SynAck targeted ransomware uses the Doppelgänging technique

securelist.com/synack-targeted-ransomware-uses-the-doppelganging-technique/85431/

Authors

- Expert Anton Ivanov
- Expert Fedor Sinitsyn
- Expert Orkhan Mamedov

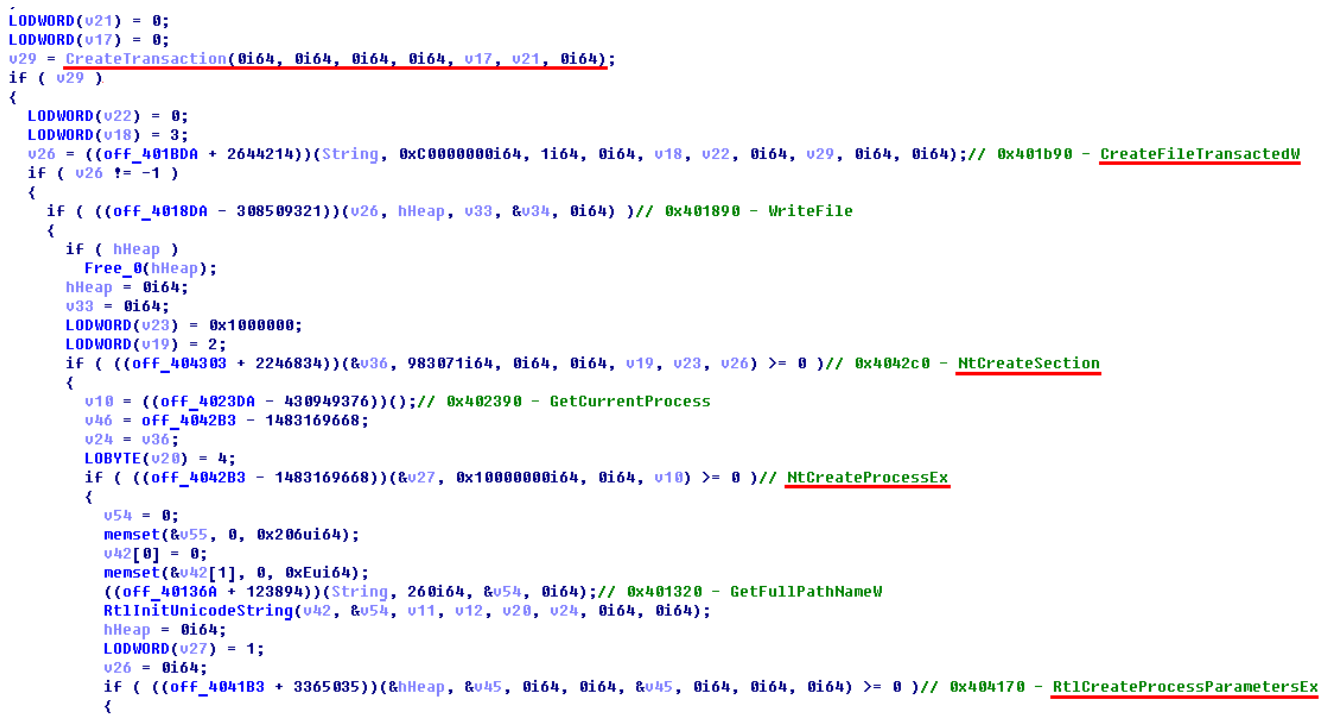
The Process Doppelgänging technique was first presented in December 2017 at the BlackHat conference. Since the presentation several threat actors have started using this sophisticated technique in an attempt to bypass modern security solutions.

In April 2018, we spotted the first ransomware employing this bypass technique – SynAck ransomware. It should be noted that SynAck is not new – it has been known since at least September 2017 – but a recently discovered sample caught our attention after it was found to be using Process Doppelgänging. Here we present the results of our investigation of this new SynAck variant.

Anti-analysis and anti-detection techniques

Process Doppelgänging

SynAck ransomware uses this technique in an attempt to bypass modern security solutions. The main purpose of the technique is to use NTFS transactions to launch a malicious process from the transacted file so that the malicious process looks like a legitimate one.



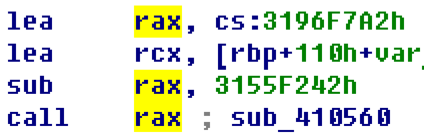
```
LODWORD(v21) = 0;
LODWORD(v17) = 0;
v29 = CreateTransaction(0i64, 0i64, 0i64, 0i64, v17, v21, 0i64);
if ( v29 )
{
  LODWORD(v22) = 0;
  LODWORD(v18) = 3;
  v26 = ((off_401BDA + 2644214))(String, 0xC0000000i64, 1i64, 0i64, v18, v22, 0i64, v29, 0i64, 0i64);// 0x401b90 - CreateFileTransactedW
  if ( v26 != -1 )
  {
    if ( ((off_4018DA - 308509321))(v26, hHeap, v33, &v34, 0i64) )// 0x401890 - WriteFile
    {
      if ( hHeap )
        Free_0(hHeap);
      hHeap = 0i64;
      v33 = 0i64;
      LODWORD(v23) = 0x1000000;
      LODWORD(v19) = 2;
      if ( ((off_404303 + 2246834))(&v36, 983071i64, 0i64, 0i64, v19, v23, v26) >= 0 )// 0x4042c0 - NtCreateSection
      {
        v10 = ((off_4023DA - 430949376))();// 0x402390 - GetCurrentProcess
        v46 = off_4042B3 - 1483169668;
        v24 = v36;
        LOBYTE(v20) = 4;
        if ( ((off_4042B3 - 1483169668))(&v27, 0x10000000i64, 0i64, v10) >= 0 )// NtCreateProcessEx
        {
          v54 = 0;
          memset(&v55, 0, 0x206ui64);
          v42[0] = 0;
          memset(&v42[1], 0, 0xEui64);
          ((off_40136A + 123894))(String, 260i64, &v54, 0i64);// 0x401320 - GetFullPathNameW
          RtlInitUnicodeString(v42, &v54, v11, v12, v20, v24, 0i64, 0i64);
          hHeap = 0i64;
          LODWORD(v27) = 1;
          v26 = 0i64;
          if ( ((off_4041B3 + 3365035))(&hHeap, &v45, 0i64, 0i64, &v45, 0i64, 0i64, 0i64) >= 0 )// 0x404170 - RtlCreateProcessParametersEx
          {
```

Binary obfuscation

To complicate the malware analysts' task, malware developers often use custom PE packers to protect the original code of the Trojan executable. Most packers of this type, however, are effortlessly unpacked to reveal the original unchanged Trojan PE file that's suitable for analysis.

This, however, is not the case with SynAck. The Trojan executable is not packed; instead, it is thoroughly obfuscated prior to compilation. As a result, the task of reverse engineering is considerably more complicated with SynAck than it is with other recent ransomware strains.

