.5, SumatraPDF Installer 3.1.1, and installers in Inno Setup format*** - that were infected with Petya.

The malicious code appears to be inserted manually, as it's present at different offsets. While the entry-point remains unchanged, the executable code that includes the decrypter, the loader, and the Petya DLL is placed at different offsets within the code. While we didn't find any executable that has the Petya code inserted at the beginning, it's probably because the malware developer wanted to avoid being detected by security software that scans network traffic. It's safe to assume these executables have been created manually.

Another interesting aspect of these executables is that the original icon has been changed and replaced with a PDF or WinRar icon.

After the executable is loaded by the dropper in user mode, the malware tries to decrypt the code that will be used for the bootloader. It does this by first searching the section with the name ".xxxx". After this, it starts decrypting it with a simple algorithm.

6 http://www.hotforsecurity.com/blog/cryptolockercryptowall-ransomware-kit-sold-for-3000-source-code-included-13020.html

7 http://www.darkreading.com/analytics/cybercrime-can-give-attackers-1425--return-on-investment/d/d-id/1320756

```
key = 0x6b81;
for (int i = 0; < bufferSize; i++)
      buffer[i] ^= key%(i + 1);
```

Once decryption is complete, it gathers information about the hard disk. It finds the partition where the operating system is installed and opens it in raw mode.

It detects the type of the partition table (GPT or MBR) and if ii is GPT, it encrypts the backup of the partition header.

It then continues to encrypt the first sector, by doing a xor '7' operation on each byte. Following this, it continues to generate the encryption key according to the algorithm presented below. Once this is over, it overwrites the first sector with the code decrypted from xxx section (the one that will do the salsa encryption) and it encrypts 33 sectors starting with the second one.

The primary encryption process finishes with the copying of the encryption key to sector54, with the overwrite of sector55 with character '7' and with the saving of the original MBR (xor-ed with '7') to sector56.

To shut down the system, it first tries to acquire SeShutdownPrivilege and, if it succeeds, it calls the function NtRaiseHardError with an argument that causes the system to display the BSOD.

Petya is somewhat different from what we've been used to with Cryptowall, Cryptolocker or other ransomware families that we've been hearing about for the past couple of years. The biggest difference is that it doesn't encrypt files per se, but **encrypts the filesystem's master file table (MFT)** responsible for telling the operating system where all disk files are located. Consequently, the entire encryption process takes far less time, as it doesn't have to encrypt each file, and uses algorithms that should make the process irreversible unless you have the decryption key.

Executing the Petya payload **triggers the user account control (UAC)**, without the ransomware even trying to bypass it. Socially engineering users into clicking "Yes" without acknowledging the risks associated with running unknown applications is not a new practice, and still seems to work.

Afterwards, it **overwrites the master boot record (MBR) with Petya's bootloader**, makes an encrypted backup of the data, then **crashes the operating system with a BSOD** (blue screen of death). During this stage the **data can actually be recovered** – provided you have ***turned off automatic restart after a system failure***. That's because only the beginning of the disk (MBR) has been overwritten. By "overwritten" we in fact mean that the first 33 sectors (starting with the second), along with the first 33 sectors where the MBR backup is located, get encrypted using a simple XOR '7'. It then places its encryption key in sector 54 and then start filling sector 55 with '7's. Basically, even multiple modifications can all be resolved. Mounting the disk on another operating system and making a backup is possible at this stage.

However, if the system boots up after the BSOD, it displays **a fake CHKDSK scan** that's actually **encrypting the MFT using a Salsa10[8] algorithm**. At this point, it is theoretically possible to bruteforce the decryption and read the contents of the disk.

The MFT is responsible for maintaining entries – including size, time and date stamps, permissions and data - for every file on the NTFS file system volume, including the MFT itself. Consequently, having an encrypted MFT means there's no way to access those files without the decryption key.

