Inside Chimera Ransomware – the first ‘doxingware’ in wild

By hasherezade

Published: 2015-12-08 · Archived: 2026-04-10 03:03:51 UTC

Ransomware have proven to be a good source of money for cybercriminals. Not surprisingly, we are nowadays facing various families of this type of malware, i.e Cryptowall, CTB-Locker,Teslacrypt to name a few.

Recently, one more joined this set: Chimera, that is distributed via targeted e-mails to small companies.

At the first sight, it appears like yet another malware encrypting user’s private files and demanding ransom for decrypting it. But it added to this feature one more twist that is supposed to put more pressure on the victim. It threatens that in case if the ransom will not be paid, all the stolen files are going to be published, along with the stolen credentials allowing to identify files’ owner*.

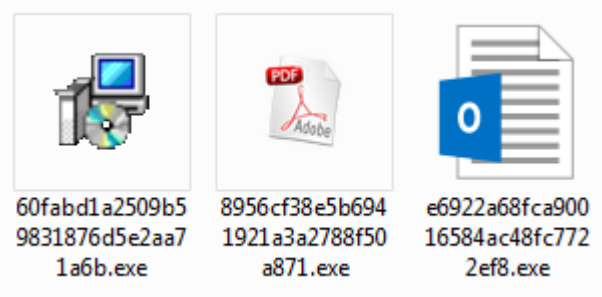
This blackmail technique, called doxing provides much more serious threat than just loosing access to files. Also, keeping backups, that helps to manage attack of a normal ransomware would not help much. I think it will be fair to make a new term to describe this new subtype of ransomware, for example: doxingware.

*- *Fortunately in case of Chimera authors didn’t decided to really upload the files on the server, so it is only a bogus threat. Yet, from the point of view of the victim the pressure is very real.*

This time we will take a high and low level view at Chimera, in order to understand the techniques used.

Analyzed samples

I will base on following three samples, captured by @JAMESWT_MHT // *– big thanks to him for sharing!* 🙂

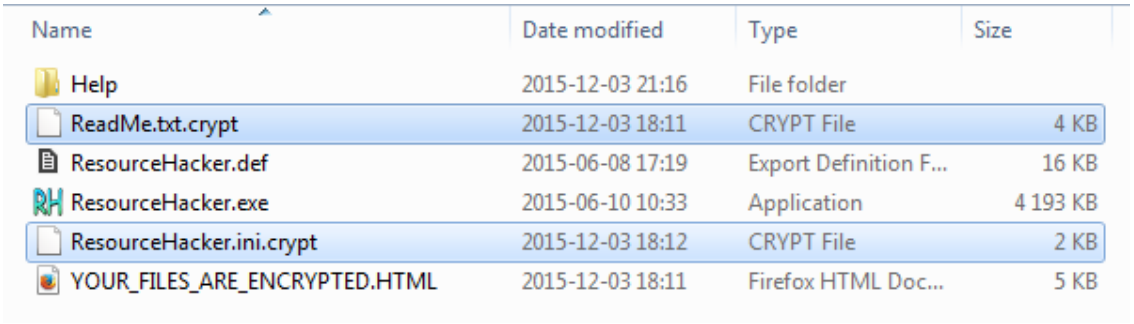


- 8956cf38e5b6941921a3a2788f50a871
- e6922a68fca90016584ac48fc7722ef8
- 60fabd1a2509b59831876d5e2aa71a6b – ***Stub.exe*** <- chosen as the main object of the analysis
  - 8df3534fe1ae95fc8c22cb85aed15336 – payload (***Loader.dll***)
    - 0a27affc77bd786beff69aa1f502d694 – payload (***Core.dll***)

Behavioral analysis

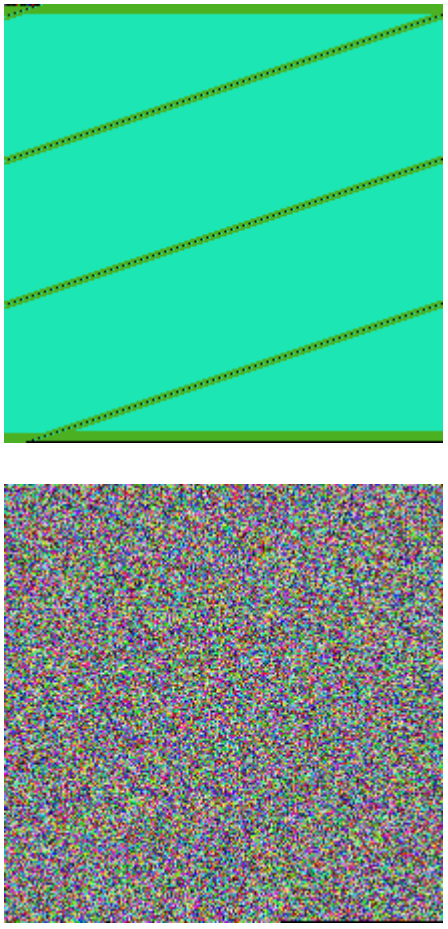
When deployed, Chimera enumerates files on all the available disks and encrypts documents recognized by some predefined extensions.

This is an example of what you may see, if on your machine Chimera was deployed – some files have been substituted by their encrypted versions with the appended extension ***.crypt***.