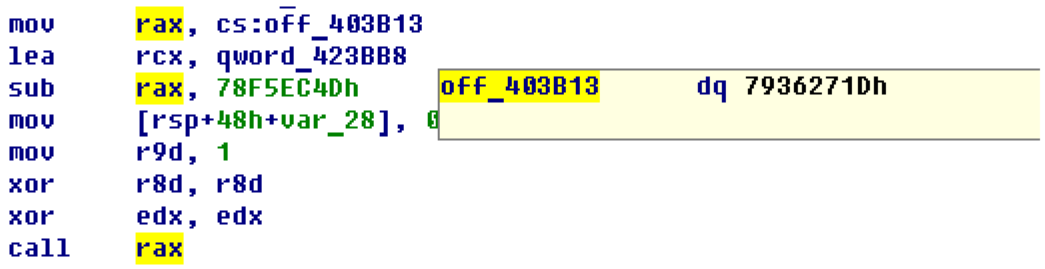
The control flow of the Trojan executable is convoluted. Most of the CALLs are indirect, and the destination address is calculated by arithmetic operation from two DWORD constants.



```
lea     rax, cs:3196F7A2h
lea     rcx, [rbp+110h+var_F0]
sub     rax, 3155F242h
call    rax ; sub_410560
```

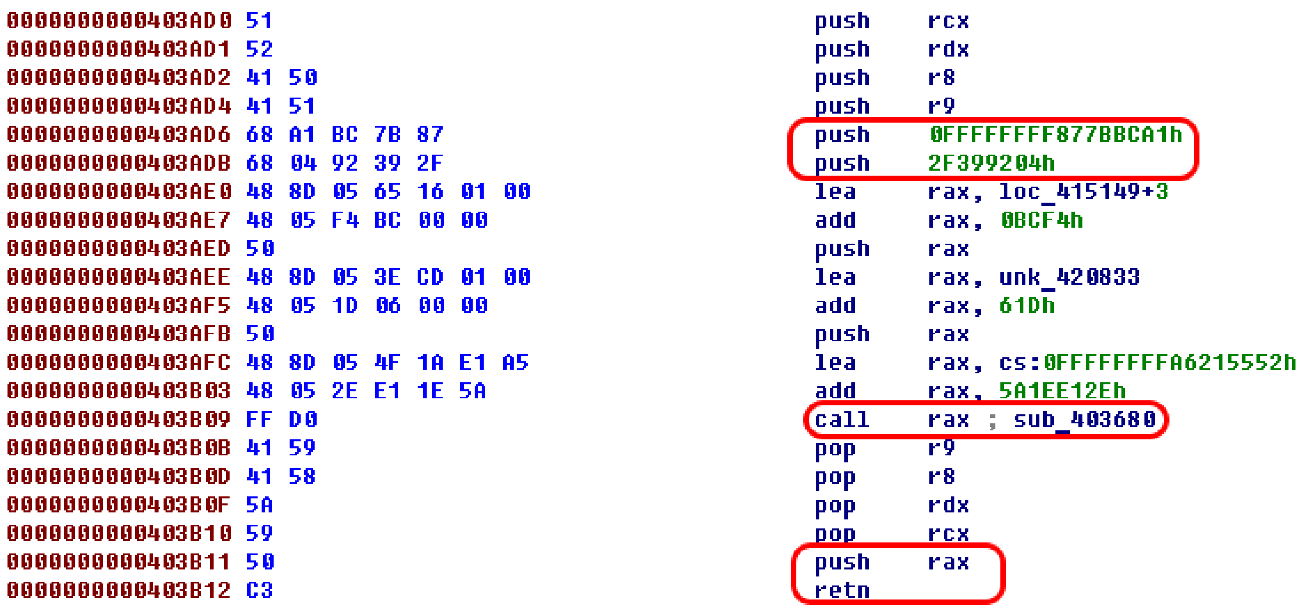
All of the WinAPI function addresses are imported dynamically by parsing the exports of system DLLs and calculating a CRC32-based hash of the function name. This in itself is neither new nor particularly difficult to analyze. However, the developers of SynAck further complicated this approach by obscuring both the address of the procedure that retrieves the API function address, and the target hash value.

Let's illustrate in detail how SynAck calls WinAPI functions. Consider the following piece of disassembly:



```
mov     rax, cs:off_403B13
lea     rcx, qword_423BB8
sub     rax, 78F5EC4Dh       off_403B13      dq 7936271Dh
mov     [rsp+48h+var_28], 0
mov     r9d, 1
xor     r8d, r8d
xor     edx, edx
call    rax
```

This code takes the DWORD located at 403b13, subtracts the constant 78f5ec4d, with the result 403ad0, and calls the procedure at this address.



```
0000000000403AD0 51                      push    rcx
0000000000403AD1 52                      push    rdx
0000000000403AD2 41 50                   push    r8
0000000000403AD4 41 51                   push    r9
0000000000403AD6 68 A1 BC 7B 87          push    0FFFFFFFF877BBCA1h
0000000000403ADB 68 04 92 39 2F          push    2F399204h
0000000000403AE0 48 8D 05 65 16 01 00    lea     rax, loc_415149+3
0000000000403AE7 48 05 F4 BC 00 00       add     rax, 0BCF4h
0000000000403AED 50                      push    rax
0000000000403AEE 48 8D 05 3E CD 01 00    lea     rax, unk_420833
0000000000403AF5 48 05 1D 06 00 00       add     rax, 61Dh
0000000000403AFB 50                      push    rax
0000000000403AFC 48 8D 05 4F 1A E1 A5    lea     rax, cs:0FFFFFFFFA6215552h
0000000000403B03 48 05 2E E1 1E 5A       add     rax, 5A1EE12Eh
0000000000403B09 FF D0                   call    rax ; sub_403680
0000000000403B0B 41 59                   pop     r9
0000000000403B0D 41 58                   pop     r8
0000000000403B0F 5A                      pop     rdx
0000000000403B10 59                      pop     rcx
0000000000403B11 50                      push    rax
0000000000403B12 C3                      retn
```

This procedure pushes two constants (N1 = ffffffff877bbca1 and N2 = 2f399204) onto the stack and passes the execution to the procedure at 403680 which will calculate the result of N1 xor N2 = a8422ea5.

This value is the hash of the API function name that SynAck wants to call. The procedure 403680 will then find the address of this function by parsing the export tables of system DLLs, calculating the hash of each function name and comparing it to the value a8422ea5. When this API function address is found, SynAck will pass the execution to this address.

Notice that instead of a simple CALL in the image above it uses the instructions PUSH + RET which is another attempt to complicate analysis. The developers of SynAck use different instruction combinations instead of CALL when calling WinAPI functions:

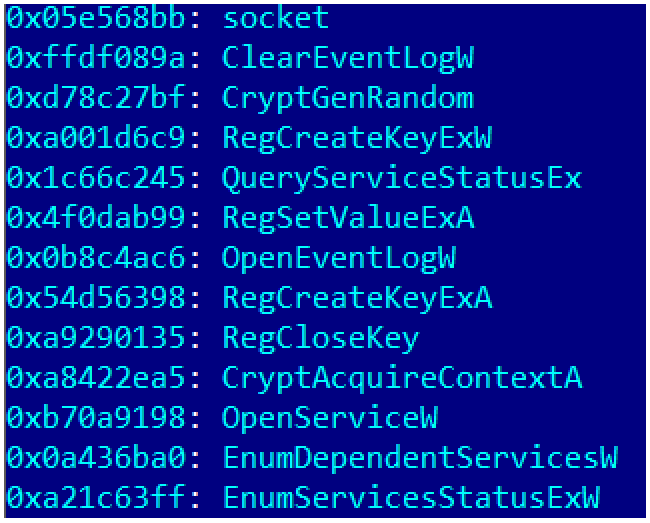
-
  ```
  push reg
  retn
  ```
- jmp reg

-
  ```
  mov [rsp-var], reg
  jmp qword ptr [rsp-var]
  ```

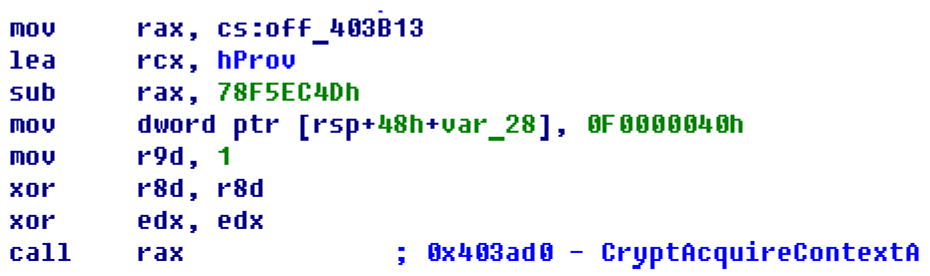
Deobfuscation

To counter these attempts by the malware developers, we created an IDAPython script that automatically parses the code, extracts the addresses of all intermediate procedures, extracts the constants and calculates the hashes of the WinAPI functions that the malware wants to import.