Similarities with other Ransomware Families

While Petya is clearly more complex than other ransomware, it **shares some similarities with** two of them in particular, namely **Chimera and Rokku.** This could indicate that the same attacker might be behind all of

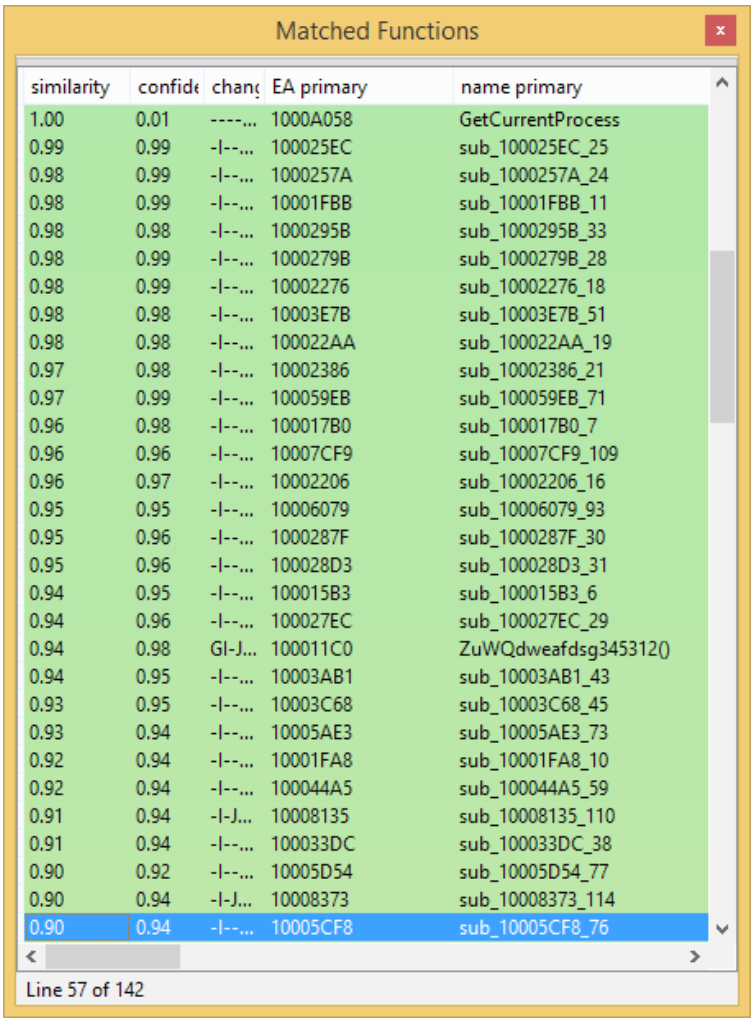
8 https://en.wikipedia.org/wiki/Salsa20

them, or maybe a common framework was used for all three. When compared to Chimera, the distribution method is very similar: both Petya and Chimera were distributed via spam campaigns and both contained a link to a file uploaded via Dropbox. More interestingly, both families were distributed via spam messages written in German. Correlations can be found even when looking at the file name. Both malware families come in binaries that have double extension or end with PDF, and in some cases the file name is the same (e.g. “BewerbungsmappePDF.exe”).

More information can be found when looking inside the binaries themselves, as Petya and Chimera seem to use the same dropper. In both cases, the main component of the malware is located inside a dll file, which has only one exported function. The dll file is further loaded by another component located inside the dropper. What stands out is that the loader and the main component are encrypted and inserted into the code section of a known and benign executable file at an arbitrary offset.

The encryption algorithms used by Chimera are again similar to the ones used by Petya, but with some differences. Chimera also relies on CBC and CTR variations of the AES algorithms, and it also uses an elliptic curve for the public key encryption. However, while it uses the secp256k1 elliptic curve instead of secp192k1, it doesn’t use the Salsa20 algorithm. It is also worth noting that both families implement encryption algorithms and don’t rely on Windows API functions.

Following these observation, we compared the two main binaries to see how many of their functions were similar. Using BinDiff[9], we determined that **57 functions have a similarity of over 90%** and **28 functions are identical**. More so, the BinDiff plugin reports that the **two binaries have a similarity of 42.7%** with a confidence of 87.37%. A printscreen that shows similar functions can be seen below.



9 https://www.zynamics.com/bindiff.html

Among the similar functions found are some that generate 128 random bytes using CryptGenRandom, and one that initializes only one structure out of 20 allocated, for generating random bytes based on the sha512 algorithm.

Analysts from Malwarebytes[10] reported that Rokku ransomware also shows similar features to the chimera ransomware. This claim is also supported by other evidence. Like Petya and Chimera, Rokku uses the same methods of loading the main dll file and the malware component. The malicious content is also embedded into the code section of a benign file at an arbitrary offset.

