Petya: the two-in-one trojan

securelist.com/petya-the-two-in-one-trojan/74609/

Authors

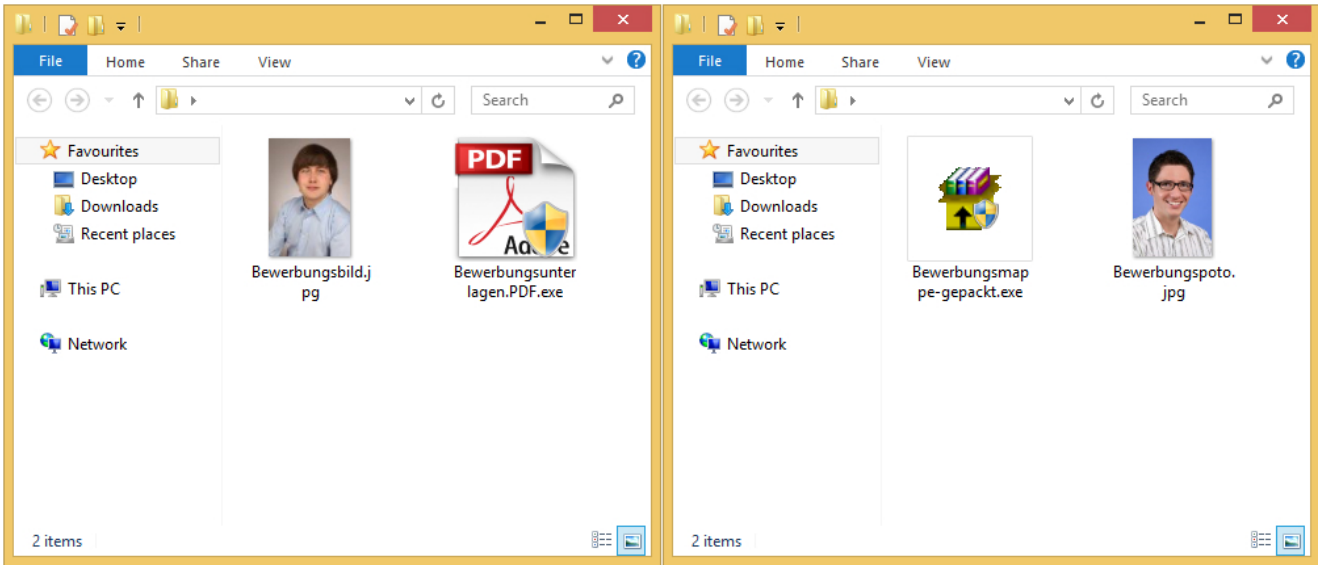
Expert Fedor Sinitsyn

Infecting the Master Boot Record (MBR) and encrypting files is nothing new in the world of malicious programs. Back in 1994, the virus OneHalf emerged that infected MBRs and encrypted the disk contents. However, that virus did not extort money. In 2011, MBR blocker Trojans began spreading (Trojan-Ransom.Win32.Mbro) that infected the MBR and prevented the operating system from loading further. The victim was prompted to pay a ransom to get rid of the problem. It was easy to treat a system infected by these blocker Trojans because, apart from the MBR, they usually didn't encrypt any data on the disk.

Today, we have encountered a new threat that's a blast from the past. The Petya Trojan (detected by Kaspersky Lab products as Trojan-Ransom.Win32.Petr) infects the MBR preventing normal system loading, and encrypts the Master File Table (MFT), an important part of the NT file system (NTFS), thus preventing normal access to files on the hard drive.

The infection scenario

The people spreading Petya attack their potential victims by sending spam messages containing links that download a ZIP archive. The archive contains the Trojan's executable file and a JPEG image. The file names are in German (Bewerbungsunterlagen.PDF.exe, Bewerbungsmappe-gepackt.exe), are made to look like resumes for job candidates, and target HR staff in German-speaking countries.



Contents of the archives downloaded from links in spam

The cybercriminals didn't bother with automatic escalation of privileges – the manifest of the Trojan's executable file contains the following standard record:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
  version="1.0.0.0"
  processorArchitecture="*"
  name="WinRAR SFX"
  type="win32"/>
<description>WinRAR SFX module</description>
<trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
  <security>
    <requestedPrivileges>
      <requestedExecutionLevel level="requireAdministrator"
      uiAccess="false"/>
    </requestedPrivileges>
  </security>
</trustInfo>
<dependency>
  <dependentAssembly>
    <assemblyIdentity
      type="win32"
```

If the user launches the malicious executable file Petya, Windows will show the standard UAC request for privilege escalation. If the system has been properly configured by the system administrators (i.e. UAC is enabled, and the user is not working from an administrator account), the Trojan won't be able to run any further.

Unfortunately, a user who has the privileges to agree to a UAC request often underestimates the potential risks associated with launching unknown software with elevated rights.

How it works

The executable file and the packer

A Petya Trojan infection begins with the launch of the malicious executable file. The samples of the Trojan that Kaspersky Lab received for analysis are, just like most other malware samples, protected with a customized packer. When the executable file launches, the malicious packer's code begins to work – it unpacks the malicious DLL Setup.dll into a newly designated RAM area, and then passes control to it.

Cybercriminals typically use packers to avoid detection – circumvent static signatures, trick the heuristic analyzer, etc. While investigating the Petya packer, we noticed an unusual trick used by the cybercriminals.

Cybercriminals often try to create the packer in such a way that a packed malicious executable file looks as similar as possible to a regular legitimate file. Sometimes, they take a legitimate file and substitute part of the code with malicious code. That's what they did with Petya, with one interesting peculiarity: it was a part of the standard compiler-generated runtime DLL that was replaced with malicious code, while the function WinMain remained intact. The illustration below shows the transition, beginning from the entry point ("start"). As can be seen, the function of unpacking malicious code (which we dubbed "evil") is called from the legal function __calloc_crt which is part of the runtime code.