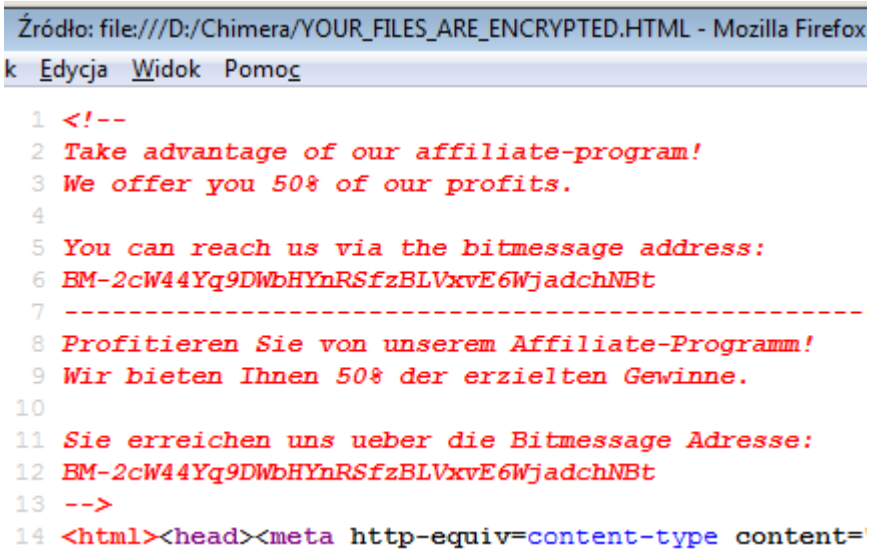
See below a visualization of bytes.

square.bmp : left – original, right encrypted with *Chimera*:



Also, there is an HTML file dropped, that teaches user what happened. The HTML can be displayed in two languages – English and German. Below the English version:

At the bottom of the HTML file we can read that, in addition to blackmail, attackers also search people willingly to cooperate – probably for franchising their criminal business. More info about it is available in the source of the HTML:

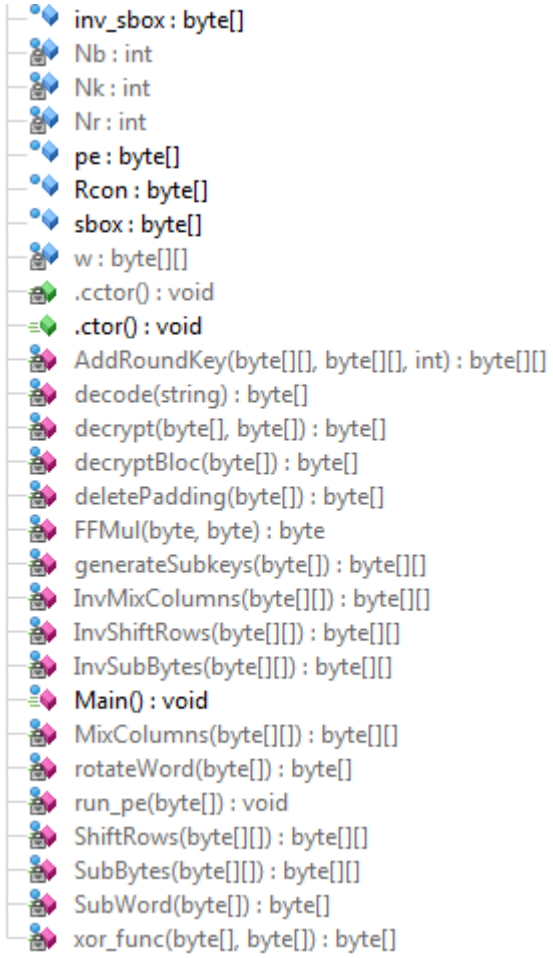


After the process of encryption of all the files is finished, this HTML is displayed in full screen mode via Internet Explorer.

Unpacking

Two out of three malicious samples (60fabd1a2509b59831876d5e2aa71a6b, e6922a68fca90016584ac48fc7722ef8) are packed by the same .NET crypter, so I decided to give a brief overview on unpacking this crypter.

It is not obfuscated and can be easily decompiled by typical tools i.e. ILSpy. Looking at function names, we can get a lot of information about the functionality, i.e it loads the payload by the RunPE technique:



(full Stub.cs: https://gist.github.com/hasherezade/5b742b46df4f79fdb784)

[code language="csharp" title="Stub.cs" firstline="600"] public static void Main() { byte[] rawAssembly = Stub.decrypt(Stub.pe, Stub.decode(BASE64_ENCODED_KEY)); Stub.run_pe(rawAssembly); }

private static void run_pe(byte[] rawAssembly) { new Stub.ManualMap().LoadLibrary(rawAssembly); } [/code]

This author of the crypter didn't relied on simple XOR based algorithm – instead, provided a custom implementation of a block cipher (Rijndael). We can find variables with familiar names like: sbox, inv_sbox (inverse S-Box), Rcon (the Round Constant), Nr, Nb, Nk… Fragment:

```
// Stub private static byte[] decrypt(byte[] input, byte[] key) {       byte[] array = new byte[inpu
```

Payloads

Loader.dll

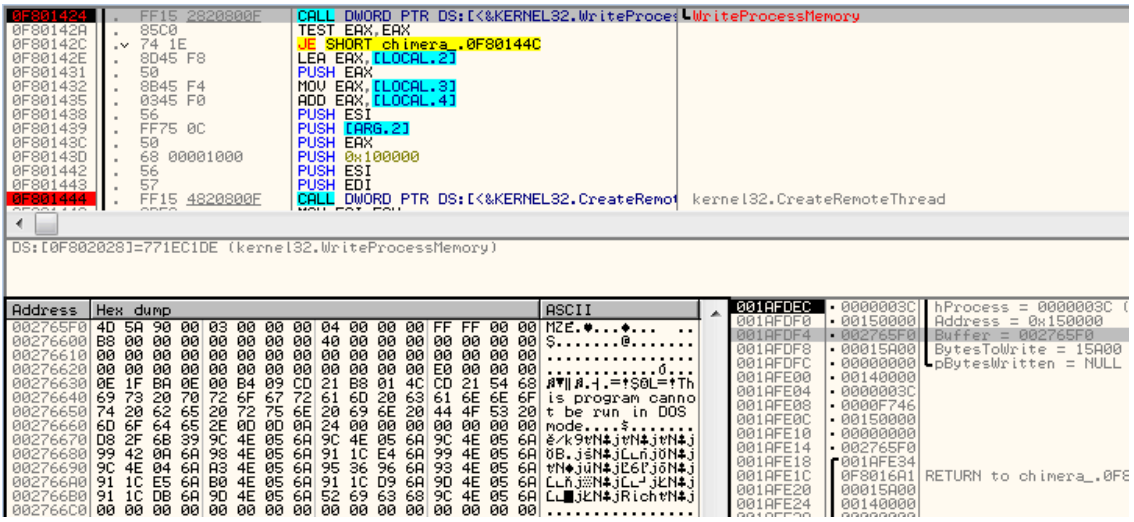
md5 = 8df3534fe1ae95fc8c22cb85aed15336

The file unpacked by Stub.exe is a DLL. It comes with a string referring to a database with debug symbols of the project, suggesting that it is not the core payload, but just a loader for it:

C:ProjectsRansombinReleaseLoader.pdb. In fact, it role is just to unpack and load the core executable.