Rokku, like Petya and Kimera, uses an elliptic curve algorithm to generate the key for the public key encryption algorithm. What is more interesting is that Rokku, like Petya, uses the same algorithm for encryption.

Reverse Engineering Petya Ransomware

This is where it all gets technical. In the following section we will dive into the malware's code and understand each step of the encryption process and how cryptography works.

Before we start, it's worth mentioning that an interesting aspect of Petya is the number of algorithms it uses to generate a random key. Besides the fact that it generates random bytes using a safe Windows function, it also calls for various other algorithms, such as variations on SHA512 and AES variations with different keys, to individually generate a random character. While the process is somehow impractical as long as it starts from random bytes, it's still worth mentioning as it is specific to this malware family.

Generating Salsa20 key

The first step in obtaining the Salsa20 encryption key is to generate a unique password.
The password is a 16 bytes buffer containing characters from the following set *'123456789abcdefghijkmnopqrstuvwxABCDEFGHJKLMNPQRSTUVWX'.*

The selected character is chosen based on a generated DWORD:

```
do
{
  getDwordForKey((int *)&outDWORD, (encryption_info *)encryptionInfo, 4u);
  currentChar = outDWORD / 54;
  LOBYTE(currentChar) = characterSet[outDWORD % 54];
  outputKey[idx++] = currentChar;
}
while ( idx < 16 );
```

To obtain these DWORDs the following steps are executed:

1. *Generate buffer of 0x80 random bytes:*

```
int __cdecl fnGenerateRandomNumbers(int a1, BYTE *pbBuffer, DWORD dwLen, DWORD *cryptContext)
{
  DWORD *v4; // edi@1

  v4 = cryptContext;
  *cryptContext = 0;
  if ( !CryptAcquireContextA((HCRYPTPROV *)&cryptContext, 0, 0, PROV_RSA_FULL, CRYPT_VERIFYCONTEXT) )
    return -60;
  if ( !CryptGenRandom((HCRYPTPROV)cryptContext, dwLen, pbBuffer) )
    return -60;
  CryptReleaseContext((HCRYPTPROV)cryptContext, 0);
  *v4 = dwLen;
  return 0;
}
```

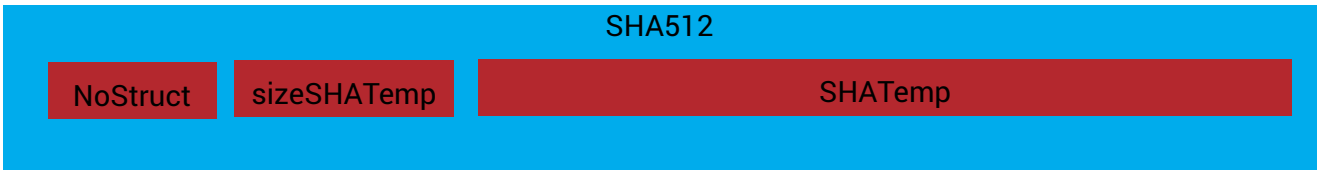
2. *Obtain the shaKey*

10 https://blog.malwarebytes.org/threat-analysis/2016/04/rokku-ransomware/

It will compute a sha512 hash on this buffer:

```
if ( size_buffer_random > 64 )
{
  // sha512
  compute_sha(size_buffer_random, buffer_random, (BYTE *)sha_hash_temp, 0);
```

This resulted hash is part of a buffer used to obtain a second hash . The format of the buffer is as following:

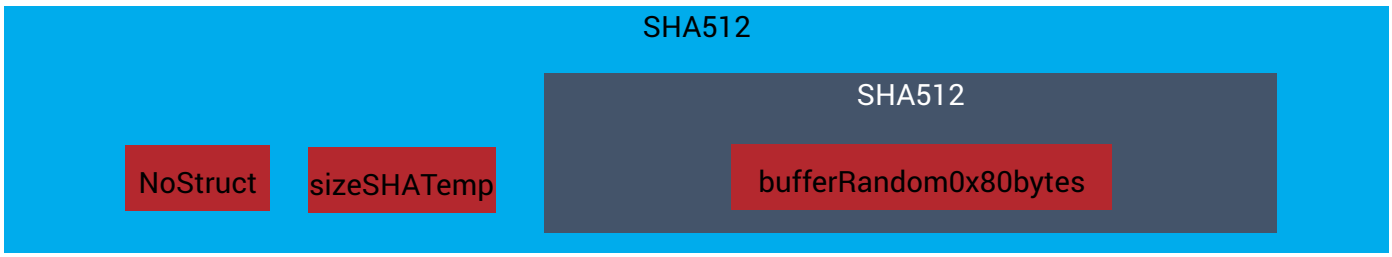


The second temp hash wold be a SHA512 on a buffer containing:

- NoStruct: the number of the used structure (value 0)
- sizeSHATemp: (value 64)
- SHATemp: the SHA hash computed on the buffer containing the 0x80 random bytes

The final SHA hash value is SHA512 function on a buffer represented by the last sha hash. So to summarize:

shaFinalHash = SHA512(SHA512(makeBuffer(0,64,SHA512(buffer_of_0x80_random_bytes)))



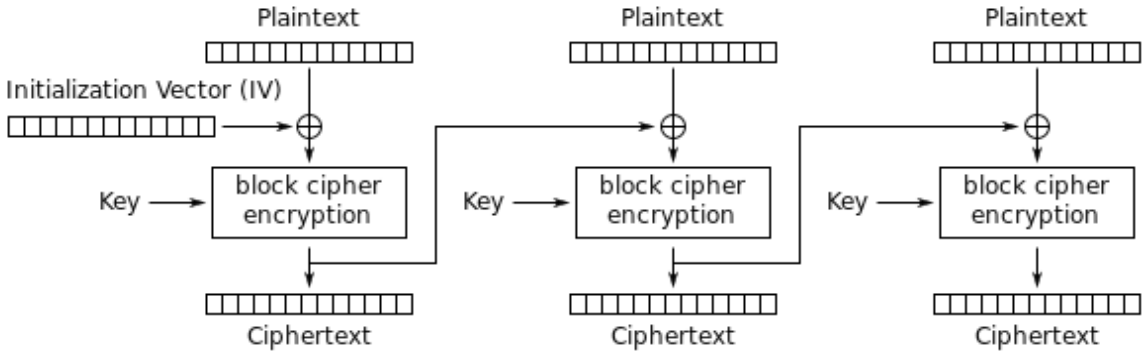
The first stage of the key is represented by the first 0x30 bytes of the shaFinalHash

```
memory_copy((int)shaKey, (BYTE *)shaFinalHash, sizeKey_0x30);
```

The shaKey is used next in the salsa key generation.

2.Encrypt the shaKey

First an AES-CBC is used:

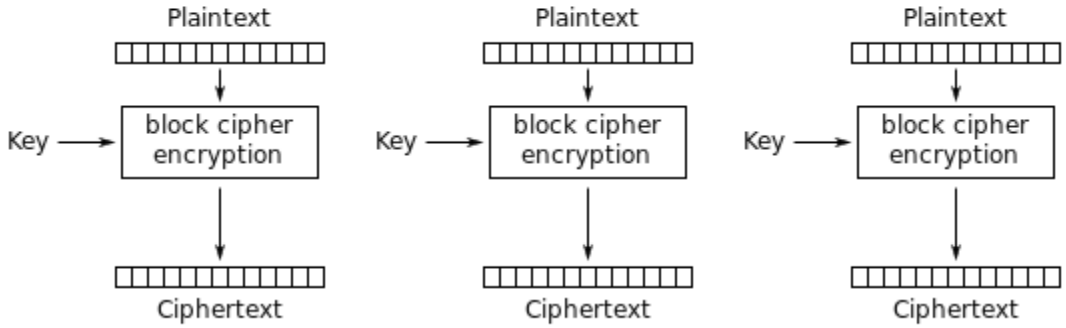


Cipher Block Chaining (CBC) mode encryption

Source: Wikipedia[11]

The plain text is a buffer that, among other elements, contains the shaKey. The initial IV is a zero bytes buffer and the initial key is a buffer containing the values from 0 to 31.

The resulted cipherText is used as a initial key in a AES-ECB algorithm



Electronic Codebook (ECB) mode encryption

Source: Wikipedia[12]

The plainText in this case is the second 16 bytes block from the cipherText.