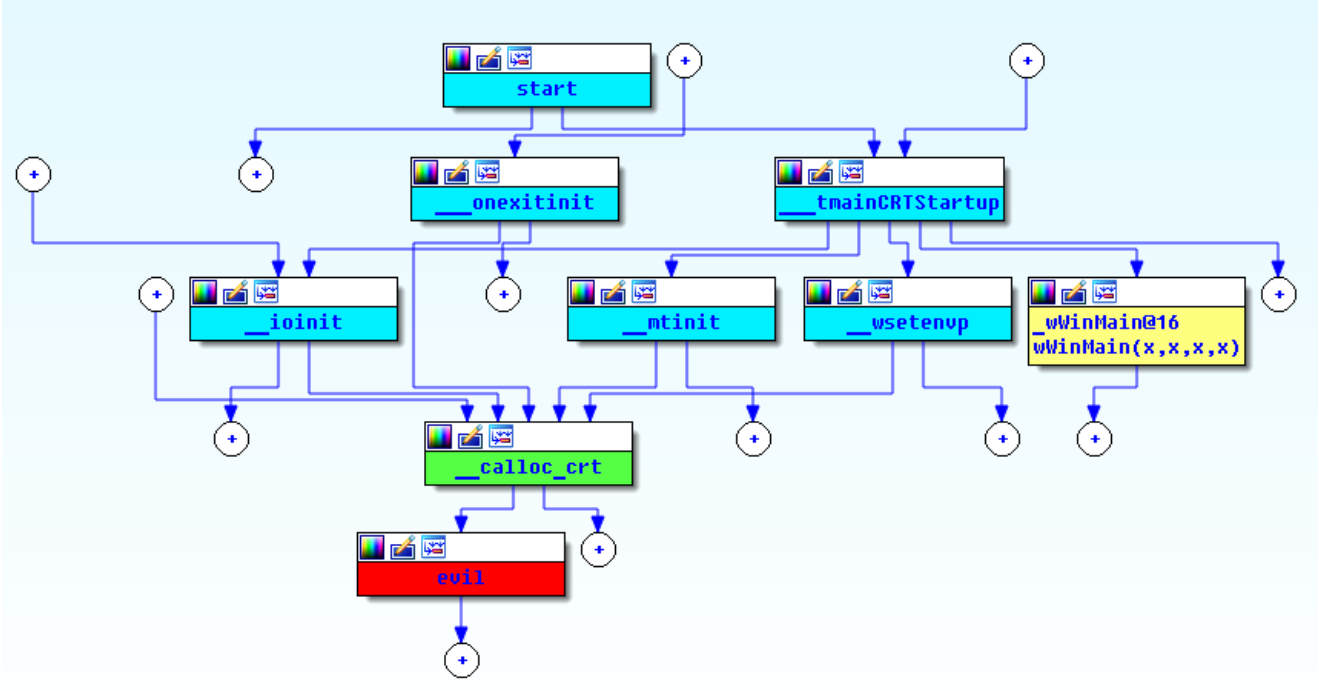


Diagram of transitions between the malicious packer's functions

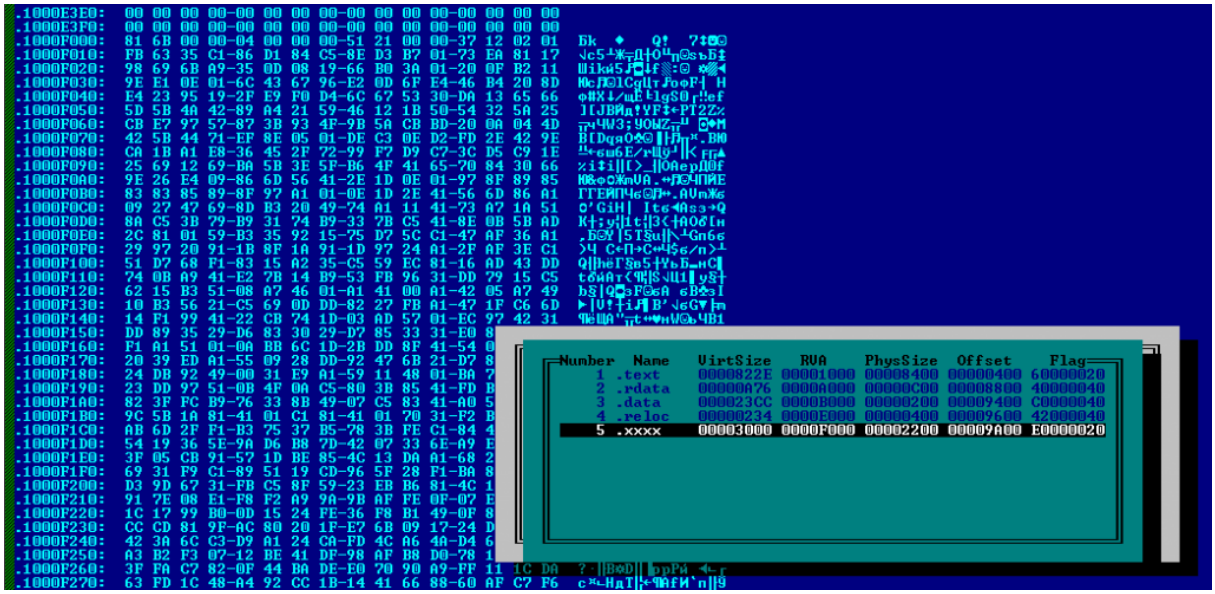
Why do it that way? Obviously, the creators of the malicious packer were trying to trick an inattentive researcher or automatic analyzers: the file looks legitimate – WinMain doesn't contain malicious code – so it's possible that it will be overlooked. Besides, if the breakpoint is set at WinMain during debugging, then the malicious code works (and sends the system into BSOD, as we will discuss later in detail) and execution is over before the breakpoint is even reached.

Kaspersky Lab has detected Petya samples that masquerade as legitimate files written in C/C++ and in Delphi.

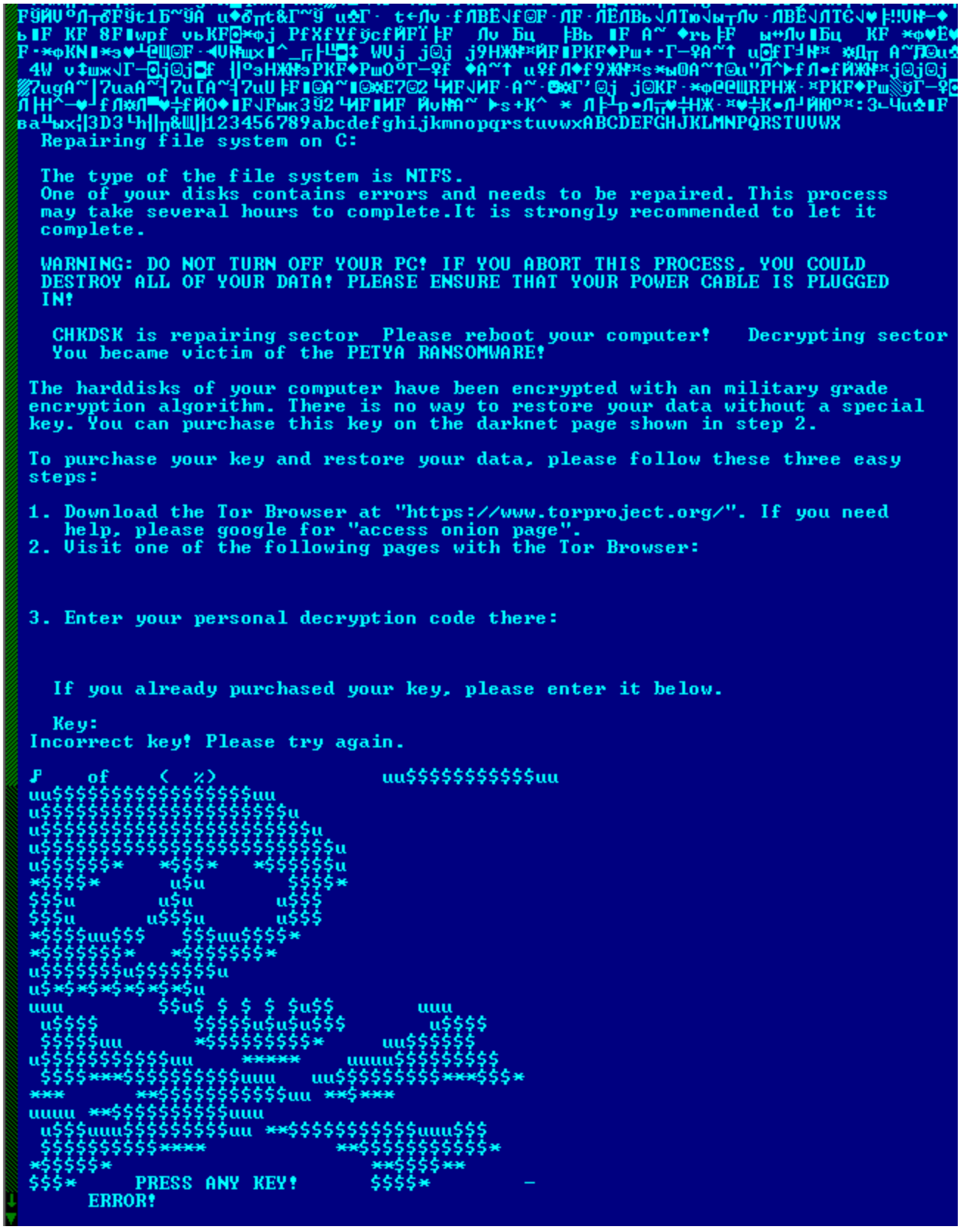
The malicious DLL

Setup.dll is a DLL with just one export: _ZuWQdweafdsg345312@0. It is written in C and compiled in Microsoft Visual Studio. The cybercriminals used an implementation of cryptographic algorithms available in the public library mbedtls (formerly polarssl). Setup.dll is not saved to the hard drive as a separate file, but always remains in the RAM.

