



# Decrypting Chimera ransomware

August 11, 2016 by [Malwarebytes Labs](#)

Last updated: November 27, 2016

We've recently wrote about the [leak of keys for Chimera ransomware](#). In this, more technical post, we will describe how to utilize the leaked keys to decrypt files. Also, we will perform some tests in order to validate the leaked material.

## Components and approach

Usually, writing a ransomware decryptor requires a deep understanding of the used algorithm and finding some flaws in it's implementation. Different vulnerabilities require different mindset in creating the cracking tool. Sometimes we need to re-create the vulnerable algorithm and make a tool for guessing keys (like in the case of cracking Petya). Sometimes, the attacked part is a generator of symmetric keys (like in the case of DMA Locker 2.0) – or the algorithm itself (see the custom encryption of 7ev3n ransomware).

But this time we have almost everything ready-made:

- [set of the keys](#) (leaked)
- decryptor provided originally by the authors of Chimera to it's victims (available [here](#))

The interesting part is only in noticing the relationships between the collected pieces and reorganizing them in the desired way.

## Reversing the original decryptor

As we described previously in the analysis of Chimera, along with the ransom note, authors provided a link from which the victim could download external tool for decryption. To make it work, user needed to purchase the private key, fitting to the public key with which his files were encrypted.

That tool is written in .NET and all the operations of decrypting files are taking place in the external component – a DLL named “PolarisSSLWrapper.dll”, exporting two functions:

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

So, there is no need to re-implement the decrypting function – we can make a use of the existing API.

Let's reverse the main (.NET) component first, to see how this function is called and what are the expected arguments.

We can see the fragment of code in which the external function is loaded and being called:

```

private bool DecryptFile(string path, byte[] privateKey)
{
    bool result;
    try
    {
        IntPtr intPtr = Marshal.AllocHGlobal(privateKey.Length);
        Marshal.Copy(privateKey, 0, intPtr, privateKey.Length);
        bool flag = Step4ViewModel.DecryptFileWrapper(path, intPtr, privateKey.Length);
        Marshal.FreeHGlobal(intPtr);
        if (flag)
        {
            File.Move(path, path.Replace(".crypt", ""));
        }
        result = flag;
    }
    catch
    {
        result = false;
    }
    return result;
}

// Token: 0x06000053 RID: 83
[DllImport("PolarSSLWrapper.dll", CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Ansi, SetLastError = true)]
public static extern bool DecryptFileWrapper(string lpPath, IntPtr pInput, int szInputLength);

```

As we can see above, the function that we are interested in is **DecryptFileWrapper**. It takes 3 parameters: path to the encrypted file (in form of an ASCII string, private key (in form of array of bytes) and the private key length. It returns a boolean value, informing if decrypting the file was successful or not. We can reconstruct it's header as:

```

bool _cdecl DecryptFileWrapper(char filePath, BYTE* privateKey, size_t privateKeySize);

```

The private key is read from the received bitmessage, and decoded from Base64 into the array of bytes:

```

this.OnPropertyChanged("Files");
if (this.DecryptFile(file.Path, Convert.FromBase64String(this.MainWindowViewModel.Session.PrivateKey)))
{
    Application.Current.Dispatcher.Invoke(new Action(delegate
    {
        file.Status = EncryptedFile.FileStatus.Decrypted;
        this.ProgressCurrent++;
    })), new object[0]);
}

