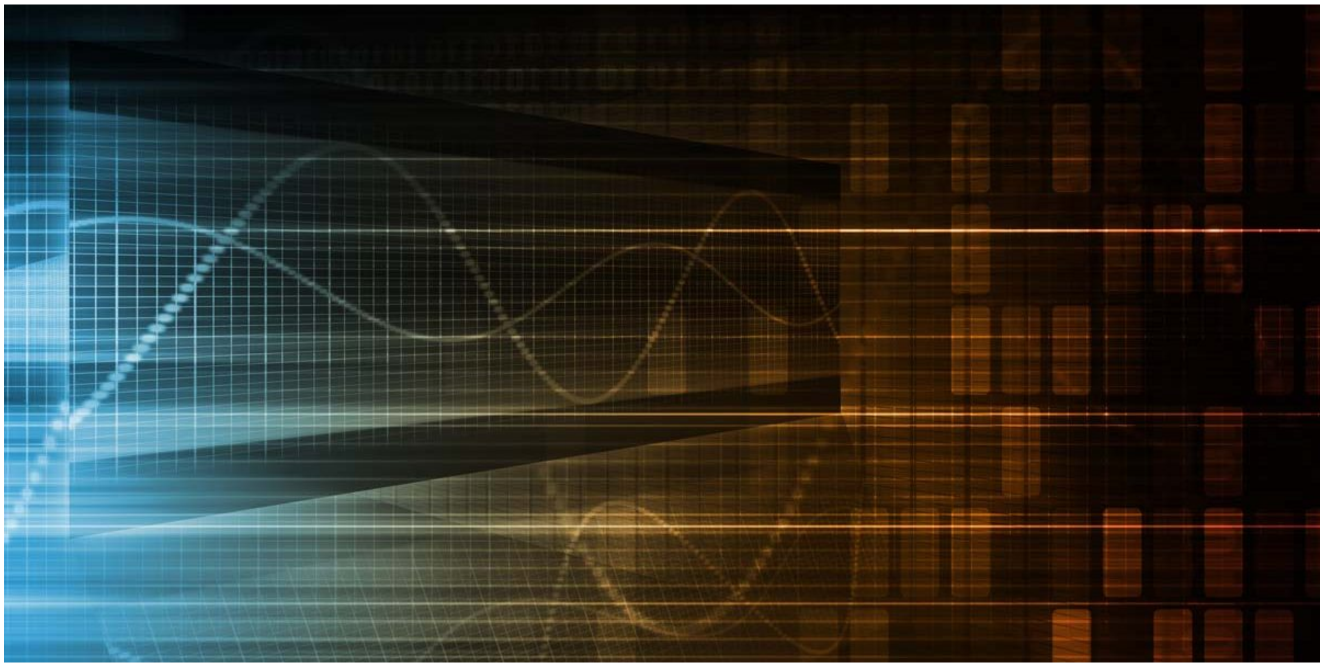
PetrWrap: the new Petya-based ransomware used in targeted attacks

securelist.com/blog/research/77762/petrwrap-the-new-petya-based-ransomware-used-in-targeted-attacks/



Authors

- Expert [Anton Ivanov]
- Expert [Fedor Sinitsyn]

Previously unknown ransomware technique

UPDATE June 27, 2017: About a new wave of Petya/Petrwrap/NotPetya/exPetr ransomware attacks read: Schroedinger's Pet(ya)

This year we found a new family of ransomware used in targeted attacks against organizations. After penetrating an organization's network the threat actors used the PsExec tool to install ransomware on all endpoints and servers in the organization. The next interesting fact about this ransomware is that the threat actors decided to use the well-known