3. *Update the main AES key*

The 16 iterations (for each character of the password) share another AES context; lets call it main AES context. After the AES-ECB is executed, the resulted cipherText is used to update the main AES key through a XOR opeation.

```
do
{
  updatedAESKey[idx] ^= updatedAESKey[idx + shaKey - updatedAESKey];
  ++idx;
}
while ( idx < 48 );
rijndael_keySetupEnc(&contextStruct->mainAesContext, (BYTE *)updatedAESKey);
result = 0;
*(_DWORD *)&contextStruct->bufferToEncrypt[0] = *(_DWORD *)&updatedAESKey[32];
*(_DWORD *)&contextStruct->bufferToEncrypt[4] = *(_DWORD *)&updatedAESKey[36];
*(_DWORD *)&contextStruct->bufferToEncrypt[8] = *(_DWORD *)&updatedAESKey[40];
*(_DWORD *)&contextStruct->bufferToEncrypt[12] = *(_DWORD *)&updatedAESKey[44];
```

The buffer used to encrypt at the next iteration is represented by the last 16 bytes of the updated key.

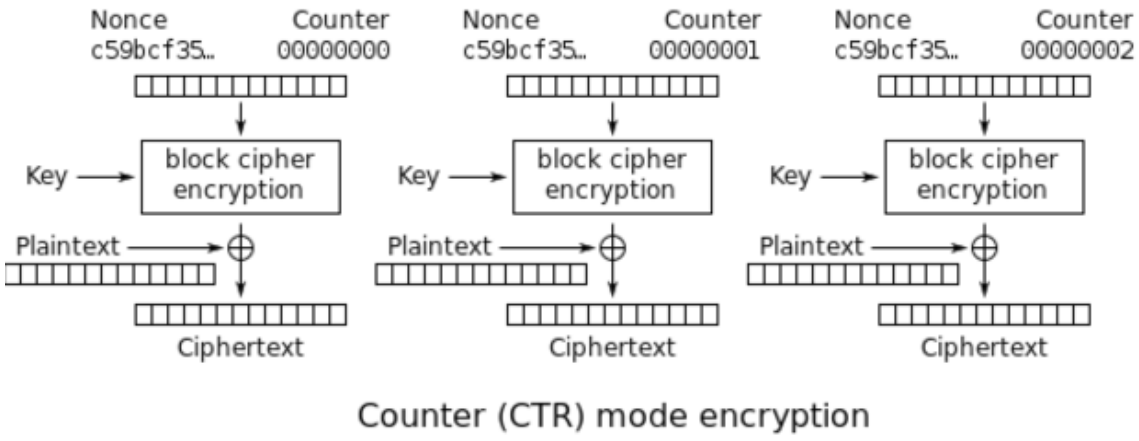
Each dword used to obtain the characters from the salsa key is obtain by encrypting one more time using the main AES context:

```
AesEncryptBloc((BYTE *)mainContextStruct, mainCtx, destinatie);

memory_copy((int)outDWORD, (BYTE *)destinatie, sizeDWORD);
```

The main AES context is applied in an AES-CTR algorithm.

11 https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

12 https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Electronic_Codebook_.28ECB.29



Counter (CTR) mode encryption

Source: Wikipedia[13]

In summary, to obtain the password used to generate the salsa key, the following steps are done:

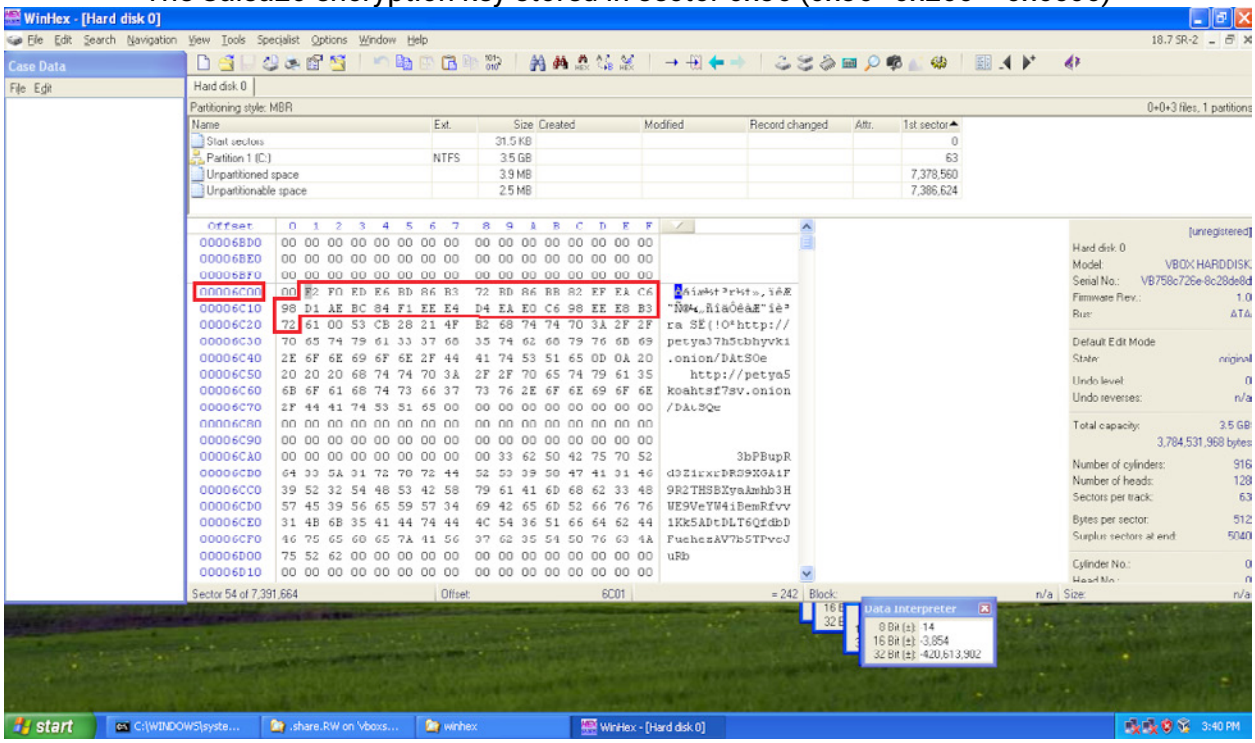
```
for (i=0;i<16;i++){
      //generate the random buffer
      randomBuffer = generateRandomBuffer()

      //get the SHA final key
      shaKey = getShaKey(randomBuffer)

      // encrypt the key
      //AES-TYPE(plainText, context, cipherText)
      aes-cbc-context = initializeContext(Aes-cbc-key)
      AES-CBC(shaKey, aes-cbc-context, cipherText_AES-CBC)
      aes-ecb-context = initializeContext(cipherText_AES-CBC
      AES-ECB(plainText, aes-ecb-context, cipherText_AES-ECB)

      //update the main AES key
      AES-CTR(main-buffer-to-encrypt, main-context, cipherText_AES-CTR)
      cipherText_AES-CTR ^= cipherText_AES-ECB
      main-buffer-to-encrypt = cipherText_AES-CTR[-16:]



      //get dword
      AES-CTR(main-buffer-to-encrypt, main-context, cipherText_AES-CTR)
      Copy(currentDword, cipherText_AES-CTR)


      //update the main AES key
      AES-CTR(main-buffer-to-encrypt, main-context, cipherText_AES-CTR)
      cipherText_AES-CTR ^= cipherText_AES-ECB
      main-buffer-to-encrypt = cipherText_AES-CTR[-16:]

      //get char for password
      currentChar = base54Alphabet[currentDword % 54]
}
```

After generating the password, the 32 bytes salsa encryption key is obtained and stored on sector 0x36 (16->32):

13 https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Counter_.28CTR.29

```
memreset(sector_0x36, 0, 512u);
password = GeneratePassword();
PersonalDecryptionCode = (BYTE *)GeneratePersonalDecriptionCode();
torProjectURL = GenerateUrls();
contor = 0;
do
{
  sector_0x36[2 * contor + 1] = password[contor] + 'z';
  sector_0x36[2 * contor + 2] = 2 * password[contor];
  ++contor;
}
while ( contor < 0x10 );
```

The Salsa20 encryption key stored in sector 0x36 (0x36 *0x200 = 0x6c00)



Following the algorithm, the password would be -> ***xsC9CAuLWBwjpLt9***

The structure for the 0x36 sector is as follows:

- The first byte can be 0 (not encrypted, hence it can be encrypted), 1 (it has been encrypted), and 2 (it has been decrypted)
- The next 32 bytes are the key.