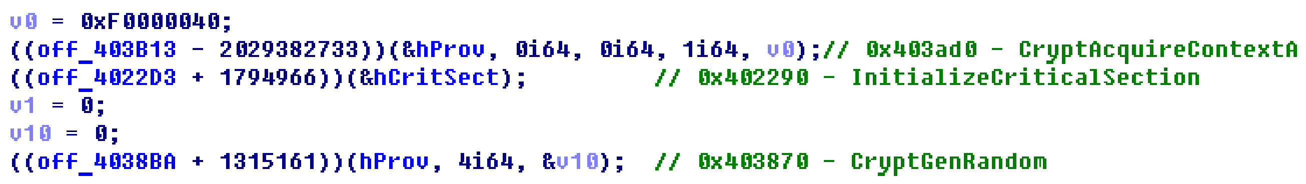
We then calculated the hash values of the functions exported from Windows system DLLs and matched them against the values required by SynAck. The result was a list showing which hash value corresponds to which API function.


```
0x05e568bb: socket
0xffdf089a: ClearEventLogW
0xd78c27bf: CryptGenRandom
0xa001d6c9: RegCreateKeyExW
0x1c66c245: QueryServiceStatusEx
0x4f0dab99: RegSetValueExA
0x0b8c4ac6: OpenEventLogW
0x54d56398: RegCreateKeyExA
0xa9290135: RegCloseKey
0xa8422ea5: CryptAcquireContextA
0xb70a9198: OpenServiceW
0x0a436ba0: EnumDependentServicesW
0xa21c63ff: EnumServicesStatusExW
```

Our script then uses this list to save comments in the IDA database to indicate which API is going to be called by the Trojan. Here is the code from the example above after deobfuscation.



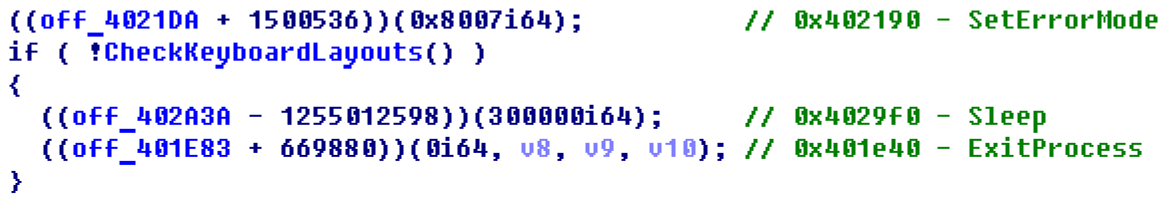
```
mov     rax, cs:off_403B13
lea     rcx, hProv
sub     rax, 78F5EC4Dh
mov     dword ptr [rsp+48h+var_28], 0F0000040h
mov     r9d, 1
xor     r8d, r8d
xor     edx, edx
call    rax                 ; 0x403ad0 - CryptAcquireContextA
```



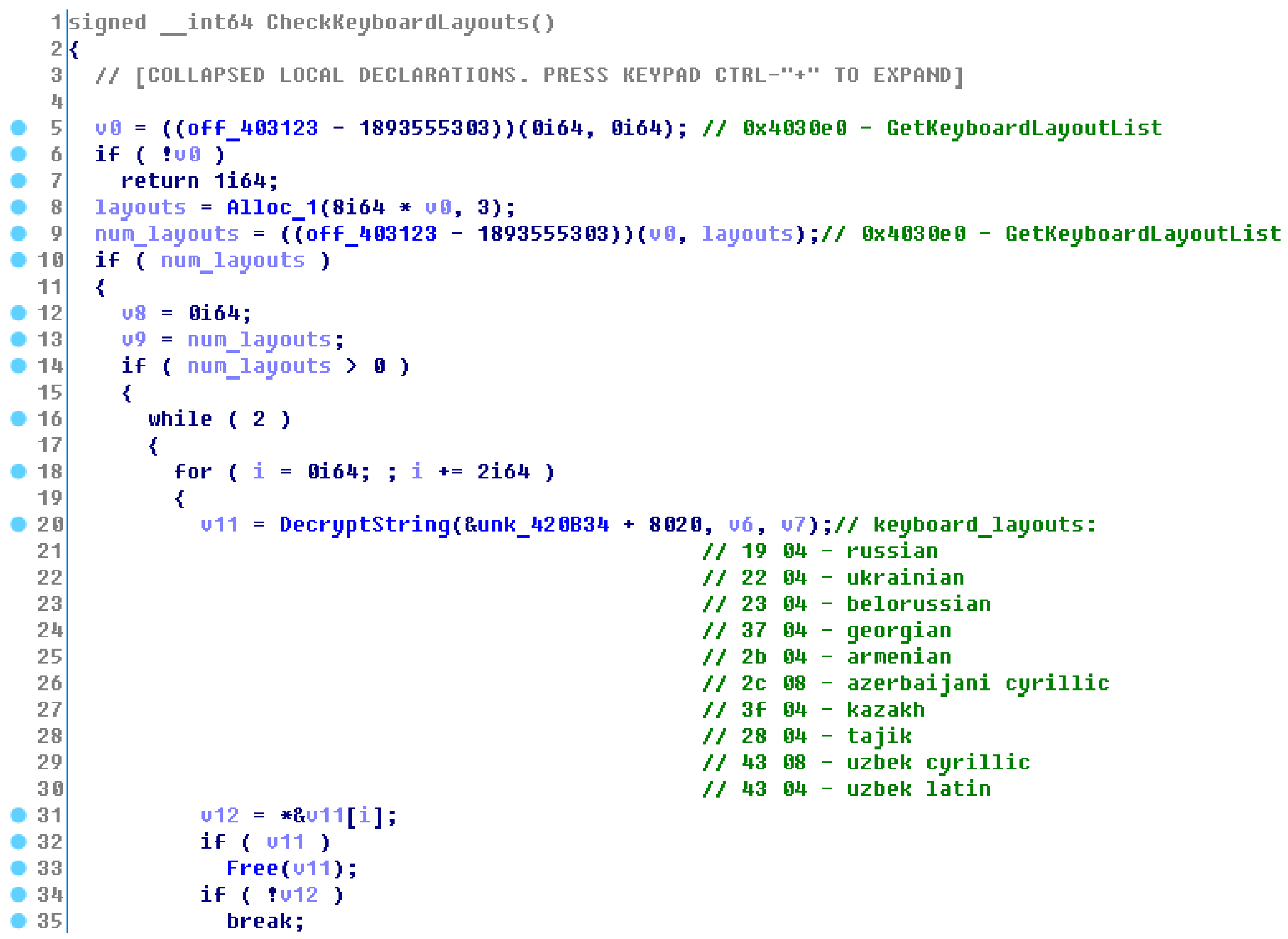
```
v0 = 0xF0000040;
((off_403B13 - 2029382733))(&hProv, 0i64, 0i64, 1i64, v0);// 0x403ad0 - CryptAcquireContextA
((off_4022D3 + 1794966))(&hCritSect);         // 0x402290 - InitializeCriticalSection
v1 = 0;
v10 = 0;
((off_4038BA + 1315161))(hProv, 4i64, &v10);  // 0x403870 - CryptGenRandom
```