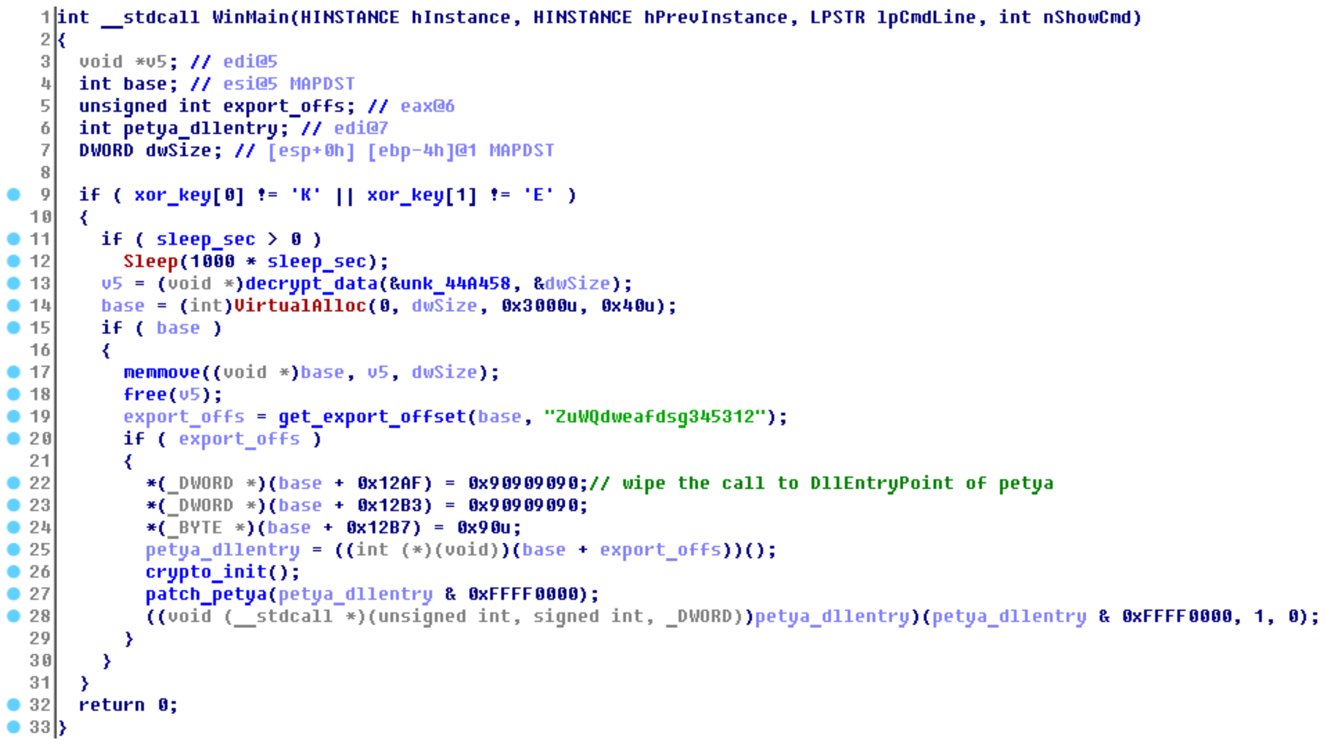
Petya ransomware to encrypt user data. As you may know, this family of ransomware has a RaaS model, but the threat actor decided not to use this ability. To get a workable version of the ransomware, the group behind PetrWrap created a special module that patches the original Petya ransomware “on the fly”. This is what makes this new malware so unique.

Tech details

The PetrWrap Trojan is written in C and compiled in MS Visual Studio. It carries a sample of the Petya ransomware v3 inside its data section and uses Petya to infect the victim’s machine. What’s more, PetrWrap implements its own cryptographic routines and modifies the code of Petya in runtime to control its execution. This allows the criminals behind PetrWrap to hide the fact that they are using Petya during infection.

Modus operandi

After being launched PetrWrap delays its execution (sleeps for 5400 seconds = 1.5 hours). After that it decrypts the main DLL of Petya from its data section and gets ready to call its exported function ZuWQdweafdsg345312. This function normally prepares Petya for further operations and starts the MBR overwrite process. PetrWrap, however, needs to hook a couple of Petya’s functions first, so it replaces the instructions that call Petya’s DllEntryPoint with NOPs (hex bytes 0x90). This prevents Petya from proceeding on its own and allows PetrWrap to make all the necessary computations and preparations before letting it continue.


```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  void *v5; // edi@5
  int base; // esi@5 MAPDST
  unsigned int export_offs; // eax@6
  int petya_dllentry; // edi@7
  DWORD dwSize; // [esp+0h] [ebp-4h]@1 MAPDST

  if ( xor_key[0] != 'K' || xor_key[1] != 'E' )
  {
    if ( sleep_sec > 0 )
      Sleep(1000 * sleep_sec);
    v5 = (void *)decrypt_data(&unk_44A458, &dwSize);
    base = (int)VirtualAlloc(0, dwSize, 0x3000u, 0x40u);
    if ( base )
    {
      memmove((void *)base, v5, dwSize);
      free(v5);
      export_offs = get_export_offset(base, "ZuWQdweafdsg345312");
      if ( export_offs )
      {
        *(_DWORD *)(base + 0x12AF) = 0x90909090;// wipe the call to DllEntryPoint of petya
        *(_DWORD *)(base + 0x12B3) = 0x90909090;
        *(_BYTE *)(base + 0x12B7) = 0x90u;
        petya_dllentry = ((int (*)(void))(base + export_offs))();
        crypto_init();
        patch_petya(petya_dllentry & 0xFFFF0000);
        ((void (__stdcall *)(unsigned int, signed int, _DWORD))petya_dllentry)(petya_dllentry & 0xFFFF0000, 1, 0);
      }
    }
  }
  return 0;
}
```