When Setup.dll receives control, it decrypts the data contained in the section '.xxxx' and then proceeds to infect the victim computer.



The encrypted '.xxxx' section containing data



Fragment of the decrypted data from the '.xxxx' section

At a higher degree of abstraction, the actions of Setup.dll come down to the following:

1. Re-write the boot record on the hard drive with its own malicious loader;
2. Generate a key, infection ID and other auxiliary information, and save them to the hard drive;
3. Cause a system abort and reboot, thereby passing control to the malicious loader.

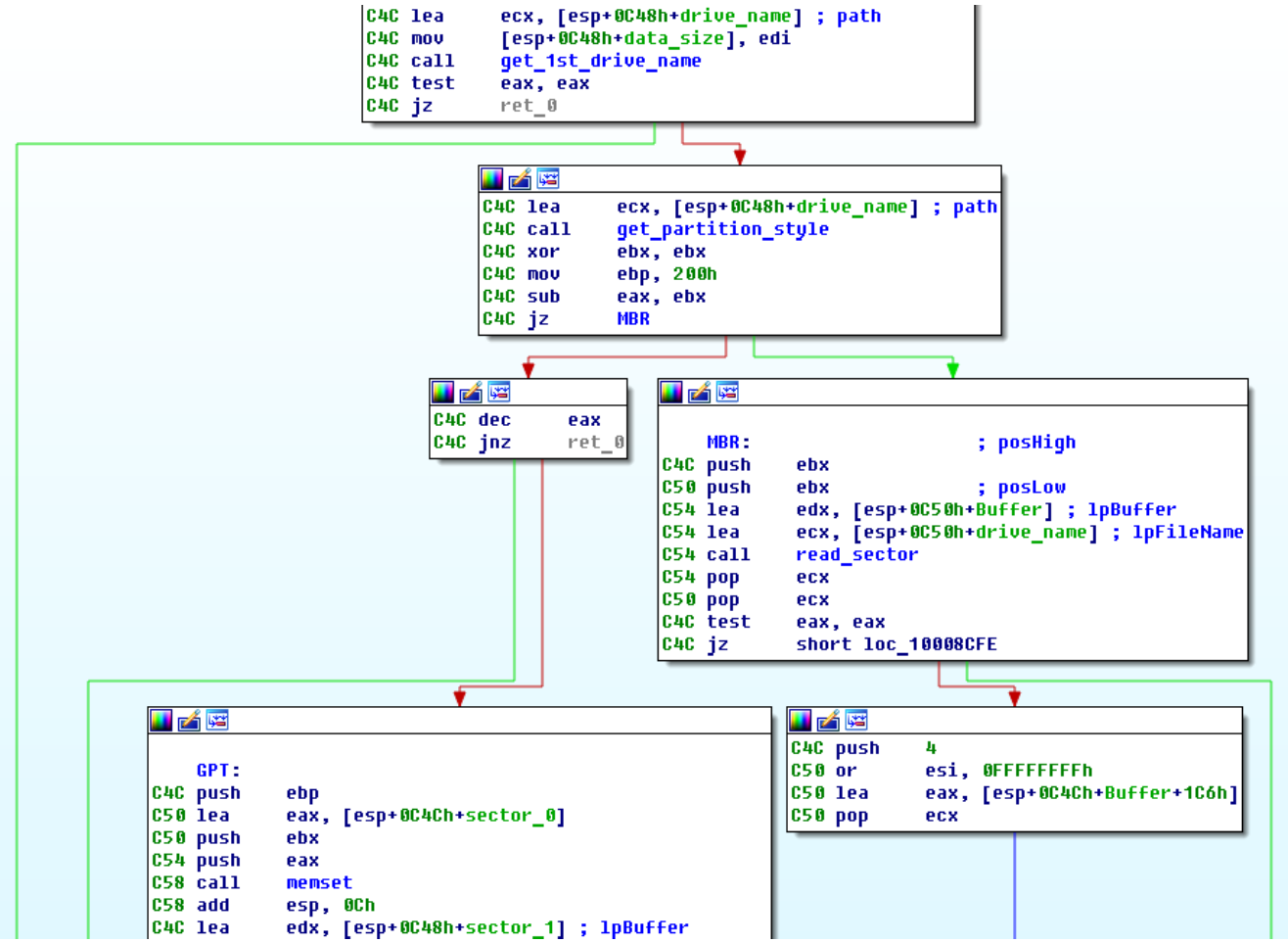
Now let's look in detail at how all of this is implemented in the Trojan. But before doing so, we need to define the terminology used.

Hard disk sector – the minimum addressable unit of a hard drive, typically 512 bytes.

Master boot record (MBR) – the code and the data written to Sector 0. After hardware is initialized, this code is used to boot the PC. Also, this sector contains the hard disks' partition table. A disk partitioned with MBR may have up to four primary partitions, and the maximum partition size is ~2.2 TB.

GUID Partition Table (GPT) – a more modern standard of hard drive layout. It supports up to 128 partitions, each up to 9.4 ZB in size (1 ZB = 10^{21} bytes.)

Now let's return to the Trojan under review. Setup.dll can infect disks partitioned according to either the older MBR standard or the more modern GPT standard. There are two alternative branches of execution sequences in the malicious program; the choice of execution branch depends on the data in the field PartitionStyle of the structure PARTITION_INFORMATION_EX.



Selection of the execution branch for disk infection, depending on whether the disk has MBR or GPT partitioning

Infecting an MBR disk

When infecting an MBR disk, Setup.dll performs the following actions:

1. Encrypts sector 0 (the original code and the MBR data) with the simple operation XOR 0x37 (ASCII '7'), writes the result to sector 56;
2. Encrypts sectors 1-33 with the same operation XOR 0x37;
3. Generates configuration data for the malicious loader, writes them to sector 54;
4. Creates the *verification sector* 55 populated with the repeating byte 0x37;
5. Copies the disk's NT signature and the partition table saved from the original MBR into its own first-level loader; writes first-level malicious code to sector 0 of the disk, and writes second-level code to sectors 34-50 (referred to here as the *malicious loader*);
6. Calls the function NtRaiseHardError, which causes the operating system to crash (BSOD – the 'blue screen of death').

When an MBR disk has been infected, the beginning of the disk has the following structure:

Number of sector	Content
0	First-level malicious loader
1 – 33	Encrypted sectors 1-33 (XOR 0x37)
34 – 50	Second-level malicious code
…	
54	Configuration sector of the malicious program
55	Verification sector (populated with byte 0x37)
56	Encrypted original MBR code (XOR 0x37)

Infecting a GPT disk

When infecting a GPT disk, Setup.dll performs more actions:

1. Based on Primary GPT Header data, it receives the address of GPT header copy;
2. Encrypts the GPT header copy with XOR 0x37;
3. Performs all the actions that are performed when encrypting an MBR disk.

When a GPT disk has been infected, the beginning of the disk has the following structure:

Number of sector	Content
0	First-level malicious loader
1 – 33	Encrypted sectors 1-33 (XOR 0x37)
34 – 50	Second-level malicious code
…	
54	Configuration sector of the malicious program
55	Verification sector (populated with byte 0x37)
56	Encrypted original MBR code (XOR 0x37)
…	
Backup LBA – Backup LBA + 33	Encrypted copy of GPT Header (XOR 0x37)

Generation of configuration data

In the configuration sector (sector 54), the Trojan keeps the data it needs to encrypt MFT and decrypt it if the victim pays the ransom. Generation of the configuration data consists of the following steps:

1. Setup.dll generates a random string that is 16 characters long [1-9, a-x, A-X]; we will call this string **password**;
2. Generate a pair of keys: **ec_session_priv** (a private key, a random large integer number) + **ec_session_pub** (public key, a point on a standard elliptic curve secp192k1);
3. Calculate the session secret: session_secret = ECDH (**ec_session_priv**, **ec_master_pub**); the cybercriminals' public key **ec_master_pub** is contained in the Trojan's body;
4. Calculate the aes_key = SHA512(session_secret) – only the first 32 bytes of the hash sum are used;
5. Encrypt the 'password' string by XORing it with the first 16 bytes of ec_session_pub: **password_xor** = **ec_session_pub**[0, 15] xor **password**;
6. Encrypt the result using AES-256 with the key aes_key: **password_aes_encr** = AES_enc(**password_xor**);
7. Create the array ec_session_data = [**ec_session_pub**, **password_aes_encr**];
8. Calculate base58: ec_session_data_b58 = base58_enc(ec_session_data);
9. Use the result to calculate SHA256: digest = sha256(ec_session_data_b58);
10. Create array: ec_data = [check1, check2, ec_session_data_b58], where check1, check2 are bytes calculated by the formulas:
    a = digest[0] & 0xF;
    b = (digest[0] & 0xF) < 10;
    check1 = (digest[0] >> 4) + 0x57 + ((digest[0] >> 4) < 10 ? 0xD9 : 0);
    check2 = a + 0x57 + (b ? 0xD9 : 0);
11. Based on the 'password', create a key for MFT encryption;

```
i = 0;
do
{
  config->salsa_key[2 * i] = password[i] + 0x7A;
  config->salsa_key[2 * i + 1] = 2 * password[i];
  ++i;
}
while ( i < 16 );
```

Pseudocode creating a key for MFT encryption

12. Generate IV – 8 random bytes which will be used during MFT encryption;
13. Generate infection ID and use it to create "personalized" URLs for ransom payment webpages.

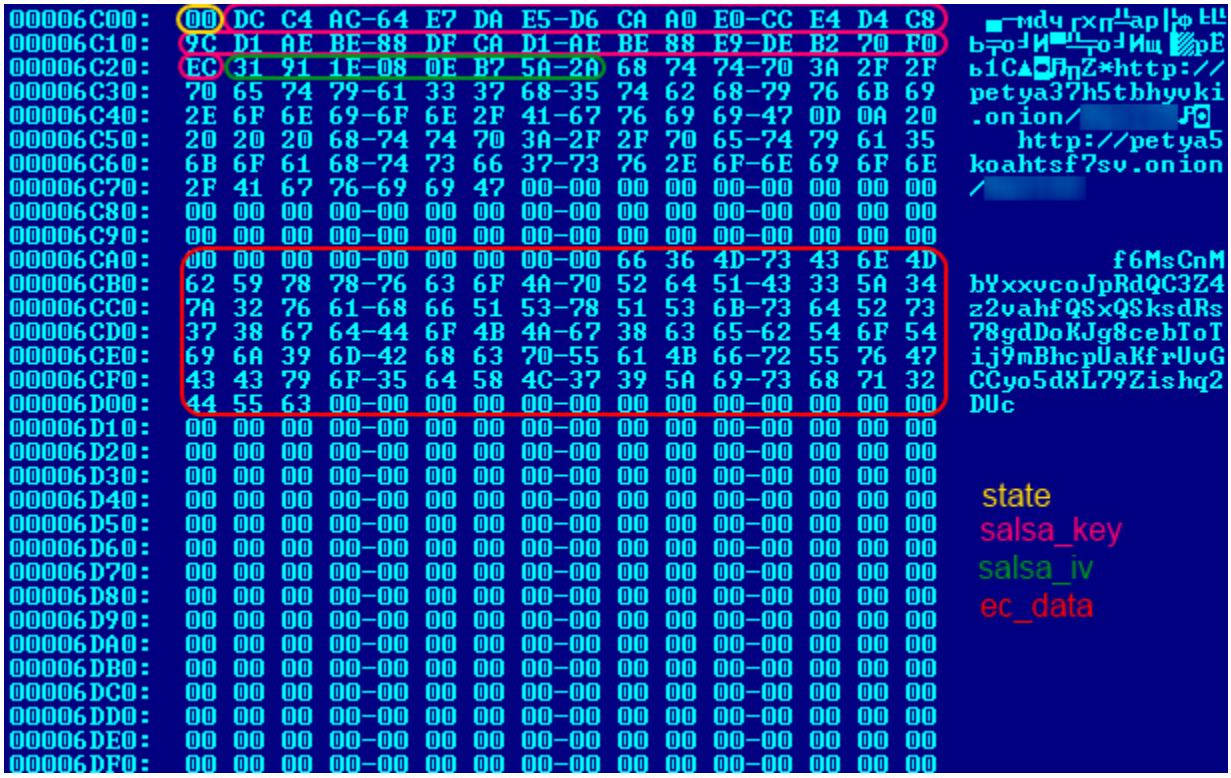
Ultimately, the configuration data structure looks like this:

```
00000000 config          struc ; (sizeof=0x200, mappedto_77)
00000000 state           db ?
00000001 salsa_key       db 32 dup(?)
00000021 salsa_iv        db 8 dup(?)
00000029 mal_urls        db 128 dup(?)
000000A9 ec_data         db 343 dup(?)
00000200 config          ends
```

In C language syntax, this structure can be presented as follows:

```
struct config
{
    char state;             //Infection state
    char salsa_key[32];     //Key for MFT encryption
    char salsa_iv[8];       //IV for MFT encryption
    char mal_urls[128];     //URLs of ransom payment webpages
    char ec_data[343];      //Key data for the payment page
};
```

This is what the configuration data looks like after it is written to the hard drive:



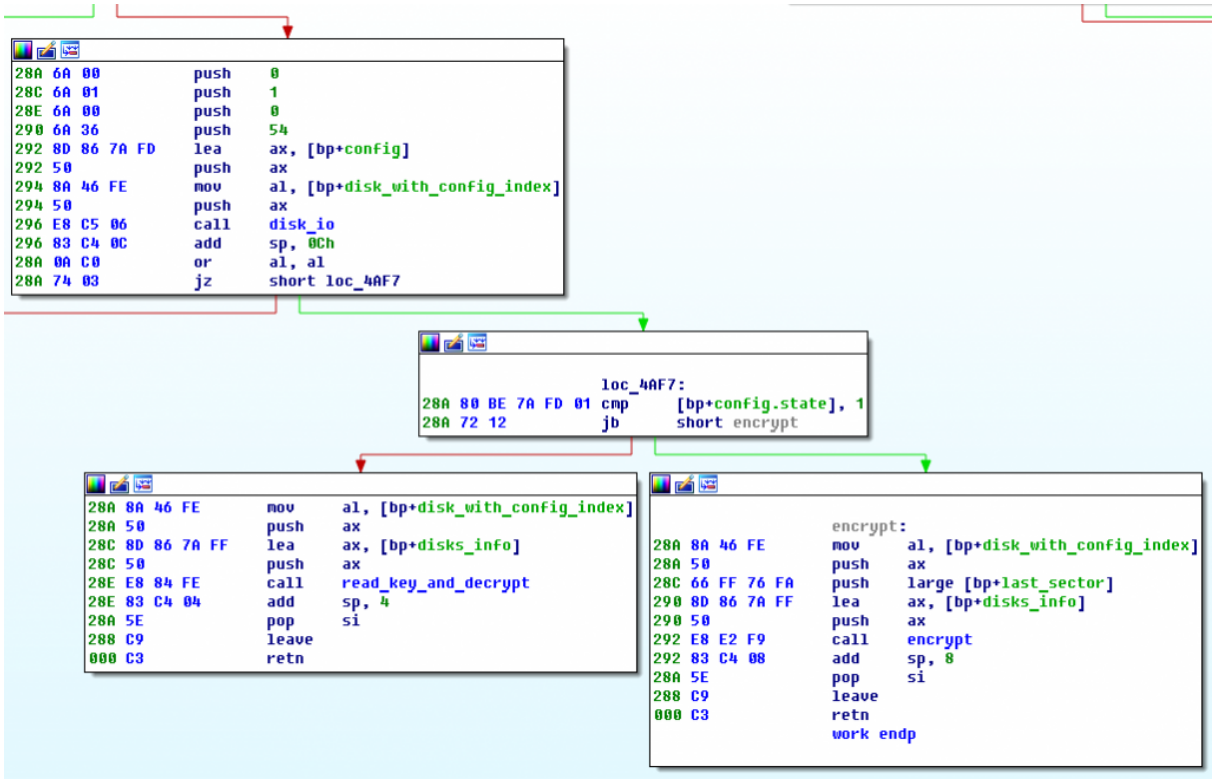
Note that if the user turns off their computer after this stage and doesn't switch it on again, only minimum damage will be done, as it is not difficult to decrypt data encrypted with 1-byte XOR. Therefore, a good piece of advice: if you launch an unknown file and your system

suddenly crashes, showing a blue screen, you should switch off your computer and get help from a qualified specialist. The specialist should be able to identify a Petya infection and restore the disk sectors encrypted with XOR.