Language check

At an early stage of execution the Trojan performs a check to find out whether it has been launched on a PC from a certain list of countries. To do this, it lists all the keyboard layouts installed on the victim’s PC and checks against a list hardcoded into the malware body. If it finds a match, SynAck sleeps for 300 seconds and then just calls ExitProcess to prevent encryption of files belonging to a victim from these countries.



```
((off_4021DA + 1500536))(0x8007i64);          // 0x402190 - SetErrorMode
if ( !CheckKeyboardLayouts() )
{
  ((off_402A3A - 1255012598))(300000i64);     // 0x4029f0 - Sleep
  ((off_401E83 + 669880))(0i64, v8, v9, v10); // 0x401e40 - ExitProcess
}
```



```
signed __int64 CheckKeyboardLayouts()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v0 = ((off_403123 - 1893555303))(0i64, 0i64); // 0x4030e0 - GetKeyboardLayoutList
  if ( !v0 )
    return 1i64;
  layouts = Alloc_1(8i64 * v0, 3);
  num_layouts = ((off_403123 - 1893555303))(v0, layouts);// 0x4030e0 - GetKeyboardLayoutList
  if ( num_layouts )
  {
    v8 = 0i64;
    v9 = num_layouts;
    if ( num_layouts > 0 )
    {
      while ( 2 )
      {
        for ( i = 0i64; ; i += 2i64 )
        {
          v11 = DecryptString(&unk_420B34 + 8020, v6, v7);// keyboard_layouts:
                                                       // 19 04 - russian
                                                       // 22 04 - ukrainian
                                                       // 23 04 - belorussian
                                                       // 37 04 - georgian
                                                       // 2b 04 - armenian
                                                       // 2c 08 - azerbaijani cyrillic
                                                       // 3f 04 - kazakh
                                                       // 28 04 - tajik
                                                       // 43 08 - uzbek cyrillic
                                                       // 43 04 - uzbek latin
          v12 = *&v11[i];
          if ( v11 )
            Free(v11);
          if ( !v12 )
            break;
```

Directory name validation

Shortly after the language check, which can be considered fairly common among modern ransomware, SynAck performs a check on the directory where its executable is started from. If there's an attempt to launch it from an 'incorrect' directory, the Trojan won't proceed and will just exit instead. This measure has been added by the malware developers to counter automatic sandbox analysis.

As with API imports, the Trojan doesn't store the strings it wants to check; instead it stores their hashes – a tactic that hinders efforts to find the original strings.

SynAck contains nine hashes; we have been able to brute-force two of them:

```
0x05f9053d == hash("output")
0x2cd2f8e2 == hash("plugins")
```

In the process we found a lot of collisions (gibberish strings that give the same hash value as the meaningful ones).

Cryptographic scheme

Like other ransomware, SynAck uses a combination of symmetric and asymmetric encryption algorithms. At the core of the SynAck algorithm lies the hybrid ECIES scheme. It is composed of 'building blocks' which interact with each other: ENC (symmetric encryption algorithm), KDF (key derivation function), and MAC (message authentication code). The ECIES scheme can be implemented using different building blocks. To calculate a key for the symmetric algorithm ENC, this scheme employs the ECDH protocol (Diffie-Hellman over a chosen elliptic curve).

The developers of this Trojan chose the following implementation:

ENC: XOR

KDF: PBKDF2-SHA1 with one iteration

MAC: HMAC-SHA1

ECDH curve: standard NIST elliptic curve **secp192r1**

ECIES-XOR-HMAC-SHA1

This is the function that implements the ECIES scheme in the SynAck sample.

Input: **plaintext, input_public_key**
Output: **ciphertext, ecies_public_key, MAC**

1. The Trojan generates a pair of asymmetric keys: **ecies_private_key** and **ecies_public_key**;
2. Using the generated **ecies_private_key** and **input_public_key** the Trojan calculates the shared secret according to the Diffie-Hellman protocol on an elliptic curve: `ecies_shared_secret = ECDH(ecies_private_key, input_public_key)`
3. Using the PBKDF2-SHA1 function with one iteration, the Trojan derives two byte arrays, **key_enc** and **key_mac**, from **ecies_shared_secret**. The size of key_enc is equal to the size of the plaintext;
4. The plaintext is XORed byte to byte with the **key_enc**;
5. The Trojan calculates the MAC (message authentication code) of the obtained ciphertext using the algorithm HMAC-SHA1 with **key_mac** as the key.

Initialization

At the first step the Trojan generates a pair of private and public keys: the private key (**session_private_key**) is a 192-bit random number and the public key (**session_public_key**) is a point on the standard NIST elliptic curve **secp192r1**.

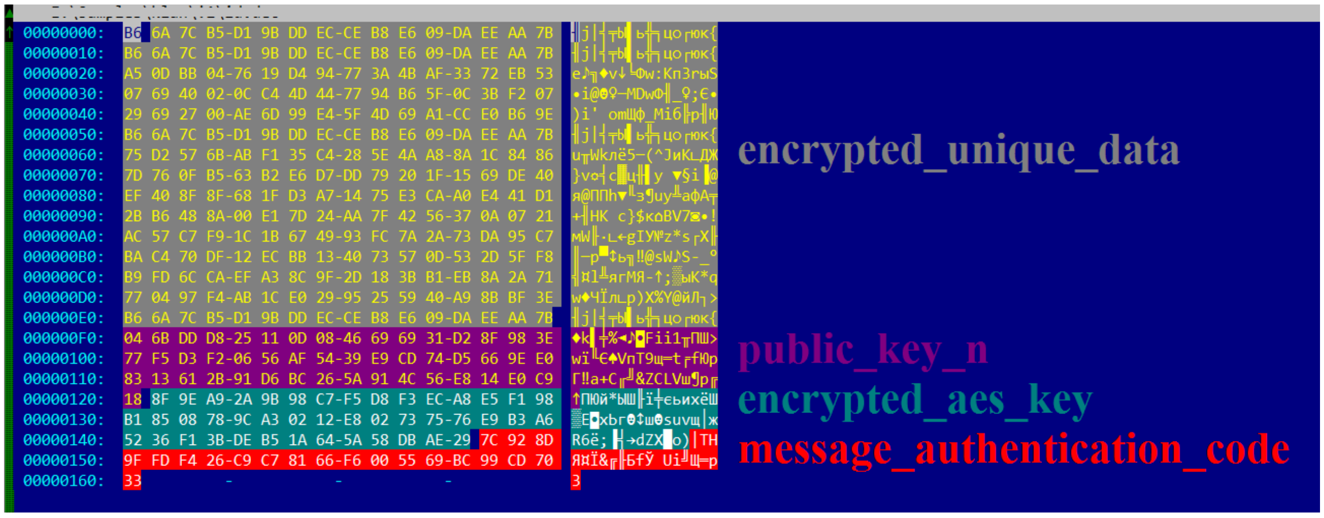
Then the Trojan gathers some unique information such as computer and user names, OS version info, unique infection ID, session private key and some random data and encrypts it using a randomly generated 256-bit AES key. The encrypted data is saved as the **encrypted_unique_data** buffer.