Main function of PetrWrap

After that PetrWrap makes the necessary cryptographic computations (we'll discuss them in more detail below), hooks two Petya procedures (which are responsible for the generation of the configuration data, dubbed petya_generate_config, and for the MBR overwrite process, dubbed petya_infect) and then passes the execution to Petya. For more information on what the original Petya was capable of, please see our previous publication.

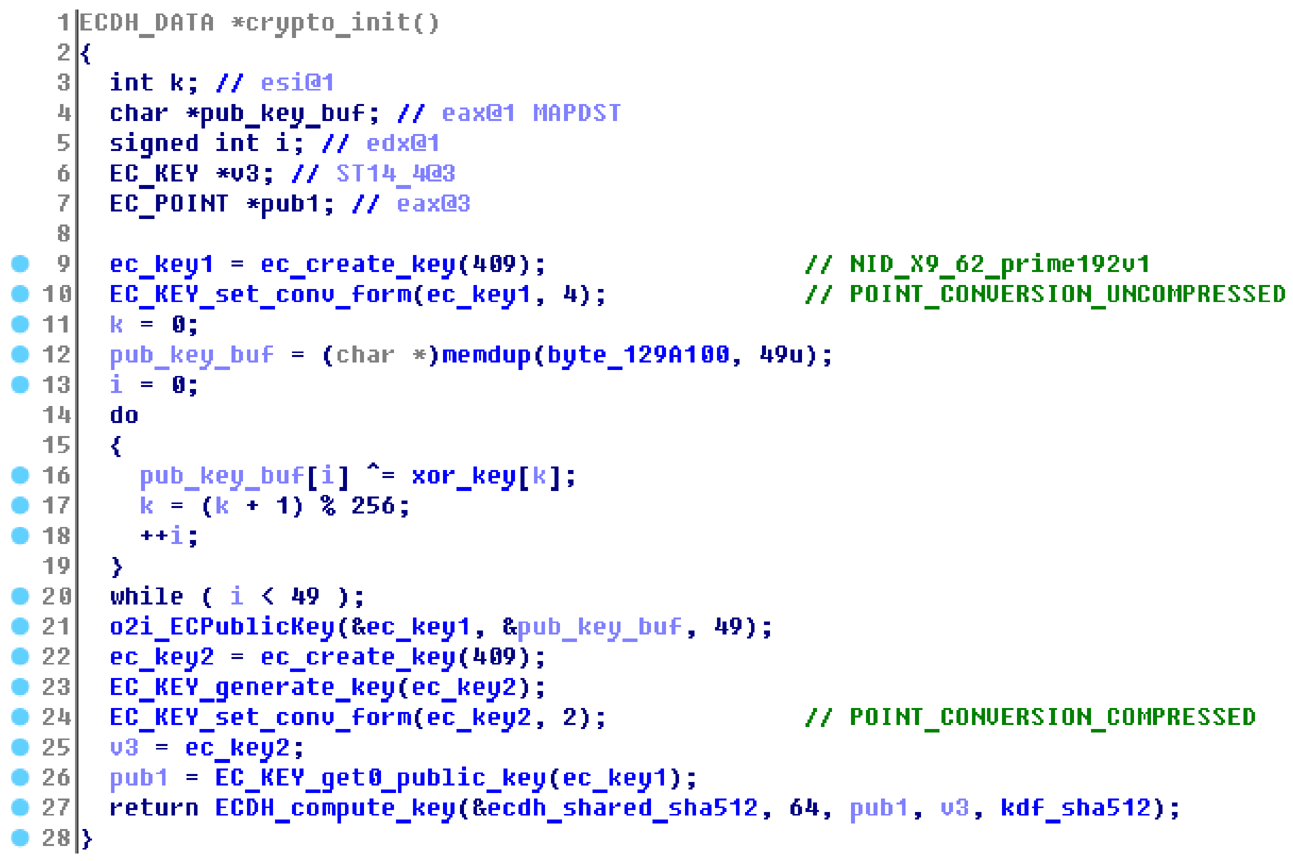
Cryptographic scheme

Normally, Petya generates a 16-byte key and uses the Salsa20 cipher to encrypt the MFT of the NTFS partitions found on local drives. To make decryption possible only by its operators, it uses the Elliptic Curve Diffie-Hellman (ECDH) key agreement algorithm with the curve secp192k1 and a public key is embedded into Petya's body.