If, however, the computer was re-booted, then the Trojan's third stage kicks in – the malicious code written to sectors 0 and 34–50.

The malicious loader

After rebooting, the code in sector 0 (the first-level loader) gains control. It loads the main second-level malicious code from sectors 34–50 into the memory and passes control to it. This code, in turn, receives information about the hard drives available in the system, searches for the disk where the configuration is written, reads the configuration data from sector 54 and, depending on the value in the field 'config.state', begins encryption (if the value is 0) or asks the user to enter the decryption key that they have purchased (if the value is 1).



Fragment of code implementing the Trojan's logic

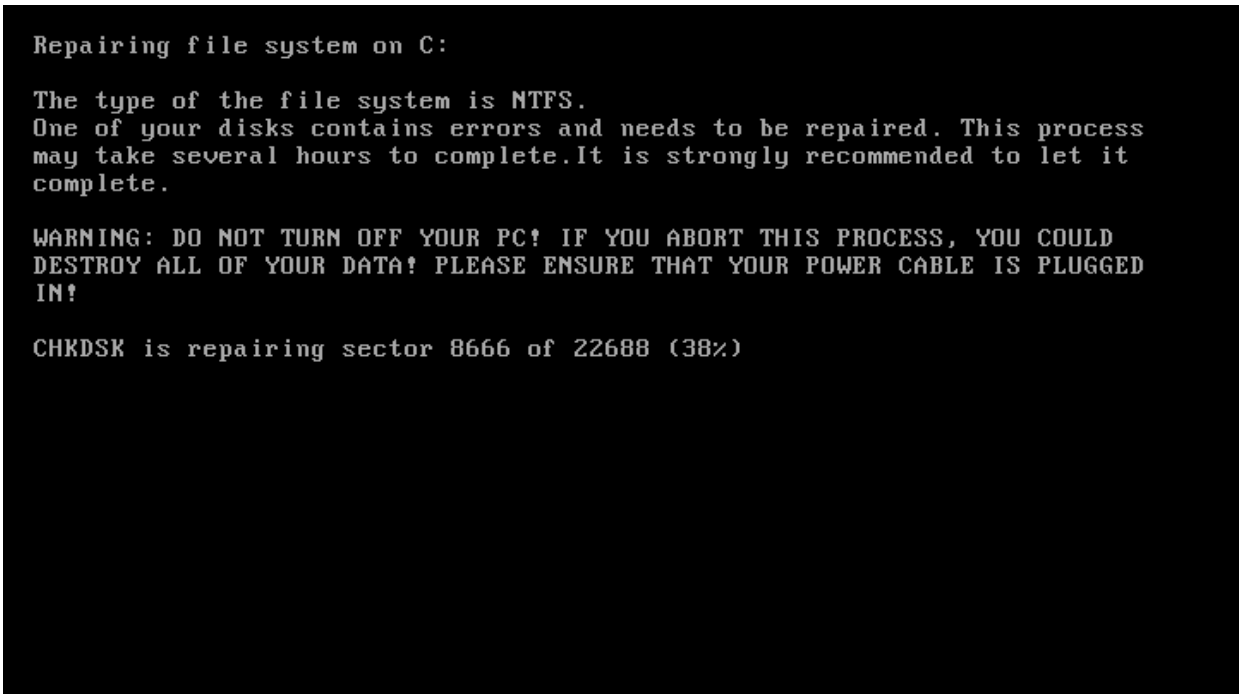
Encryption of MFT

The master file table (MFT) is a data structure with information about every file and directory on a volume formatted into NTFS, the file system that is used in all modern versions of Windows. The table contains the service data required to find each file on the disk. It can be

compared to a table of contents in a book that tells you on which page to find a chapter. Similarly, MFT indicates which logical cluster a file is located in.

It is namely this critical area that is attacked by Petya. If the value of 'config.state' is equal to 0 during launch, it does the following:

1. Displays a fake disk check message:

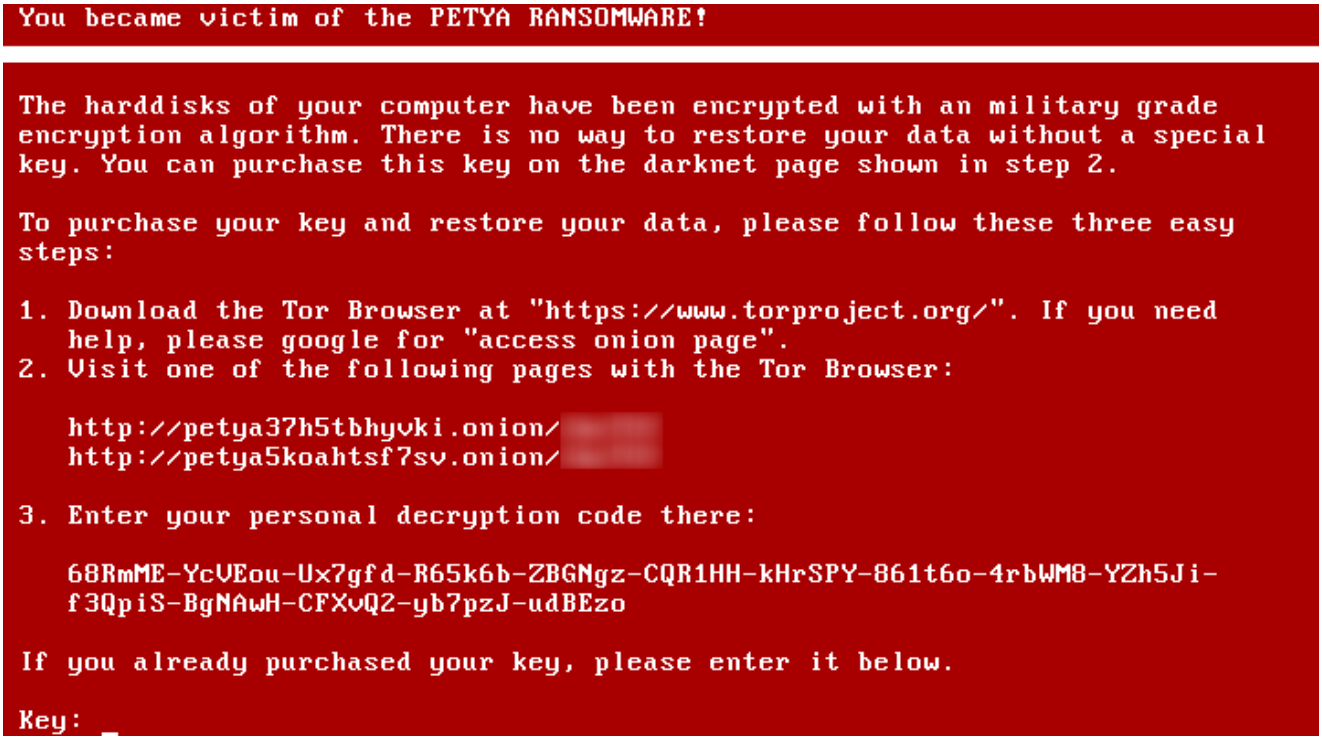


2. Reads the key 'config.salsa_key' from the configuration sector into a local array; sets this field to zero on the disk, sets 'config.state' field at 1;
3. Encrypts the verification sector 55 with the stream cipher Salsa20; this sector is populated beforehand with the byte 0x37 (see the section 'Infecting an MBR disk' above);
4. Searches for each partition's MFT on each connected hard drive;
5. Encrypts the MFT data with cipher Salsa20. Encryption is performed in parts of 8 sectors (i.e. the size of each part is 4 KB). A counter of the encrypted parts is kept in sector 57 of the first disk.