To encrypt the AES key, the Trojan uses the ECIES-XOR-HMAC-SHA1 function (see description above; hereafter referred to as the ECIES function). SynAck passes the AES key as the *plaintext* parameter and the hardcoded cybercriminal's **master_public_key** as *input_public_key*. The field *encrypted_aes_key* contains the ciphertext returned by the function, *public_key_n* is the ECIES public key and *message_authentication_code* is the MAC.

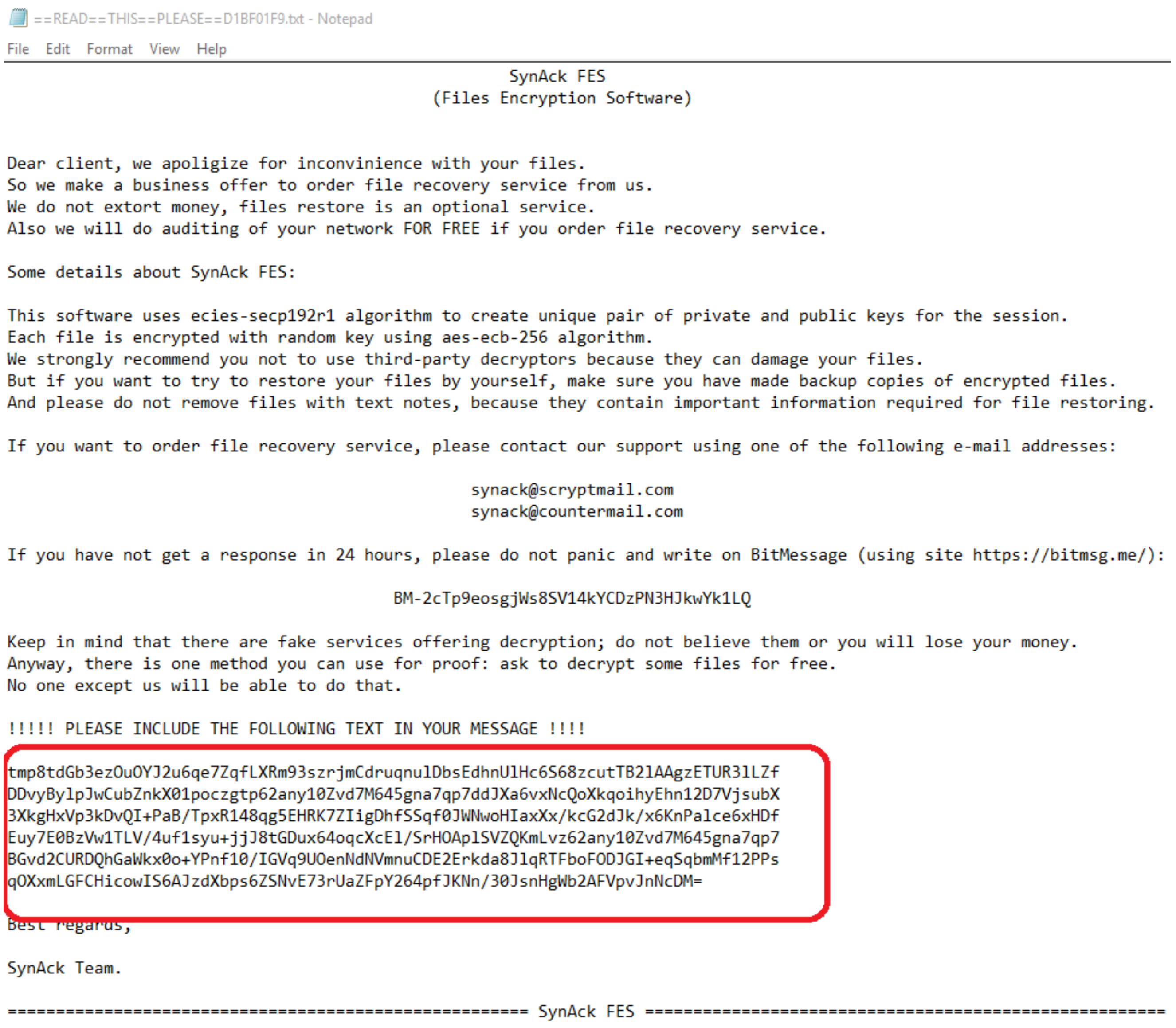
At the next step the Trojan forms the structure *cipher_info*.

```
 struct cipher_info
{
uint8_t encrypted_unique_data[240];
uint8_t public_key_n[49];
uint8_t encrypted_aes_key[44];
uint8_t message_authentication_code[20];
};
```

It is shown in the image below.


```
00000000:  B6 6A 7C B5-D1 9B DD EC-CE B8 E6 09-DA EE AA 7B
00000010:  B6 6A 7C B5-D1 9B DD EC-CE B8 E6 09-DA EE AA 7B
00000020:  A5 0D BB 04-76 19 D4 94-77 3A 4B AF-33 72 EB 53
00000030:  07 69 40 02-0C C4 4D 44-77 94 B6 5F-0C 3B F2 07
00000040:  29 69 27 00-AE 6D 99 E4-5F 4D 69 A1-CC E0 B6 9E
00000050:  B6 6A 7C B5-D1 9B DD EC-CE B8 E6 09-DA EE AA 7B
00000060:  75 D2 57 6B-AB F1 35 C4-28 5E 4A A8-8A 1C 84 86
00000070:  7D 76 0F B5-63 B2 E6 D7-DD 79 20 1F-15 69 DE 40
00000080:  EF 40 8F 8F-68 1F D3 A7-14 75 E3 CA-A0 E4 41 D1
00000090:  2B B6 48 8A-00 E1 7D 24-AA 7F 42 56-37 0A 07 21
000000A0:  AC 57 C7 F9-1C 1B 67 49-93 FC 7A 2A-73 DA 95 C7
000000B0:  BA C4 70 DF-12 EC BB 13-40 73 57 0D-53 2D 5F F8
000000C0:  B9 FD 6C CA-EF A3 8C 9F-2D 18 3B B1-EB 8A 2A 71
000000D0:  77 04 97 F4-AB 1C E0 29-95 25 59 40-A9 8B BF 3E
000000E0:  B6 6A 7C B5-D1 9B DD EC-CE B8 E6 09-DA EE AA 7B
000000F0:  04 6B DD D8-25 11 0D 08-46 69 69 31-D2 8F 98 3E
00000100:  77 F5 D3 F2-06 56 AF 54-39 E9 CD 74-D5 66 9E E0
00000110:  83 13 61 2B-91 D6 BC 26-5A 91 4C 56-E8 14 E0 C9
00000120:  18 8F 9E A9-2A 9B 98 C7-F5 D8 F3 EC-A8 E5 F1 98
00000130:  B1 85 08 78-9C A3 02 12-E8 02 73 75-76 E9 B3 A6
00000140:  52 36 F1 3B-DE B5 1A 64-5A 58 DB AE-29 7C 92 8D
00000150:  9F FD F4 26-C9 C7 81 66-F6 00 55 69-BC 99 CD 70
00000160:  33          -           -           -
```

This data is then encoded in base64 and written into the ransom note.