It's important to note that the key, although it has 32 bytes, is derived from a 16 byte key. The key will be overwritten with 0 as soon as it start encrypting, meaning the key will be there only if the first byte is 0. Immediately afterwards is an .onion link where the user has to input the decryption key.

At the 169 offset there's the "Personal Decryption Code," that's actually the decryption key, encrypted with the secp192k1 elliptic curve.

After reboot following Salsa20 encryption:

Using the password, the decryption is done.

```
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/DAtSQe
   http://petya5koahtsf7sv.onion/DAtSQe

3. Enter your personal decryption code there:

   3bPBup-Rd3Z1r-xrDRS9-XGA1F9-R2THSB-XyaAmh-b3HWE9-VeYW4i-BemRfv-v1Kk5A-
   DtDLT6-QfdbDF-uehezA-V7b5TP-vcJuRb

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

Key: xsC9CAuLWBwjpLt9
Decrypting sector 25178 of 39136 (64%)
```

After it finishes, it displays a message prompting the user to restart his machine.

How the Decryption Key is Transmitted

Since the malware performs the encryption in an environment with no internet connection, it has no way of transmitting it to the attacker. To receive the decryption key, the victim has to go to the tor address displayed by the ransomware and enter the Personal Decryption Code.

The Personal Decryption code is actually the salsa20 encryption key encrypted with public key cryptographic system. The steps taken to construct the Personal Decryption Code are as follows:

First, a pair of public and private keys is generated for the infected victim using the secp192k1 elliptic curve.

We will call these PubKeyV and PrivKeyV. The attacker also has a pair, but only its public key is known. We will call the public key for the attacker PubKeyA , while the private is called PrivKeyA.

PubKeyA is hardcoded in the binary:

```
PubKeyA=04C480AF982B11269CB438A01C4679A8329B5A5F4E800C869EA3D52677F3261EC88DD171ECA5A9066F4D8F26
DCA648FEF9
```

The next step is to generate a key that will be used in encryption. This is generated based on a value (called shared secret or ECDHE) that each part can compute based on its personal private key and the peer's public key.

The shared secret is computed by multiplying the private key with the peer public key.

```
SharedSecret = PubKeyA*PrivKeyV
```

It computes a hash over the SharedSecret, using the sha512 algorithm and will use the first 32 bytes as the encryption key for the AES256 algorithm.

```
TransferEncryptionKey = Sha512(SharedSecret)[0:32]
```

Using this key, will encrypt the salsa20 key. However, before it does this it first xors the salsa20 key with the first 16 bytes from the victims public key.

```
EncryptedSalsaKey = AES256(salsa20key ^ PubKeyV[0:16], key=
TransferEncryptionKey);
```

For the attacker to obtain the Salsa20 key, it needs the EncryptedSalsaKey as well as the victim's generated public key. Thus it makes a buffer containing the generated public key, which is 49 bytes, and the encrypted salsa key, which is 16 bytes long.

```
PersonalBuffer = PubKeyV,EncryptedSalsaKey
```

The final thing that must be done is to transform each buffer into a printable one. It does this using base58 encoding (https://en.wikipedia.org/wiki/Base58). It doesn't use 64 encoding to prevent the user from confusing characters that look very similar (such as l and I or 0(zero) and O). After this is done, it performs a SHA256 hash on the resulted string and prepends the first byte of the hash, in hexadecimal format, to the base58 string. This is used as checksum over the resulted string.

```
PrintablePersonalBuffer = Base58Encoding(PersonalBuffer)
      PersonalDecryptionCode = Sha256(PrintablePersonalBuffer)[0],
PrintablePersonalBuffer
PersonalBuffer = PubKeyV,EncryptedSalsaKey
```

For the attacker to obtain the Salsa20 key, he has to reverse the above steps. The only thing that is different is obtaining the TransferEncryptionKey. He can compute the key in a similar way to the one performed on the victim's computer. The shared secret value (ECDHE) is computed by multiplying the victim's public key (which is encoded in the Personal Decryption Code) with the private key of the attacker. Due to how the keys are generated on a public key cryptography system, the shared secret will have the same value.