Automatic analysis: https://malwr.com/analysis/Zjc0MDg0ZmRlMjhkNGYxZTlmZWI1NzIxMTlhYmEyODU/

Loader.dll unpacks a new PE file, writes into process memory and runs it in a new thread:



Core.dll

md5 = 0a27affc77bd786beff69aa1f502d694

The original name of the executable unpacked by the Loader is ***Core.dll*** (it also comes with a analogical string: ***C:ProjectsRansombinReleaseCore.pdb***) and is responsible for all the malicious activities.

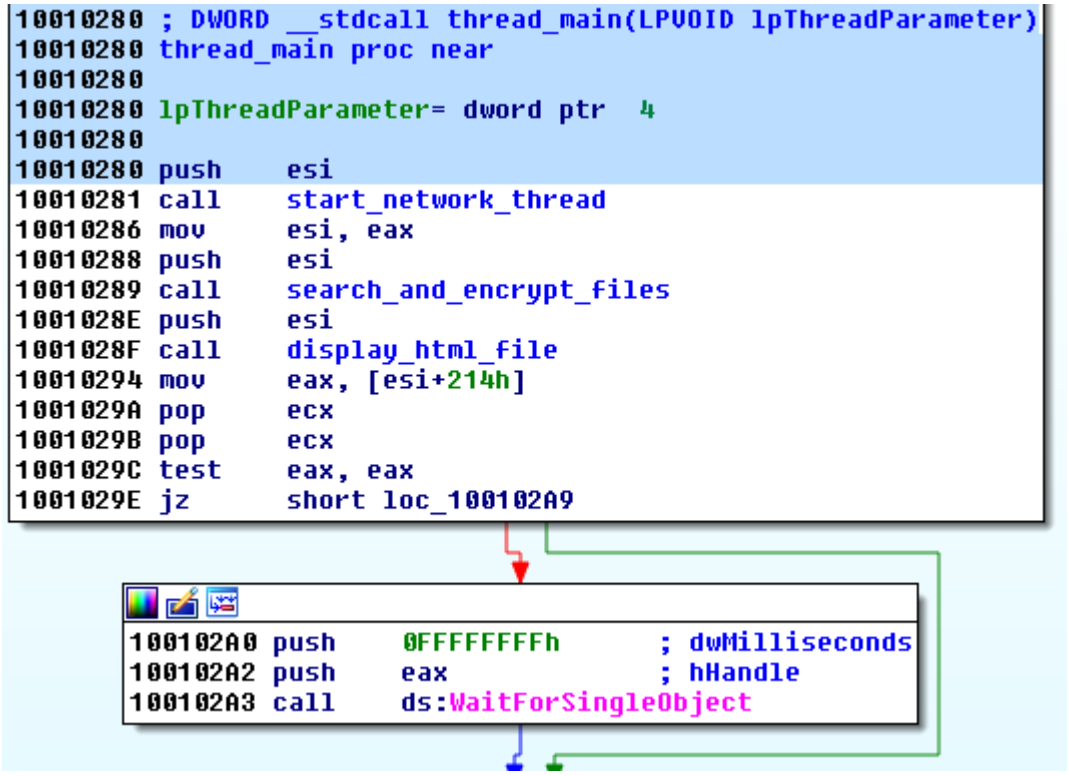
Offset	Name	Value	Meaning
148D0	Characteristics	0	
148D4	TimeDateStamp	55F9CDCC	
148D8	MajorVersion	0	
148DA	MinorVersion	0	
148DC	Name	15702	Core.dll
148E0	Base	1	
148E4	NumberOfFunctions	1	
148E8	NumberOfNames	1	
148EC	AddressOfFunctions	156F8	
148F0	AddressOfNames	156FC	
148F4	AddressOfNameOrdinals	15700	

Details

Offset	Ordinal	Function RVA	Name RVA	Name
148F8	1	10346	1570B	_ReflectiveLoader@4