The criminals behind PetrWrap faced a problem: if they used Petya as is, they would be unable to decrypt the victim's machine because they would need the Petya operators' private key. So what they decided to do was to completely replace the ECDH part of Petya with their own independent implementation and use their own private and public keys.

PetrWrap implementation uses cryptographic routines from OpenSSL (whereas Petya used the mbedtls library) and proceeds as follows:

- The Trojan contains an embedded public key master_pub (which is a point on the curve prime192v1 which is again different from the one chosen by Petya);
- During each infection PetrWrap generates a new pair of session keys ec_session_priv + ec_session_pub;
- Computes ecdh_shared_digest = SHA512(ECDH(master_pub, ec_session_priv));
- 'Intercepts' the salsa key generated by Petya and encrypts it using ecdh_shared_digest (there are a number of semi-useless manipulations which come down to essentially encrypting the salsa key with AES-256 using different parts of ecdh_shared_digest as the key and IV);
- Constructs user_id which is a string representation that contains the encrypted salsa key and the ec_session_pub;
- Passes this user_id to Petya, which uses it as if it was its own data (puts it into the configuration for the bootloader to be shown to the user after the PC reboot).



```
ECDH_DATA *crypto_init()
{
  int k; // esi@1
  char *pub_key_buf; // eax@1 MAPDST
  signed int i; // edx@1
  EC_KEY *v3; // ST14_4@3
  EC_POINT *pub1; // eax@3

  ec_key1 = ec_create_key(409);                     // NID_X9_62_prime192v1
  EC_KEY_set_conv_form(ec_key1, 4);                 // POINT_CONVERSION_UNCOMPRESSED
  k = 0;
  pub_key_buf = (char *)memdup(byte_129A100, 49u);
  i = 0;
  do
  {
    pub_key_buf[i] ^= xor_key[k];
    k = (k + 1) % 256;
    ++i;
  }
  while ( i < 49 );
  o2i_ECPublicKey(&ec_key1, &pub_key_buf, 49);
  ec_key2 = ec_create_key(409);
  EC_KEY_generate_key(ec_key2);
  EC_KEY_set_conv_form(ec_key2, 2);                 // POINT_CONVERSION_COMPRESSED
  v3 = ec_key2;
  pub1 = EC_KEY_get0_public_key(ec_key1);
  return ECDH_compute_key(&ecdh_shared_sha512, 64, pub1, v3, kdf_sha512);
}
```

The ECDH shared key computation implemented in PetrWrap

Hooked procedures

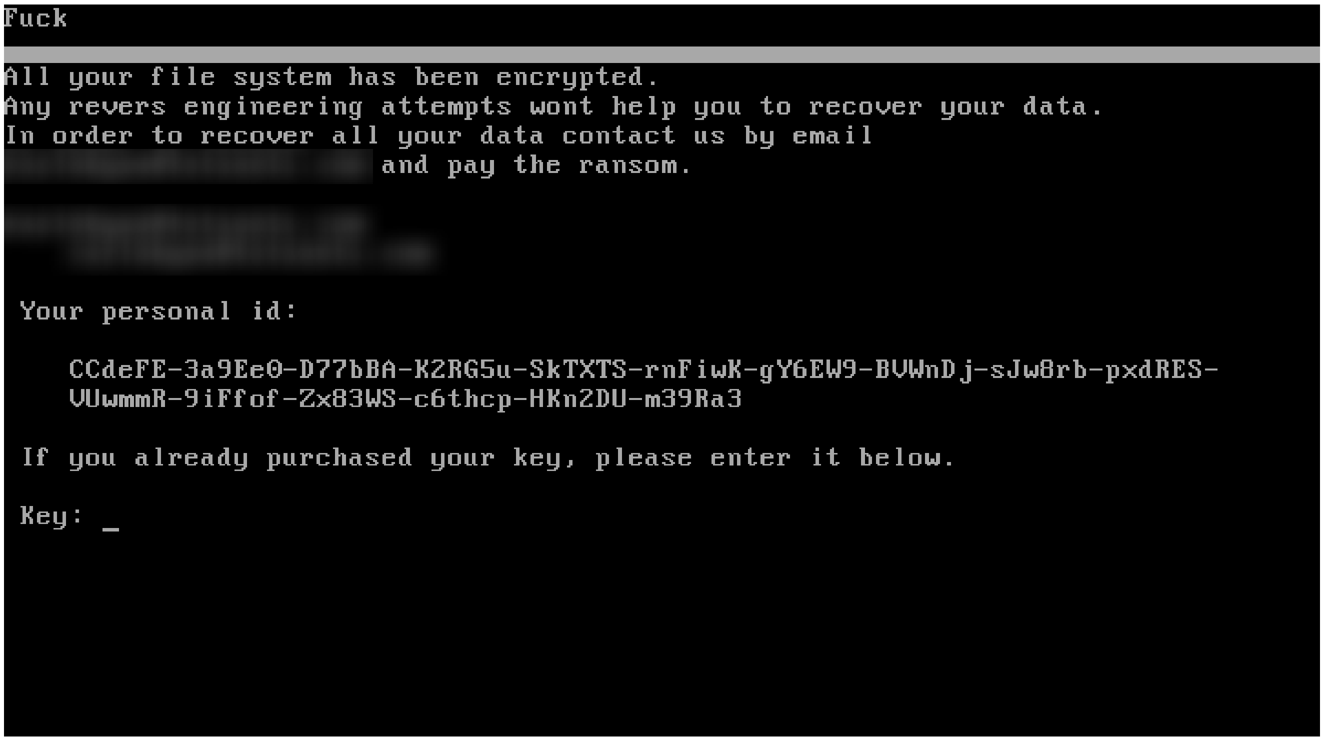
PetrWrap hooks two procedures in Petya which we will call petya_infect and petya_generate_config and replaces them with its own procedures dubbed wrap_infect and wrap_generate_config.