The Truth About the Petya Ransomware

While ransomware is usually extremely resilient in terms of using strong encryption to deny access to all disk files, Petya goes about it differently, by preventing the file system from reading the contents of the disk. This approach has obvious performance benefits as it's a lot faster than going through each disk file and encrypting it, and it's just as effective.

Malware developers behind Petya exhibit some in-depth low level-knowledge about how Microsoft's file system technology works, along with advance assembly language skills, needed to create their own bootloader and design a user interface for bitcoin payment.

With most ransomware usually running in user mode, the overall user experience is usually consistent and allows victims to easily interact with the ransomware and pay the fee. Since with Petya this was not the case

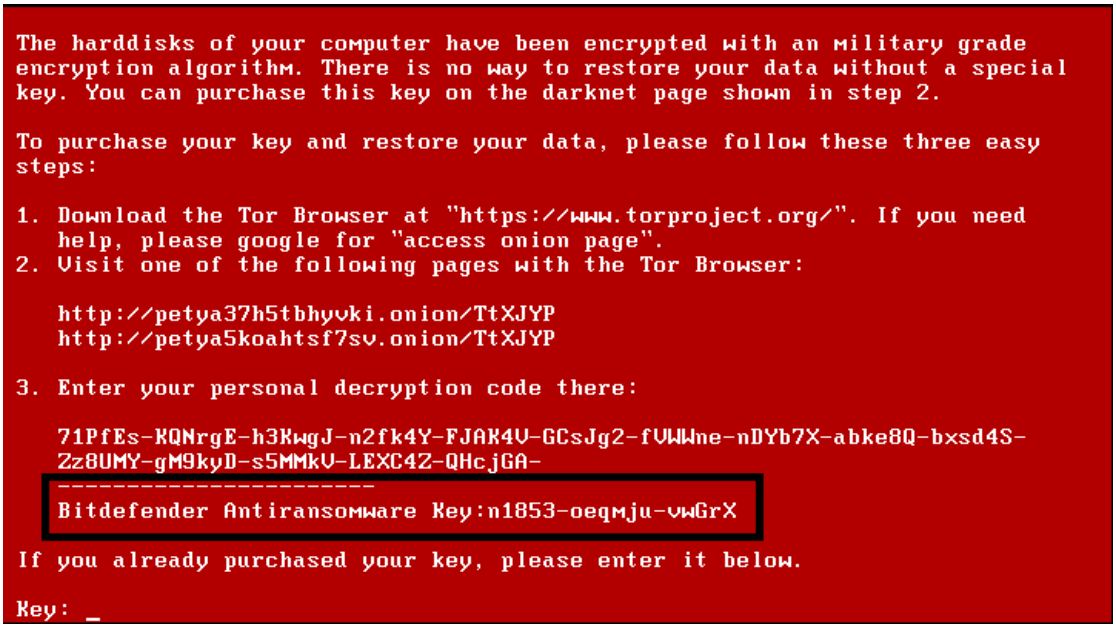
as they decided to go with a custom – outside the operating system – interface, the overall user experience is cumbersome and clunky.

While the skills necessary to write the Petya ransomware have scored high marks, ransomware is usually about making it easy for the victim to pay while at the same time making it impossible to regain access to their files without the decryption key.

Those that have experience a Petya ransomware infection are strongly encouraged to **download the Bitdefender Petya Ransomware Vaccine now!**

Note: While the Bitdefender Petya Ransomware Vaccine does not protect against encryption, its purpose is to make decryption a lot easier. Particularly, in case of Petya infection, the user will be presented with the decryption key so he can start decrypting his data immediately.

Below, you'll find an actual screenshot with how the Bitdefender Petya Ransomware Vaccine provides the decryption key.



About Bitdefender

Bitdefender is a global security technology company that delivers solutions in more than 100 countries through a network of value added alliances, distributors and reseller partners. Since 2001, Bitdefender has consistently produced award-winning business and consumer security technology, and is a leading security provider in virtualization and cloud technologies. Through R&D, alliances and partnership teams, Bitdefender has elevated the highest standards of security excellence in both its number-one-ranked technology and its strategic alliances with the world's leading virtualization and cloud technology providers. More information is available at http://www.bitdefender.com/.

B

Publication Date: April 2016

Bitdefender

BD-NGZ-Apr.19.2016-TK#: 86847