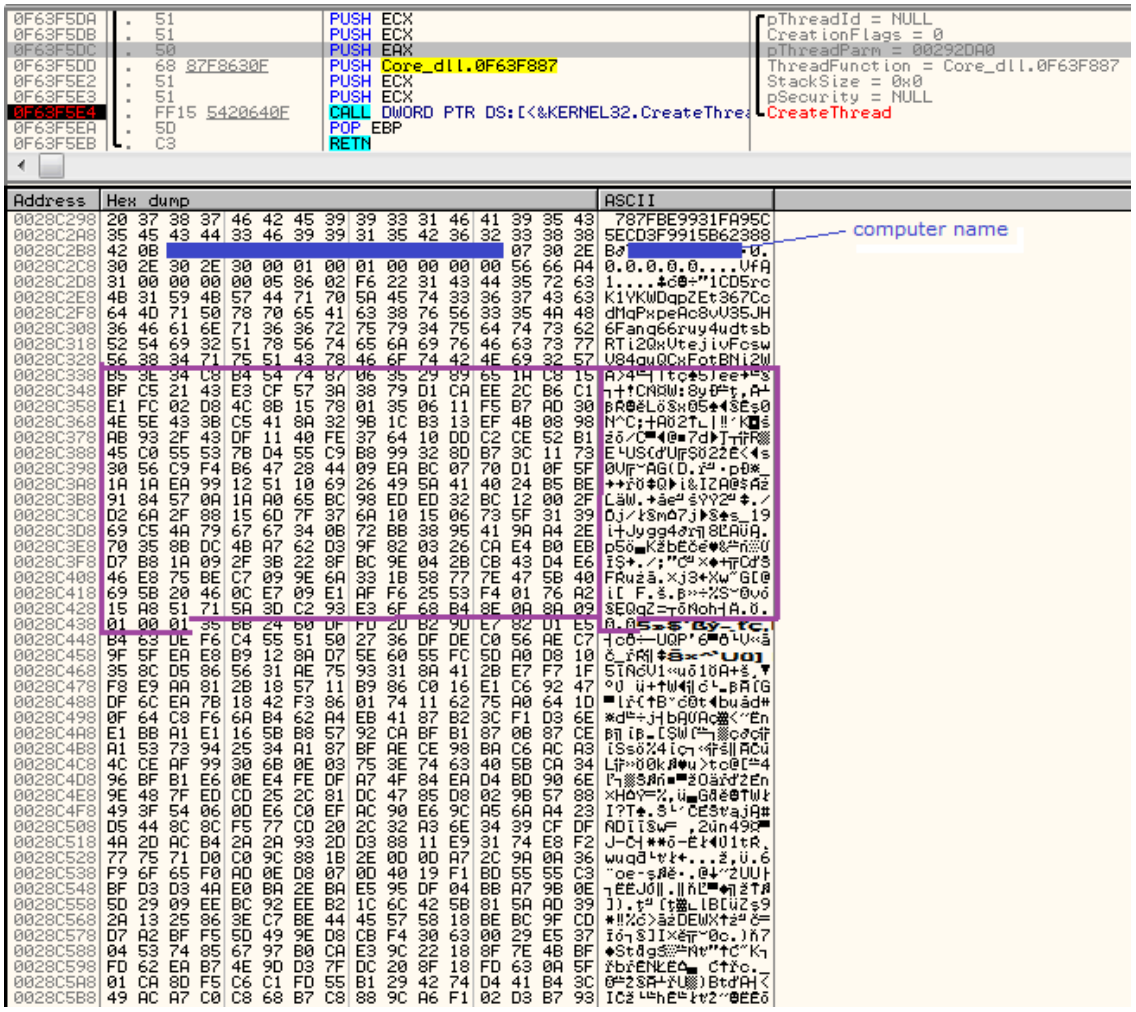
At this stage we can see clearly all the strings and api calls. Also, the full list of extensions of files that are going to be encrypted. (Full list of strings: https://gist.github.com/hasherezade/ceef1c2fed2c70f37d6e)

DllMain sets a mutex automatically generated from the volume serial number (to ensure that the malware is not run more than once), and then starts a new thread that deploys following three procedures:



In the function **start_network_thread** Chimera prepares all the data to be sent to the C&C and after that deploys a new thread, that handles all the network-related operations.

First is the information gathering phase. The victim ID is generated basing on hardware – also, some other information about the local machine is collected: computer name and external IP (by querying address: **bot.whatismyipaddress.com** – if the computer is offline 0.0.0.0 is used as the IP). This data, along with the generated bitcoin wallet address and a generated key pair are supplied as a parameter to the newly created thread.



It is deployed before the process of file encryptions starts – and the public key from this pair is passed forward to the function encrypting files.

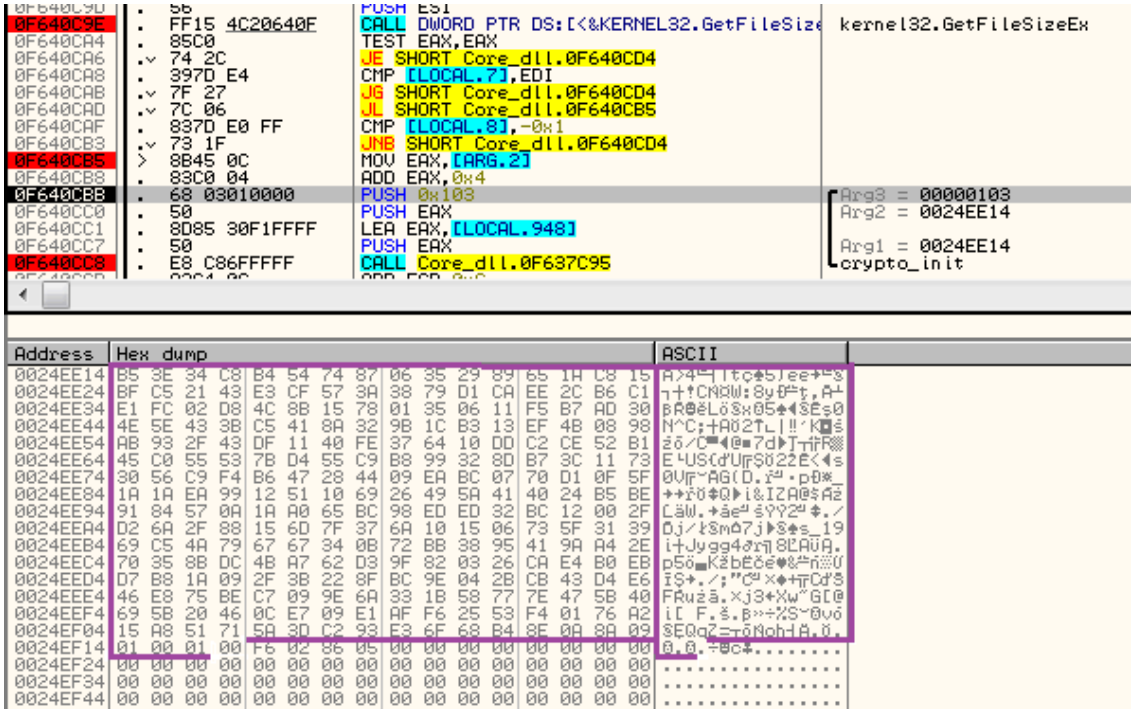
File encryption function (beginning):

```
10010C6A ; int __cdecl encrypt_file(LPCSTR lpFileName, int key)
10010C6A encrypt_file proc near
10010C6A
10010C6A cryptoctx= byte ptr -0ED0h
10010C6A inBuf= byte ptr -928h
10010C6A outBuf= byte ptr -528h
10010C6A NewFileName= byte ptr -128h
10010C6A var_24= dword ptr -24h
10010C6A FileSize= LARGE_INTEGER ptr -20h
10010C6A Overlapped= _OVERLAPPED ptr -18h
10010C6A hHandle= dword ptr -4
10010C6A lpFileName= dword ptr  8
10010C6A key= dword ptr  0Ch
10010C6A
10010C6A push    ebp
10010C6B mov     ebp, esp
10010C6D sub     esp, 0ED0h
10010C73 push    esi
10010C74 push    edi
10010C75 xor     edi, edi
10010C77 push    edi             ; hTemplateFile
10010C78 push    40000080h       ; dwFlagsAndAttributes
10010C7D push    3               ; dwCreationDisposition
10010C7F push    edi             ; lpSecurityAttributes
10010C80 push    edi             ; dwShareMode
10010C81 push    0C0000000h      ; dwDesiredAccess
10010C86 push    [ebp+lpFileName] ; lpFileName
10010C89 mov     [ebp+hHandle], edi
10010C8C call    ds:CreateFileA
```