6. When encryption is over, it triggers a system reboot.

After the reboot, Petya displays an animated image of a flashing red and white skull drawn in ACCII-art style.

If the user presses any key, the Trojan displays a text which tells the victim in no uncertain terms what has happened.

Ransom demand and decryption



On this screen Petya displays links to the ransom payment webpages located in the Tor network (the addresses are specified in config.mal_urls), and the “personal decryption code” which the victim has to enter at either of the above sites. In reality, this “code” is the content of the field ‘config.ec_data’, hyphenated every six characters.

So, how do the cybercriminals plan to decrypt MFT, and are they even capable of doing so?

The ‘Key:’ field on this screen accepts a text string from the user. This string is checked for length (a 16-character long string is required), and then the Trojan uses it to calculate a 32-byte ‘salsa_key’ (following the algorithm discussed above in the section ‘Generation of configuration data’). The Trojan then attempts to decrypt the *verification sector* 55 with this key, and checks that the decrypted sector is completely populated with the byte 0x37. If it is, the key is considered correct, and Petya uses it to decrypt MFT. Then it decrypts all starting sectors encrypted with XOR 0x37, decrypts the original MBR and prompts the user to reboot the computer.

Thus, the correct string to be entered in the ‘Key:’ field is that very same ‘**password**‘ string that is generated in the first step when the configuration data is created.

```
Please reboot your computer!
```

Screen message displayed after successful decryption

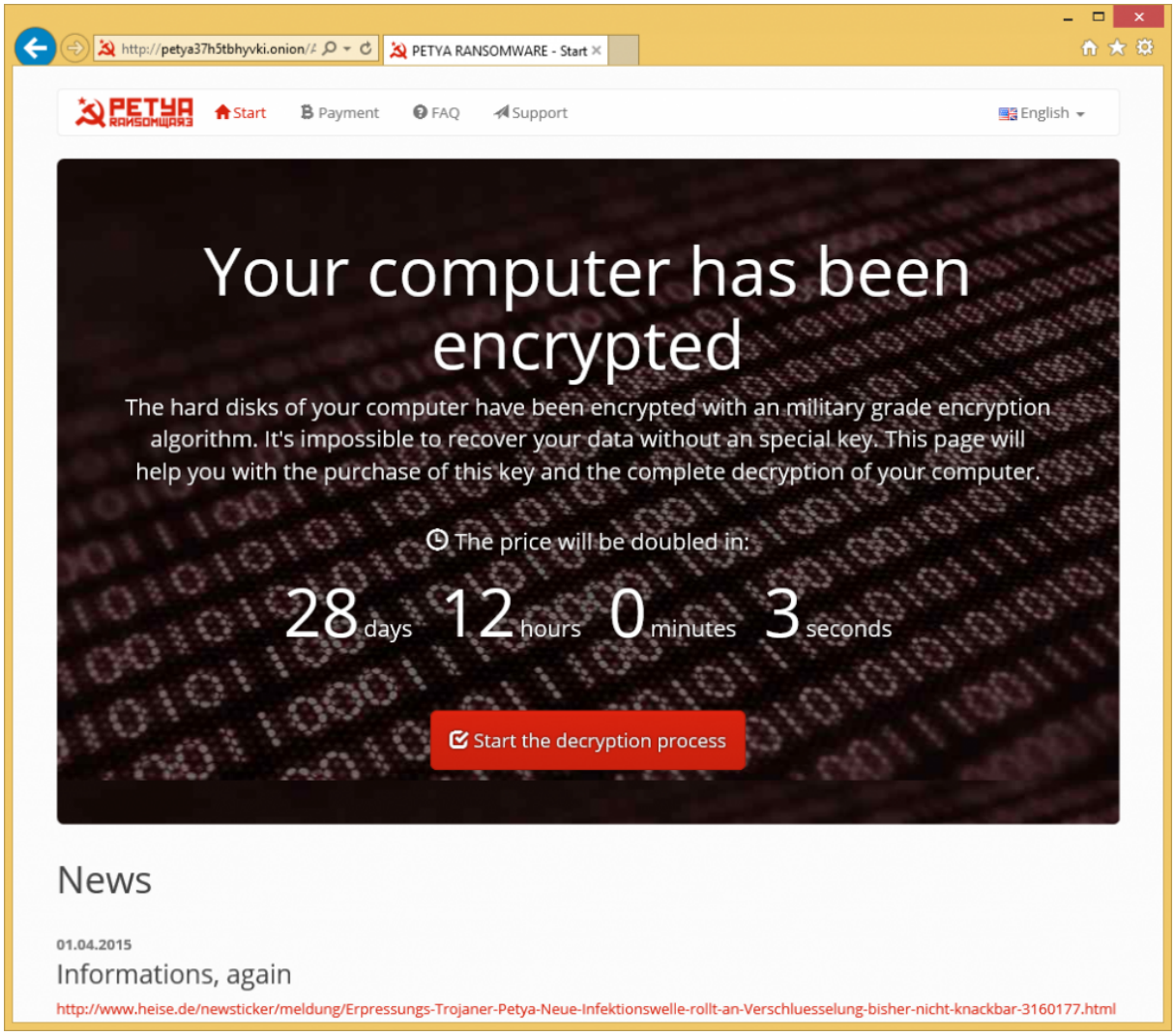
The question remains: how do the cybercriminals know this string so they can communicate it to a victim who has paid the ransom? No automatic communication with C&C servers is established during the entire infection life cycle. The answer lies in the description of the algorithm for generating configuration data.

The victim is prompted to manually enter their “personal decryption code” ec_data on the ransom payment webpage. The cybercriminal can then perform the following actions:

1. Decode base58: base58_dec(ec_session_data_b58) = ec_session_data = [**ec_session_pub**, **password_aes_encr**]
2. Calculate session_secret = ECDH(**ec_session_pub**, **ec_master_priv**), in accordance with the Elliptic curve Diffie–Hellman properties, where **ec_master_priv** is a private key known to the Trojan’s creators only;