wrap_infect implements the following functionality:

- saves the salsa key generated by Petya for further use;
- patches the Petya bootloader code and ransom text in order to skip the flashing skull animation and to wipe all mention of Petya in the ransom message;
- passes execution to the original petya_infect procedure.

wrap_generate_config in turn does the following:

- calls the original petya_generate_config procedure;
- generates the user_id string according to the algorithm described in the previous paragraph;
- replaces Petya's id string with this newly generated user_id.



```
Fuck

All your file system has been encrypted.
Any revers engineering attempts wont help you to recover your data.
In order to recover all your data contact us by email
[redacted] and pay the ransom.

[redacted]

 Your personal id:

    CCdeFE-3a9Ee0-D77bBA-K2RG5u-SkTXTS-rnFiwK-gY6EW9-BVWnDj-sJw8rb-pxdRES-
    VUwmmR-9iFfof-Zx83WS-c6thcp-HKn2DU-m39Ra3

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

 Key: _
```

The screen of the infected machine

Technical summary

As a result of all the manipulations described above, PetrWrap achieves the following goals:

1. The victim's machine is locked and the MFT of NTFS partitions is encrypted securely (because Petya v3 which is used in this attack doesn't have flaws of the earlier versions and implements Salsa20 correctly);

2. The lockscreen doesn't show the flashing skull animation and doesn't contain any mentions of Petya which makes it harder to assess the situation and determine the extent of the caused damage;

3. The developers of PetrWrap didn't have to write the low-level bootloader code and risk making mistakes similar to the ones observed in earlier versions of Petya.

Decryption

Unfortunately, this family of ransomware uses a strong encryption algorithm, meaning a decryption tool is out of the question. However, victims can try restoring files using third-party tools such as R-Studio.

Detection

Kaspersky products successfully detect this ransomware as Trojan-Ransom.Win32.PetrWrap and PDM:Trojan.Win32.Generic.

Conclusion

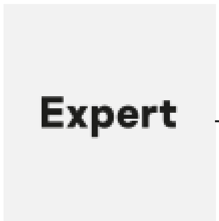
Targeted attacks on organizations with the main aim of encrypting data are becoming more popular. The groups using ransomware in their targeted attacks usually try to find vulnerable servers or servers with unprotected RDP access. After penetrating an organization's network they use special frameworks like Mimikatz to obtain the necessary credentials for installing ransomware throughout the network. To protect against such attacks, organizations need to keep their server software up to date, use secure passwords for remote access systems, install security solutions on their servers and use security solutions with behavioral detection components on their endpoints.

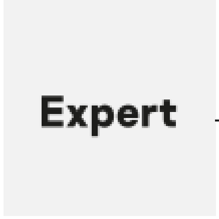
Sample MD5

17c25c8a7c141195ee887de905f33d7b – Trojan-Ransom.Win32.PetrWrap.b

- APT
- Encryption
- Financial malware
- Malware Technologies
- Petya
- RaaS
- Ransomware
- Targeted attacks

Authors

- 

 Anton Ivanov
- 

 Fedor Sinitsyn

PetrWrap: the new Petya-based ransomware used in targeted attacks

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