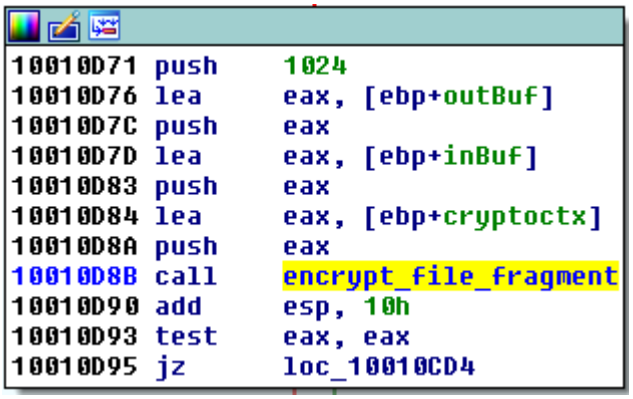
Preparing random symmetric key for each file:

```
10010CB5
10010CB5 make_and_encrypt_symmetric_key:
10010CB5 mov     eax, [ebp+public_key]
10010CB8 add     eax, 4
10010CBB push    259             ; key_len
10010CC0 push    eax             ; buffer
10010CC1 lea     eax, [ebp+file_key]
10010CC7 push    eax             ; a1
10010CC8 call    prepare_file_key
10010CCD add     esp, 0Ch
10010CD0 test    eax, eax
10010CD2 jnz     short loc_10010CE3
```

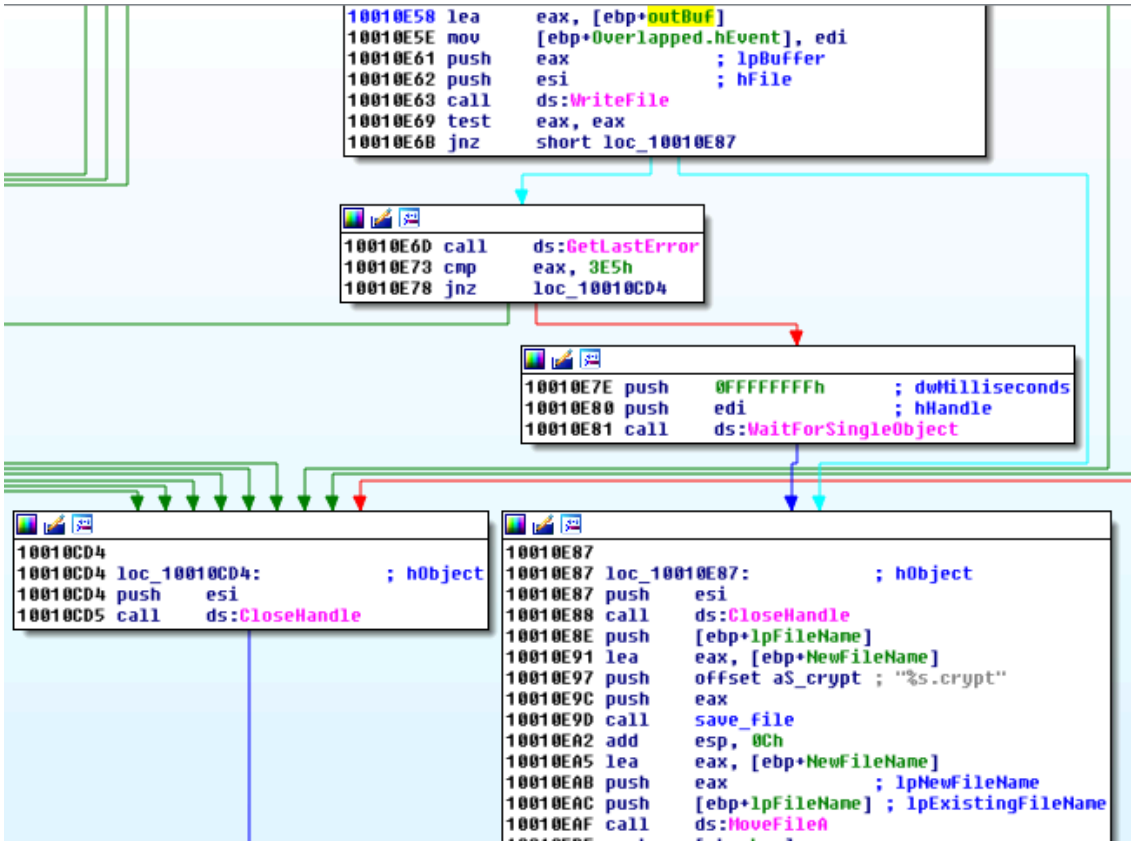
The public key (marked purple) is passed to the function responsible for generating random key for each file. Every symmetric key is encrypted by the public key and then stored in the file:



File is processed chunk by chunk:

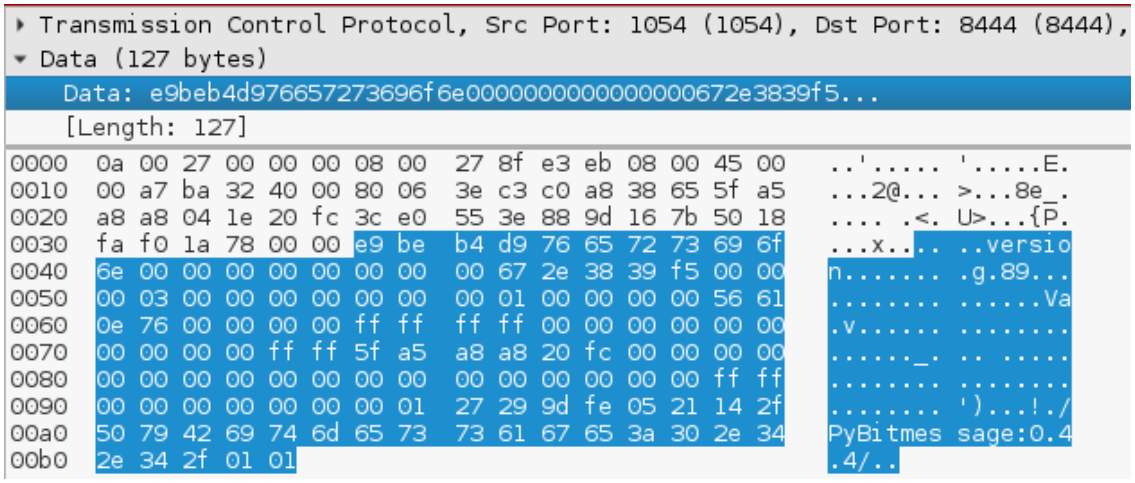


then,saved under the name with suffix *.crypt* added:



Communication

Chimera authors have chosen Bitmessage P2P protocol for the communication with C&C (as well as for the contact with eventual recruits).



To bootstrap the connection the bot uses two hard-coded hosts and receives addresses from them.

```
1000F8D2 lea     eax, [ebp+var_10]
1000F8D5 push    eax             ; int
1000F8D6 push    20FCh           ; hostshort
1000F8DB push    offset name     ; "95.165.168.168"
1000F8E0 call    connect_to_host
1000F8E5 lea     eax, [ebp+var_10]
1000F8E8 push    eax             ; int
1000F8E9 push    1F90h           ; hostshort
1000F8EE push    offset a158_222_211_81 ; "158.222.211.81"
1000F8F3 call    connect_to_host
1000F8F8 lea     eax, [ebp+var_10]
```

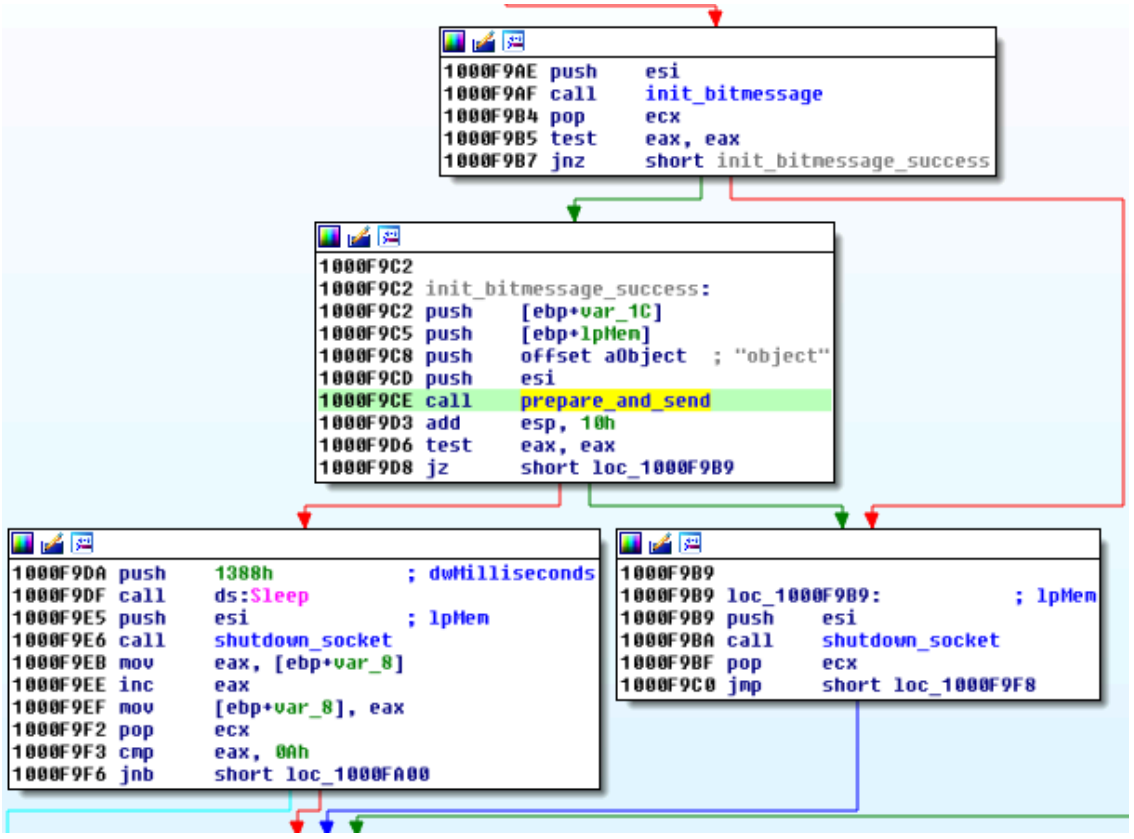
Mind the fact, that those addresses are **not** C&Cs of the malware, but just nodes of Bitmessage. Below – fragment of [original file from Bitmessage protocol]:

```
10  def createDefaultKnownNodes(appdata):
11      ################ Stream 1 ################
12      stream1 = {}
13
14      #stream1[shared.Peer('2604:2000:1380:9f:82e:148b:2746:d0c7', 8080)] = int(time.time())
15      stream1[shared.Peer('5.45.99.75', 8444)] = int(time.time())
16      stream1[shared.Peer('75.167.159.54', 8444)] = int(time.time())
17      stream1[shared.Peer('95.165.168.168', 8444)] = int(time.time())
18      stream1[shared.Peer('85.180.139.241', 8444)] = int(time.time())
19      stream1[shared.Peer('158.222.211.81', 8080)] = int(time.time())
```