3. Calculate aes_key = SHA256(session_secret);
4. Decrypt AES-256: **password_xor** = AES_dec(**password_encr**);
5. Knowing **ec_session_pub**, calculate the original **password** based on **password_xor**.

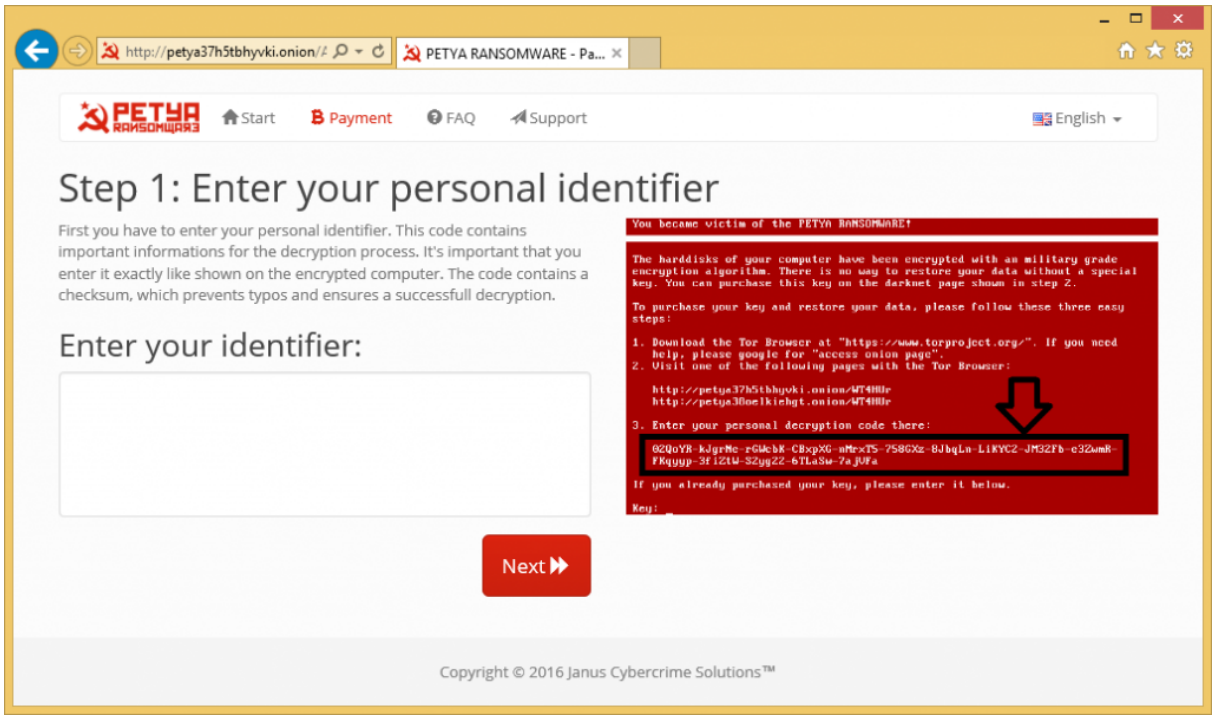
The ransom payment webpage

When we visit the Tor site at the URL provided by the Trojan, we see a page that requires a CAPTCHA to be entered, after which the main ransom payment page is loaded. The design of the page immediately catches the eye, with its hammer and sickle and the word ‘ransomware’ in pseudo-Cyrillic. It looks like a USSR parody along the lines of the game Red Alert.



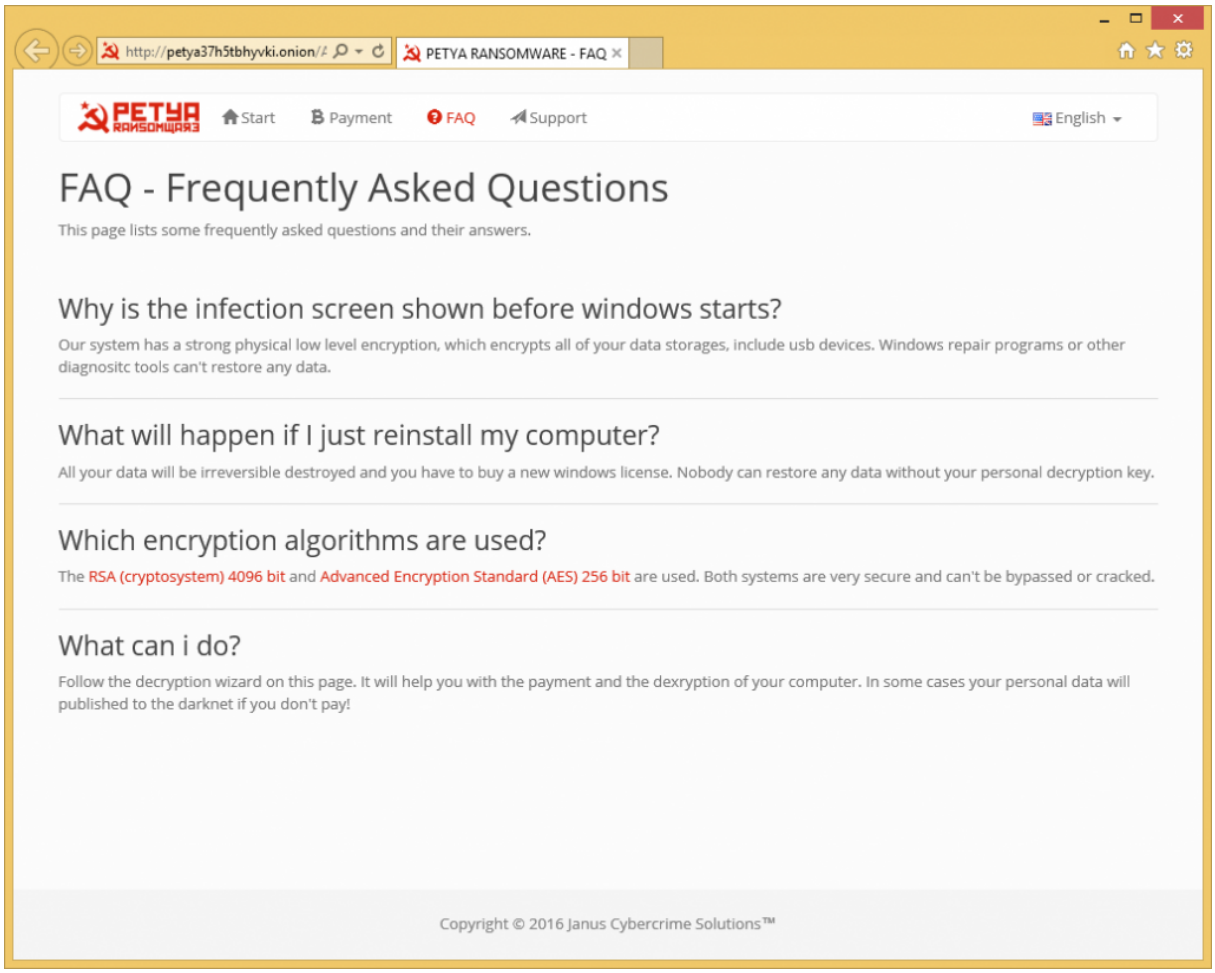
This page displays a countdown clock showing when the ransom price will be doubled, as well as regularly updated links to news and publications related to Petya.

When the ‘Start the decryption process’ button is pressed, you end up on a page that asks you to enter the value of ‘ec_data’, which is now called “your identifier” rather than “your personal decryption code”. It looks like the cybercriminals still haven’t decided what to call this part.