==READ==THIS==PLEASE==D1BF01F9.txt - Notepad
File Edit Format View Help

```
                                            SynAck FES
                                    (Files Encryption Software)


Dear client, we apoligize for inconvinience with your files.
So we make a business offer to order file recovery service from us.
We do not extort money, files restore is an optional service.
Also we will do auditing of your network FOR FREE if you order file recovery service.

Some details about SynAck FES:

This software uses ecies-secp192r1 algorithm to create unique pair of private and public keys for the session.
Each file is encrypted with random key using aes-ecb-256 algorithm.
We strongly recommend you not to use third-party decryptors because they can damage your files.
But if you want to try to restore your files by yourself, make sure you have made backup copies of encrypted files.
And please do not remove files with text notes, because they contain important information required for file restoring.

If you want to order file recovery service, please contact our support using one of the following e-mail addresses:

                                      synack@scryptmail.com
                                      synack@countermail.com

If you have not get a response in 24 hours, please do not panic and write on BitMessage (using site https://bitmsg.me/):

                                BM-2cTp9eosgjWs8SV14kYCDzPN3HJkwYk1LQ

Keep in mind that there are fake services offering decryption; do not believe them or you will lose your money.
Anyway, there is one method you can use for proof: ask to decrypt some files for free.
No one except us will be able to do that.

!!!!! PLEASE INCLUDE THE FOLLOWING TEXT IN YOUR MESSAGE !!!!

tmp8tdGb3ezOuOYJ2u6qe7ZqfLXRm93szrjmCdruqnulDbsEdhnUlHc6S68zcutTB2lAAgzETUR3lLZf
DDvyBylpJwCubZnkX01poczgtp62any10Zvd7M645gna7qp7ddJXa6vxNcQoXkqoihyEhn12D7VjsubX
3XkgHxVp3kDvQI+PaB/TpxR148qg5EHRK7ZIigDhfSSqf0JWNwoHIaxXx/kcG2dJk/x6KnPalce6xHDf
Euy7E0BzVw1TLV/4uf1syu+jjJ8tGDux64oqcXcEl/SrHOAplSVZQKmLvz62any10Zvd7M645gna7qp7
BGvd2CURDQhGaWkx0o+YPnf10/IGVq9UOenNdNVmnuCDE2Erkda8J1qRTFboFODJGI+eqSqbmMf12PPs
qOXxmLGFCHicowIS6AJzdXbps6ZSNvE73rUaZFpY264pfJKNn/30JsnHgWb2AFVpvJnNcDM=

Best regards,

SynAck Team.

==================================================== SynAck FES ====================================================
```

As we can see, the criminals ask the victim to include this encoded text in their message.

File encryption

The content of each file is encrypted by the AES-256-ECB algorithm with a randomly generated key. After encryption, the Trojan forms a structure containing information such as the encryption label 0xA4EF5C91, the used AES key, encrypted chunk size and the original file name. This information can be represented as a structure:

```
 struct encryption_info
{
uint32_t label = 0xA4EF5C91;
uint8_t aes_key[32];
uint32_t encrypted_chunk_size;
uint32_t reserved;
uint8_t original_name_buffer[522];
};
```

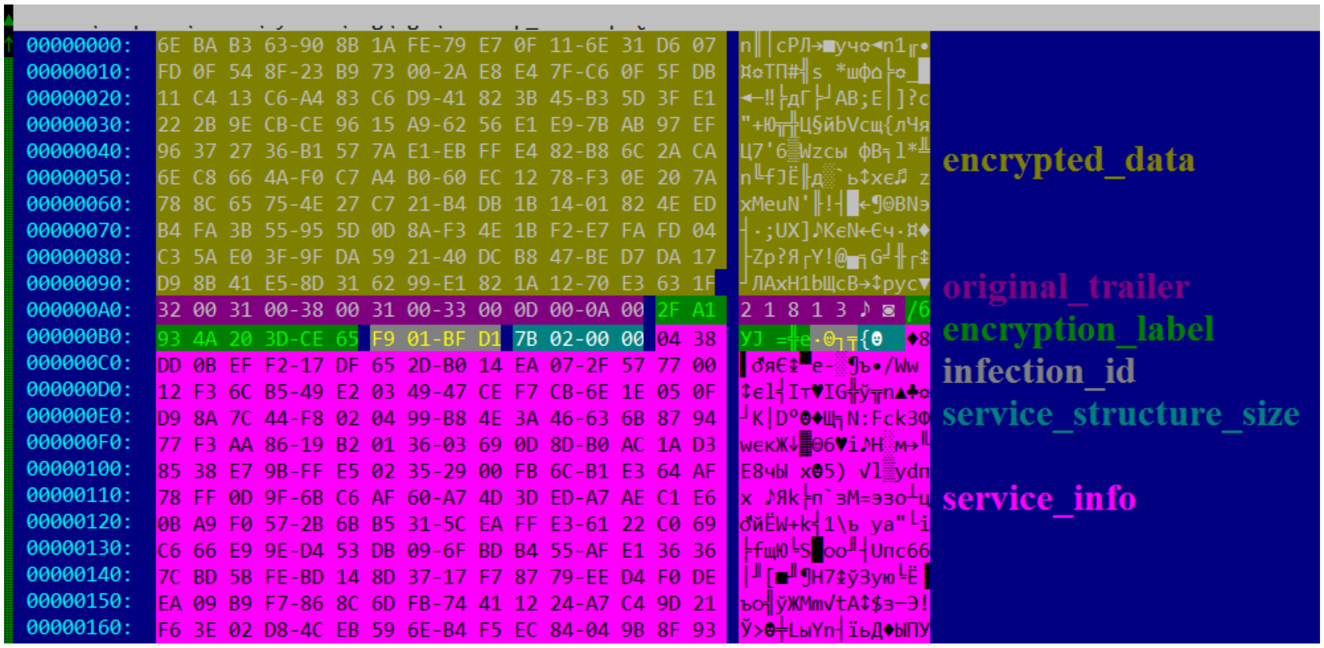
The Trojan then calls the ECIES function and passes the *encryption_info* structure as the *plaintext* and the previously generated **session_public_key** as the *input_public_key*. The result returned by this function is saved into a structure which we dubbed *file_service_structure*. The field *encrypted_file_info* contains the ciphertext returned by the function, *ecc_file_key_public* is the ECIES public key and *message_authentication_code* is the MAC.

```
 struct file_service_structure
{
uint8_t ecc_file_key_public[49];
encryption_info encrypted_file_info;
uint8_t message_authentication_code[20];
};
```

This structure is written to the end of the encrypted file. This results in an encrypted file having the following structure:

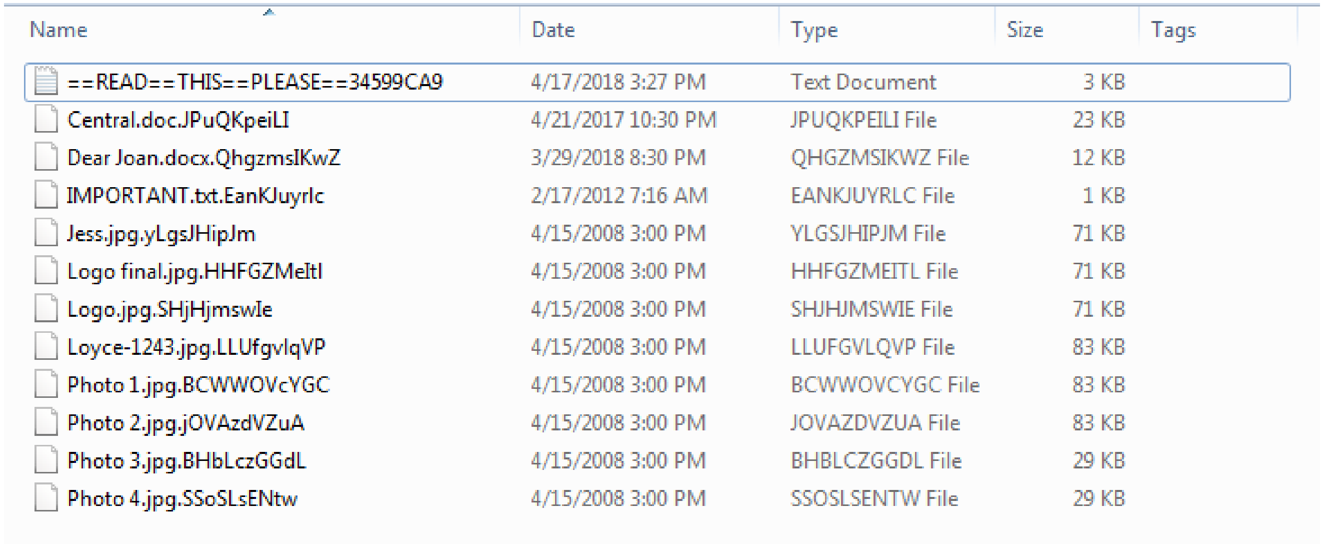
```
 struct encrypted_file
{
uint8_t encrypted_data[file_size - file_size % AES_BLOCK_SIZE];
uint8_t original_trailer[file_size % AES_BLOCK_SIZE];
uint64_t encryption_label = 0x65CE3D204A93A12F;
uint32_t infection_id;
uint32_t service_structure_size;
file_service_structure service_info;
};
```

The encrypted file structure is shown in the image below.



```
00000000: 6E BA B3 63-90 8B 1A FE-79 E7 0F 11-6E 31 D6 07  n║│cPЛ→■учо◄n1╓•
00000010: FD 0F 54 8F-23 B9 73 00-2A E8 E4 7F-C6 0F 5F DB  ¤☼TП#╣s *шф⌂╞☼_█
00000020: 11 C4 13 C6-A4 83 C6 D9-41 82 3B 45-B3 5D 3F E1  ◄─‼╞дГ╞┘AB;E│]?с
00000030: 22 2B 9E CB-CE 96 15 A9-62 56 E1 E9-7B AB 97 EF  "+Ю╥╬Ц§йbVсщ{лЧя
00000040: 96 37 27 36-B1 57 7A E1-EB FF E4 82-B8 6C 2A CA  Ц7'6▒Wzсы ФВ╕l*╩
00000050: 6E C8 66 4A-F0 C7 A4 B0-60 EC 12 78-F3 0E 20 7A  n╚fJЁ╟д░`ь↕xє♫ z
00000060: 78 8C 65 75-4E 27 C7 21-B4 DB 1B 14-01 82 4E ED  xMeuN'╟!┤█←¶☺BNэ
00000070: B4 FA 3B 55-95 5D 0D 8A-F3 4E 1B F2-E7 FA FD 04  ┤·;UX]♪KєN←Єч·¤♦
00000080: C3 5A E0 3F-9F DA 59 21-40 DC B8 47-BE D7 DA 17  ├Zр?Я┌Y!@▄╕G╛╫┌↨
00000090: D9 8B 41 E5-8D 31 62 99-E1 82 1A 12-70 E3 63 1F  ┘ЛAхН1bЩсВ→↕pус▼
000000A0: 32 00 31 00-38 00 31 00-33 00 0D 00-0A 00 2F A1  2 1 8 1 3 ♪ ◙ /б
000000B0: 93 4A 20 3D-CE 65 F9 01-BF D1 7B 02-00 00 04 38  УJ =╬e·☺┐╤{☻  ♦8
000000C0: DD 0B EF F2-17 DF 65 2D-B0 14 EA 07-2F 57 77 00  ▌♂яЄ↨▀e-░¶ъ•/Ww
000000D0: 12 F3 6C B5-49 E2 03 49-47 CE F7 CB-6E 1E 05 0F  ↕єl╡Iт♥IG╬ў╦n▲♣☼
000000E0: D9 8A 7C 44-F8 02 04 99-B8 4E 3A 46-63 6B 87 94  ┘K|D°☻♦Щ╕N:Fck3Ф
000000F0: 77 F3 AA 86-19 B2 01 36-03 69 0D 8D-B0 AC 1A D3  wєкЖ↓▓☺6♥i♪H░м→╙
00000100: 85 38 E7 9B-FF E5 02 35-29 00 FB 6C-B1 E3 64 AF  E8чЫ х☻5) √l▒ydп
00000110: 78 FF 0D 9F-6B C6 AF 60-A7 4D 3D ED-A7 AE C1 E6  x ♪Яk╞п`зM=эзо┴ц
00000120: 0B A9 F0 57-2B 6B B5 31-5C EA FF E3-61 22 C0 69  ♂йЁW+k╡1\ъ уa"└i
00000130: C6 66 E9 9E-D4 53 DB 09-6F BD B4 55-AF E1 36 36  ╞fщЮ╘S█○o╜┤Uпс66
00000140: 7C BD 5B FE-BD 14 8D 37-17 F7 87 79-EE D4 F0 DE  |╜[■╜¶H7↨ўЗyю╘Ё▐
00000150: EA 09 B9 F7-86 8C 6D FB-74 41 12 24-A7 C4 9D 21  ъ○╣ўЖMm√tA↕$з─Э!
00000160: F6 3E 02 D8-4C EB 59 6E-B4 F5 EC 84-04 9B 8F 93  Ў>☻╪LыYn┤їьД♦ЫПУ
```

After encryption the files will have randomly generated extensions.



Name	Date	Type	Size	Tags
==READ==THIS==PLEASE==34599CA9	4/17/2018 3:27 PM	Text Document	3 KB	
Central.doc.JPuQKpeiLI	4/21/2017 10:30 PM	JPUQKPEILI File	23 KB	
Dear Joan.docx.QhgzmsIKwZ	3/29/2018 8:30 PM	QHGZMSIKWZ File	12 KB	
IMPORTANT.txt.EanKJuyrlc	2/17/2012 7:16 AM	EANKJUYRLC File	1 KB	
Jess.jpg.yLgsJHipJm	4/15/2008 3:00 PM	YLGSJHIPJM File	71 KB	
Logo final.jpg.HHFGZMeItl	4/15/2008 3:00 PM	HHFGZMEITL File	71 KB	
Logo.jpg.SHjHjmswIe	4/15/2008 3:00 PM	SHJHJMSWIE File	71 KB	
Loyce-1243.jpg.LLUfgvlqVP	4/15/2008 3:00 PM	LLUFGVLQVP File	83 KB	
Photo 1.jpg.BCWWOVcYGC	4/15/2008 3:00 PM	BCWWOVCYGC File	83 KB	
Photo 2.jpg.jOVAzdVZuA	4/15/2008 3:00 PM	JOVAZDVZUA File	83 KB	
Photo 3.jpg.BHbLczGGdL	4/15/2008 3:00 PM	BHBLCZGGDL File	29 KB	
Photo 4.jpg.SSoSLsENtw	4/15/2008 3:00 PM	SSOSLSENTW File	29 KB	

Other features

Termination of processes and services

Prior to file encryption, SynAck enumerates all running processes and all services and checks the hashes of their names against two lists of hardcoded hash values (several hundred combined). If it finds a match, the Trojan will attempt to kill the process (using the TerminateProcess API function) or to stop the service (using ControlService with the parameter SERVICE_CONTROL_STOP).