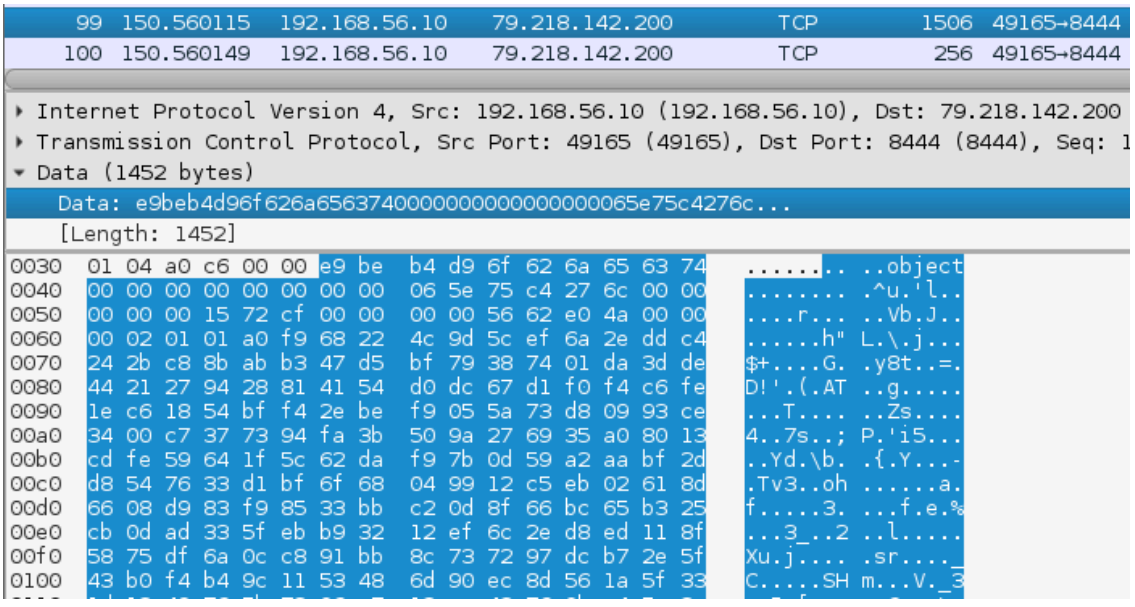
Sample response from one of the servers (**95.165.168.168**):

```
▸ Internet Protocol Version 4, Src: 95.165.168.168 (95.165.168.168), Dst:
▸ Transmission Control Protocol, Src Port: 8444 (8444), Dst Port: 49163 (4
▾ Data (1460 bytes)
    Data: e9beb4d96164647200000000000000000000ab1ecab489f...
    [Length: 1460]
0030  01 00 58 b4 00 00 e9 be  b4 d9 61 64 64 72 00 00   ..X.....  ..addr..
0040  00 00 00 00 00 00 00 00  0a b1 ec ab 48 9f 48 00   ........  ....H.H.
0050  00 00 00 56 61 76 29 00  00 00 01 00 00 00 00 00   ...Vav).  ........
0060  00 00 01 00 00 00 00 00  00 00 00 00 00 ff ff 4d   ........  .......M
0070  ec 41 92 87 d0 00 00 00  00 56 61 7a 2f 00 00 00   .A......  .Vaz/...
0080  01 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00   ........  ........
0090  00 00 00 ff ff 57 13 5e  82 20 fc 00 00 00 00 56   .....W.^  . .....V
00a0  61 71 fe 00 00 00 01 00  00 00 00 00 00 00 01 00   aq......  ........
00b0  00 00 00 00 00 00 00 00  00 ff ff 4d ec 41 92 a1   ........  ...M.A..
00c0  3f 00 00 00 00 56 61 85  2b 00 00 00 01 00 00 00   ?....Va.  +.......
00d0  00 00 00 00 01 00 00 00  00 00 00 00 00 00 00 ff   ........  ........
00e0  ff 4e 34 c5 ef 20 fc 00  00 00 00 56 61 6d 99 00   .N4..  ..  ...Vam..
00f0  00 00 01 00 00 00 00 00  00 00 01 00 00 00 00 00   ........  ........
0100  00 00 00 00 00 ff ff 4d  ec 41 92 af 5e 00 00 00   .......M  .A..^...
0110  00 56 61 7c 06 00 00 00  01 00 00 00 00 00 00 00   .Va|....  ........
0120  01 00 00 00 00 00 00 00  00 00 00 ff ff 4d ec 41   ........  .....M.A
```

Using the received list, it starts a new Bitmessage connection and sends there an ***object***:



Example of sending an ***object*** (containing client data) to a new address: **79.218.142.200**:

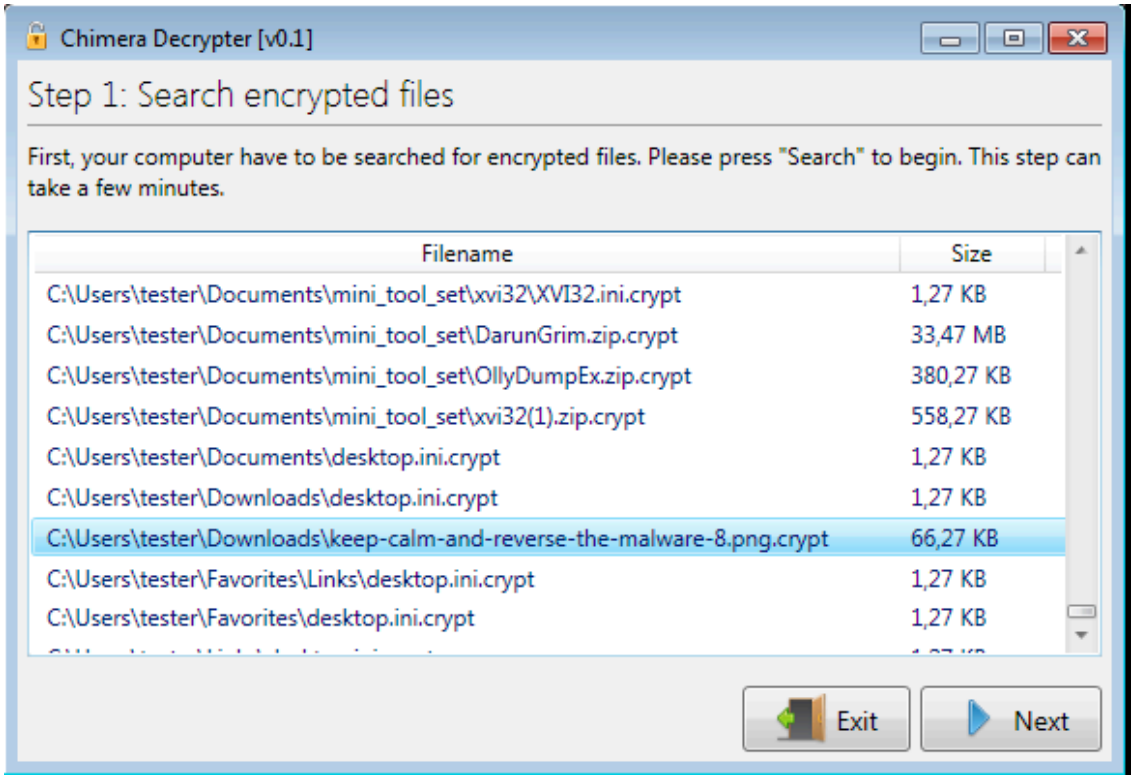


The same protocol is used also to obtain the private key when the ransom is payed. Below – fragment of code of decompiled Decrypter:

```
foreach (InboxBitMessage current in this._bitMessageApiClient.GetAllInboxMessages())
{
    if (current.From.Equals("BM-2cWsXQDYSueEKCtcJS8wzAka3KiYYYC9rB") && current.Subject.Equals("PaymentBroadcast"))
    {
        string[] array = current.Content.Split(new char[]
        {
            '\n'
        });
        string[] array2 = array;
        for (int i = 0; i < array2.Length; i++)
        {
            string text2 = array2[i];
            try
            {
                string[] array3 = text2.Split(new char[]
                {
                    ':'
                });
                if (array3[0].Equals(text))
                {
                    this.MainWindowViewModel.Session.PrivateKey = array3[1];
                    string from = this._bitMessageApiClient.CreateRandomAddress("Address", false, 1, 1);
                    this._bitMessageApiClient.SendMessage("BM-2cWsXQDYSueEKCtcJS8wzAka3KiYYYC9rB", from, "DecryptionMessage", text, 2);
                    this.Running = false;
                    break;
                }
            }
            catch
            {
```

Decrypter

Decrypter is delivered as an .msi installer. It have very friendly user interface and guides a victim through full process of decrypting files.



However, to work properly it requires that the full environment will be set as the malware left it. If we remove ransom notes of try to decrypt files moved from another computer – we will have unpleasant surprise.

Decrypter fetches bitcoin wallet address from the ransom notes – that's why leaving it is necessary to make it work. Also, a hardware ID generated for the current machine must be the same like of the machine on which files have been encrypted. Decryption proceeds only if the payment to a particular address have been received.

Decoder is an executable written in C# and can be easily decompiled. However, it's core functions related to decrypting and hardware id generation are imported from and external dll (that is included in the decoder's package):

```
// Decrypter.ViewModel.Step4ViewModel
[DllImport("PolarSSLWrapper.dll", CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Ansi, SetLastError = true)]
public static extern bool DecryptFileWrapper(string lpPath, IntPtr pInput, int szInputLength);
```

Export table of PolarSSLWrapper.dll:

Offset	Name	Value	Meaning
23E30	Characteristics	0	
23E34	TimeDateStamp	55F95EF7	
23E38	MajorVersion	0	
23E3A	MinorVersion	0	
23E3C	Name	24A6C	Wrapper.dll
23E40	Base	1	
23E44	NumberOfFunctions	2	
23E48	NumberOfNames	2	

Details

Offset	Ordinal	Function RVA	Name RVA	Name
23E58	1	13A20	24A78	DecryptFileWrapper
23E5C	2	13900	24A8B	GetHardwareIdWrapper

Conclusion

Chimera does not have any outstanding obfuscation and once we unpack the core, analysis becomes easy. However, it comes with several ideas that are novel and may slowly become a new trend.

It's communication over P2P protocol is an interesting countermeasure from botnet take down. Also, the idea of blackmailing the user by leaking documents was not found in any malware before. In this case authors ended on bogus threats (sending huge amount of files to the C&C and storing them is much more costly) – but the idea itself is dangerous.

If others cybercriminals will get inspired and decide to implement it, we will have a new headache.

Appendix

http://www.techwalls.com/chimera-ransomware-now-even-harder-decrypt/ – about Chimera's distribution method

http://www.bleepingcomputer.com/news/security/chimera-ransomware-uses-a-peer-to-peer-decryption-service/ – more about Chimera's communication

About the author

Unpacks malware with as much joy as a kid unpacking candies.