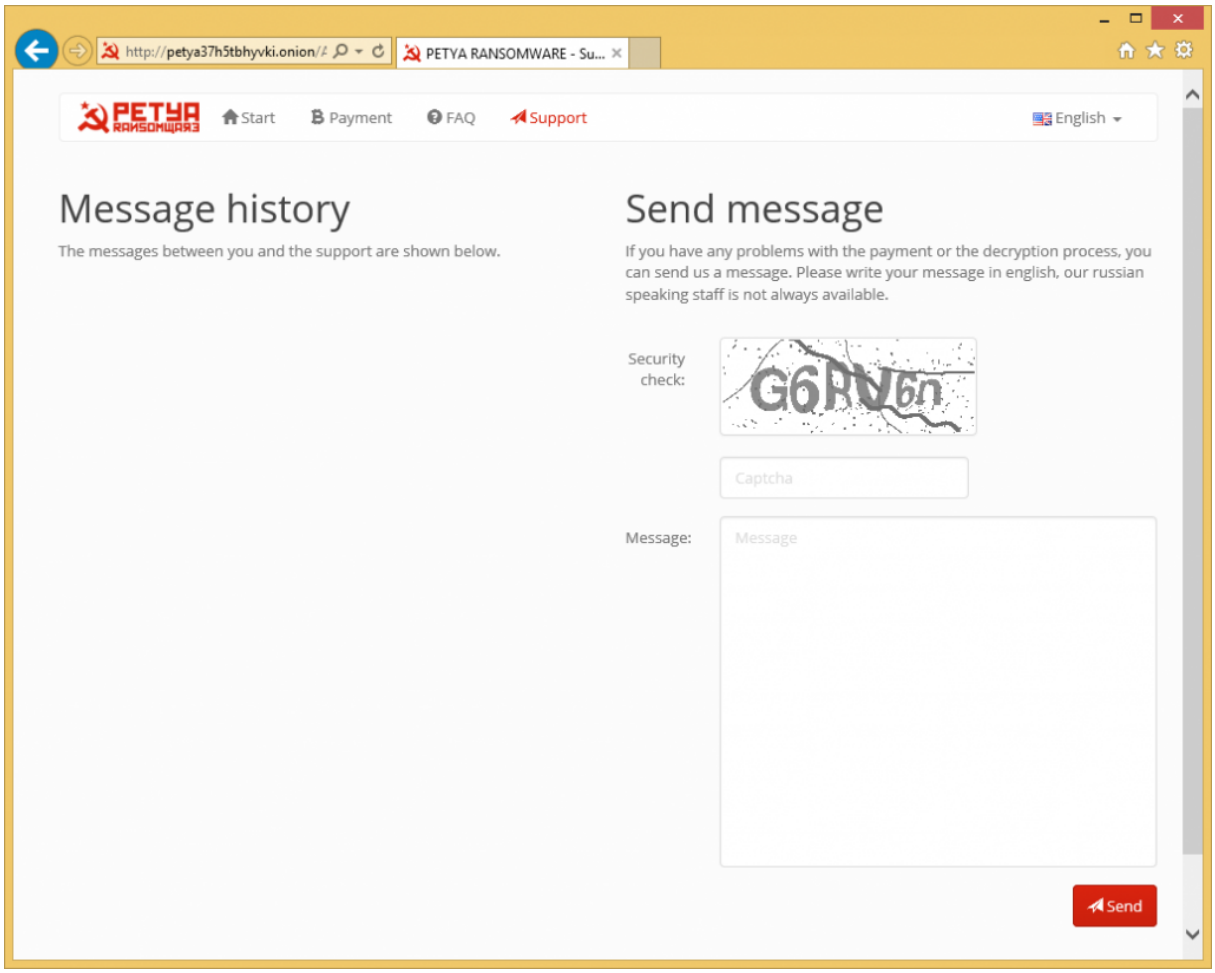
When the user enters this string, the site displays the amount of ransom in BTC, information on how to purchase bitcoins, and the address where the money should be sent.

As well as that, there are two other pages on the website: FAQ and Support.



The FAQ page

The FAQ page is interesting in that it contains false information: in reality, RSA is not used by the Trojan in any way, at any stage of infection.

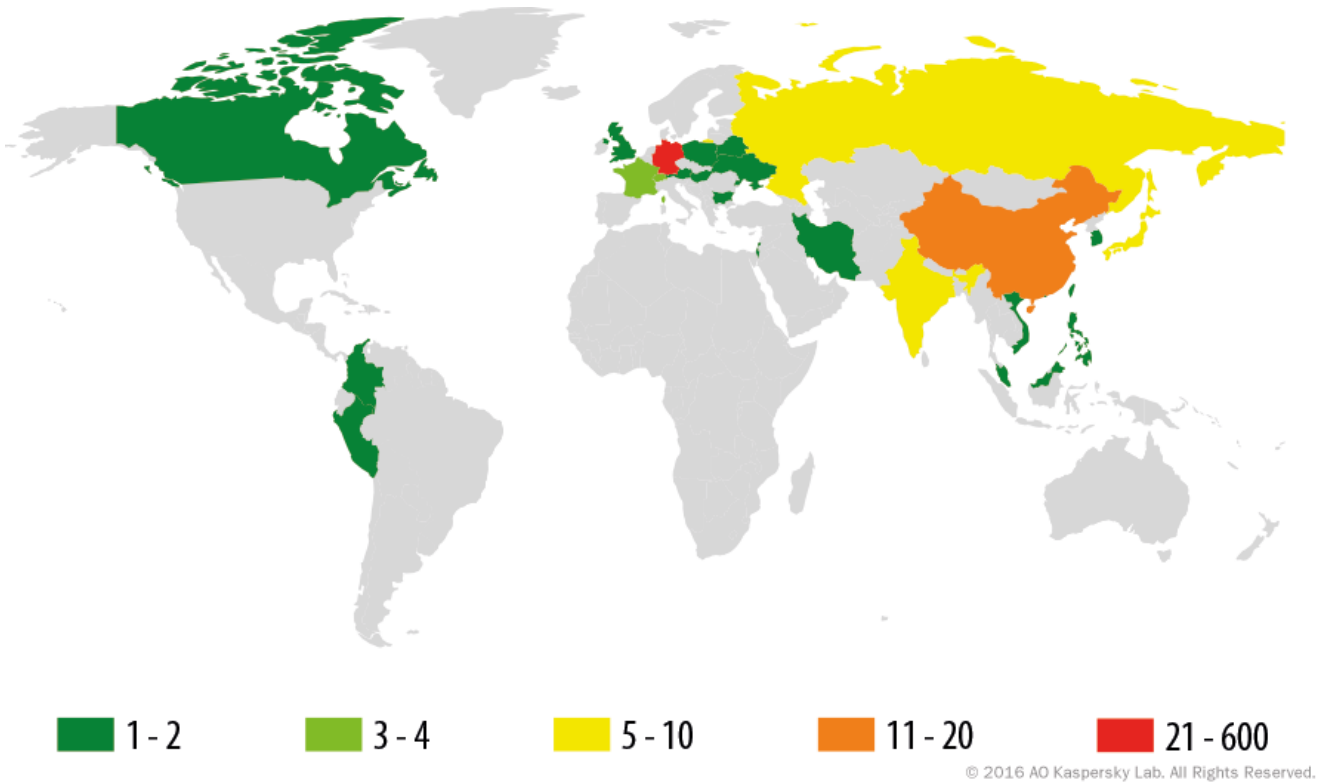


The Support page

On the Support page, the user is given the option of sending a message to the cybercriminals. One phrase in particular stands out: “Please write your message in english, our russian speaking staff is not always available”. This implies that there is at least one person in the group who speaks Russian.

Geographic distribution

As we noted above, the spam messages target German-speaking victims. KSN statistics clearly show that Germany is the main target for the cybercriminals.



TOP 5 countries attacked by Petya Trojan by the number of attacked users:

	Country	Number of attacked users
1	Germany	579
2	China	19
3	India	8
4	Japan	5
5	Russian Federation	5

Conclusion

After analyzing the Petya Trojan, we discovered that it is an unusual hybrid of an MBR blocker and data encryptor: it prevents not only the operating system from booting but also blocks normal access to files located on the hard drives of the attacked system.

Although Petya is noticeably different from the majority of ransomware that has emerged in the recent years, it can hardly be described as a fundamentally new development. The ideas behind the Trojan have been seen before in earlier malware; the creators of Petya have simply combined them all in a single creation. That said, it should be acknowledged that it requires a certain degree of technical skill to implement a low-level code to encrypt and decrypt data prior to OS booting.

Another interesting peculiarity about Petya is the pseudo-Soviet graphic design on the ransom payment website; the name of the Trojan also fits into the image of a “Russian Trojan” designed by cybercriminals. There is no certainty as to whether the Trojan’s creators originally come from Russia or other former Soviet states; however, the text on the payment page suggests there is at least one Russian speaker in the gang.

Kaspersky Lab’s products protect users from this threat: Petya’s executable files are detected with the verdict Trojan-Ransom.Win32.Petr; in addition, the behavior analyzer proactively detects even unknown versions of this Trojan with the verdict PDM:Trojan.Win32.Generic.

P.S. How to decrypt your data without paying the ransom

On April 8, some independent researchers reported that they had found a method of restoring the **password** without paying the ransom to the cybercriminals. The method is based on a genetic algorithm; with the 8-byte long IV (stored in configuration sector 54) and the content of the encrypted verification sector 55, you can calculate the value of the **password** that generates the salsa key, which can then be used to decrypt the MFT.

- Financial malware
- Malware Descriptions
- MBR
- Petya
- Ransomware

Authors

Fedor Sinitsyn

Petya: the two-in-one trojan

Your email address will not be published. Required fields are marked *