To find out which processes it wants to terminate and which services to stop, we brute-forced the hashes from the Trojan body. Below are some of the results.

Processes		Services	
Hash	**Name**	**Hash**	**Name**
0x9a130164	dns.exe	0x11216a38	vss
0xf79b0775	lua.exe	0xe3f1f130	mysql
0x6475ad3c	mmc.exe	0xc82cea8d	qbvss
0xe107acf0	php.exe	0xebcd4079	sesvc
0xf7f811c4	vds.exe	0xf3d0e358	vmvss
0xcf96a066	lync.exe	0x31c3fbb6	wmsvc
0x167f833f	nssm.exe	0x716f1a42	w3svc

0x255c7041	ssms.exe	0xa6332453	memtas
0xbdcc75a9	w3wp.exe	0x82953a7a	mepocs
0x410de6a4	excel.exe		
0x9197b633	httpd.exe		
0x83ddb55a	ilsvc.exe		
0xb27761ed	javaw.exe		
0xfd8b9308	melsc.exe		
0xa105f60b	memis.exe		
0x10e94bcc	memta.exe		
0xb8de9e34	mepoc.exe		
0xeaa98593	monad.exe		
0x67181e9b	mqsvc.exe		
0xd6863409	msoia.exe		
0x5fcab0fe	named.exe		
0x7d171368	qbw32.exe		
0x7216db84	skype.exe		
0xd2f6ce06	steam.exe		
0x68906b65	store.exe		
0x6d6daa28	vksts.exe		
0x33cc148e	vssvc.exe		
0x26731ae9	conime.exe		
0x76384ffe	fdhost.exe		
0x8cc08bd7	mepopc.exe		
0x2e883bd5	metray.exe		
0xd1b5c8df	mysqld.exe		
0xd2831c37	python.exe		

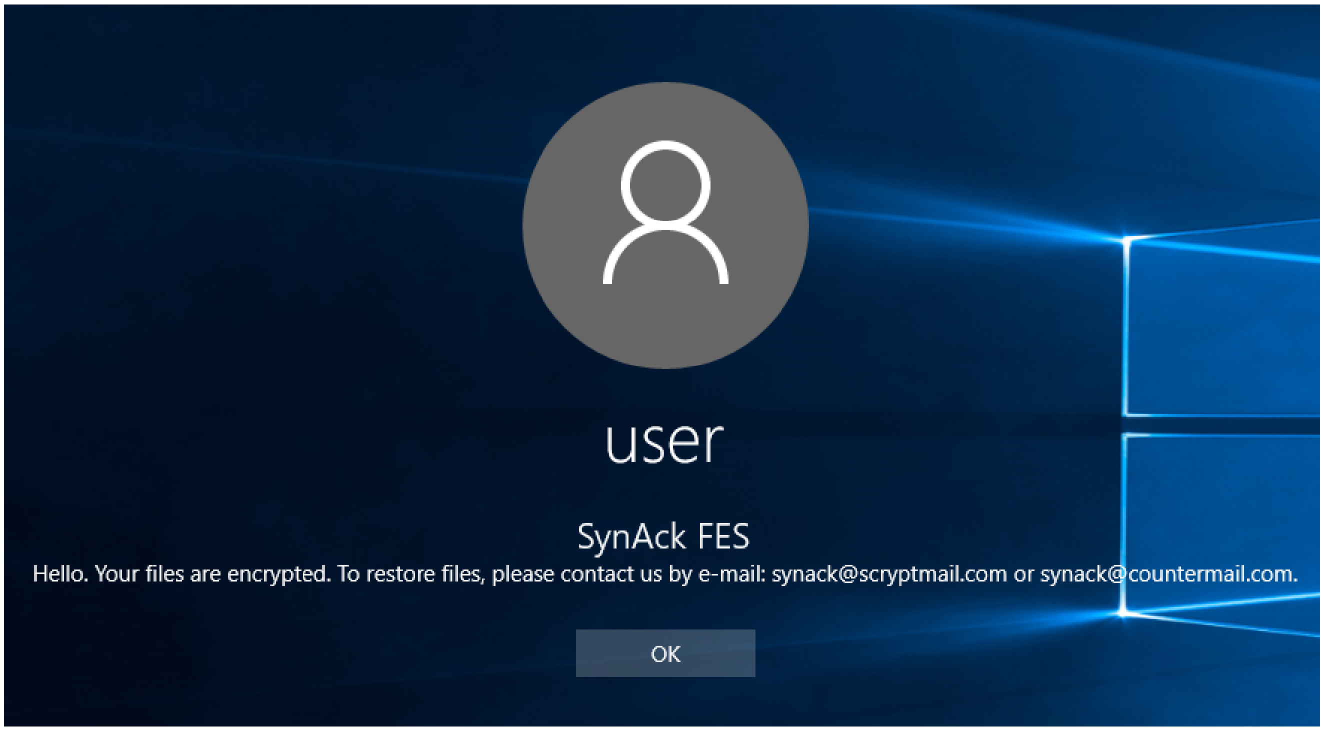
0xf7dc2e4e	srvany.exe
0x8a37ebfa	tabtip.exe

As we can see, SynAck seeks to stop programs related to virtual machines, office applications, script interpreters, database applications, backup systems, gaming applications and so on. It might be doing this to grant itself access to valuable files that could have been otherwise used by the running processes.

Clearing the event logs

To impede possible forensic analysis of an infected machine, SynAck clears the event logs stored by the system. To do so, it uses two approaches. For Windows versions prior to Vista, it enumerates the registry key SYSTEM\CurrentControlSet\Services\EventLog and uses OpenEventLog/ClearEventLog API functions. For more modern Windows versions, it uses the functions from EvtOpenChannelEnum/EvtNextChannelPath/EvtClearLog and from Wevtapi.dll.

SynAck is also capable of adding a custom text to the Windows logon screen. It does this by modifying the LegalNoticeCaption and LegalNoticeText keys in the registry. As a result, before the user signs in to their account, Windows shows a message from the cybercriminals.


user

SynAck FES

Hello. Your files are encrypted. To restore files, please contact us by e-mail: synack@scryptmail.com or synack@countermail.com.

Attack statistics

We have currently only observed several attacks in the USA, Kuwait, Germany, and Iran. This leads us to believe that this is targeted ransomware.

Detection verdicts

Trojan-Ransom.Win32.Agent.abwa
Trojan-Ransom.Win32.Agent.abwb
PDM:Trojan.Win32.Generic

IoCs

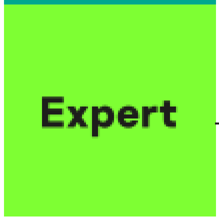
0x6F772EB660BC05FC26DF86C98CA49ABC
0x911D5905CBE1DD462F171B7167CD15B9

- Malware Descriptions
- Malware Technologies
- Obfuscation
- Ransomware
- Trojan

Authors

-

 Anton Ivanov
-

 Fedor Sinitsyn
- 

 Orkhan Mamedov

SynAck targeted ransomware uses the Doppelgänging technique

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