```

The infrastructure of Chimera is dead from some months, so we cannot see how the traffic looks in real life at the moment. But thanks to the research published at [Bleeping Computer](#) we can see what exactly was the structure of the message. Example:

```
56209A92A96E9F96B0D9E6F962D0D9EF : 5Zn9azBBDQQznI9znnHZBDs6+nQz6nB9/6DBa0nXbD
z0aghs6fg62Rn9ZzxNDGEzRQ9tFdIZDfa05Lz+n1b6IGnzSDQz0tznrUzGgq9N1bzx0Zus12aH
n6nzZZE6tbQqe/vzbASDuanTBL5SazSARe52Qsq6BEzD5rGqzZhnaLaZr-fbI6bN6A6nnnH51gbe
SAzXdz+6eNxqt9ITziIxF+eDFBBBVZ+zHf6esQzzH2uBnQnaHzzi6tDna9Xngfh6bzQQZBfq+v
FbZ9ZfvnTL6D2arAnBzb6A06Qzfn2zRD5hz5eLzZDIDR00/anb1bU2bvRTH6ZGaXDeBQQ5NHGhQ
EAADZBtx/VaVQsrrdZdasadBHi0RDeZz0Da6g1NRz9/U59zXaaeAN0Di5eb2zQtvr5h6Fvb6tzB
9THtRUGS6Qzt6BxAz/zTg6gs56h5xXzSns1BQRzngandHzFTaBHix692DLxDaziQnZBQDQRB/Zz
5HXQfNzz5aad90+uHasDBDDVzZsngtbgSHDTzA2X5zQs2tHba5z99qf66IQHqZaZD2erzuDQzzx
9N1XaTZEiH2IrVGSZizxEFQzLB16+BDDn5zuG6x6zBhfANB56nDUXt6BnzzuvNVq2xzDTn21Su/
QrHzNbFdGSLazTh29z5Fx9D5Zt6QBBRQ+aFuNDhASgDDH20UDnQB00gBa2t6/i+Lq6eV9ZHzzDd
En2T6HXfg+BEivr5tXB5zZL2zbtvBVxFX2QsBZ9ZrzzG6fIvvnvz6NZ5endiz0IzAQ2Dbqa6gnn
sSnznsVIizznibdZIFvF/nsqbVQZB9Zn2nBQUdTQzzDT65NxHzLRvz6VzZsQV5bih+S+shaD6AS
aDFzHNQD9ZVii+ZrafBxes6zqQaBfEs6z+Igd6nZhEzDLZZN/9QbhQz1zfbf5IFL0nqt2qqnSeq
gz0bzQgZ9Dq6r50SB6xHhn9DfnNa6hR0DuiubDH5z0+n60UD0Uzz6ti6faD5Sz99f6dtQN6LUFZ
n/RagfiqzTXZrvBVz95a9aT9u9tE6qLH0BSNArn0I9rF2eBDQH92zFUBBBVs9e/ZrXZ2Qaz/zn
6BzxQ05qtqNQTZ6AS22z6nBf/200L66zasDHrzZ9agHAX2qBNU1RaUDFszInzzaLD5IzGQeQaAU
60zz6nZQvVe906uDTXG1aZDDfBQUzD/nRDrZUaa2h59Hf5gbeezTHNiHBaBDzzQBx0BX5Zz92Z2
zanfSZZ/2BRnzzzQBzxa/iIxiDNb6Qdd9Bh6/FQnfeznSv5rZ6DZitTGZZUAdzgD5azVAN2G/G+
EssFuhV5aBb0N/N2q+2656zgxBBDzn2+0NIZLudxTXRsDNDza0V/9gzzEaqBdZax6QD1fnQhAVI
n9XZu/D+gr9+ZRqz5266IBST5E5LBZed/s0zS2QHeBrIHnzZtez+02+Q+50g+ZUD6nrhbR2f+N
zB6NgZ6ID9e5hnEEQ99x1nSDZT2aQN6QBRueDRZzNaTz6bvQrGhaBaeF196hZDZLUDIu6rAzB
```

It is: [victim ID]:[base64 encoded key]

After decoding the key we get 1155 (0x483) bytes long array of bytes.

So, that array of raw bytes is the *private key* with which we have to feed the DLL.

## Parsing the keys

The leaked keys are a list of hexadecimal strings. If we convert them to raw, each of them is 0x483 bytes long. It is a good sign, because the format is same as above, and no additional processing is required – only basic parsing from hexadecimal to raw binary.

The file with the leak have a consistent format. Every key ends with a new line. We can just remove the beginning message and use this file as an input:

```
chimera-leak.txt - Notepad
File Edit Format View Help
Like the analysts already detected, Mischa uses parts of the Chimera source. We are NOT connected to the people behind Chimera.
Earlier this year we got access to big parts of their development system, and included parts of Chimera in our project.
Additionally we now release about 3500 decryption keys from Chimera. They are RSA private keys and shown below in HEX format.
It should not be difficult for antivirus companies to build a decrypter with this informations.
!!! Please also check our Raas system, which now has its registration opened: http://janusqgdo2zx75e1.onion/ !!!
ACD367FCF5678E5CE8D5739F2CE0D83B3053B183936525BF73DE47846584719CD1F421910336C5A9C606CB6374B504E413BBE2CC631E380C210CE45B154DA1982:
25AB41E923C137E64B41088B72BB42D0EDC3C3BDE8156B58A871881830EBE8F75B4FCAEF9ECB570A785174AA86285AFCFF1C0E09ECF4E1DF7B898B3AA54F288AB:
A3D2B550D41F5FB1AAF156422F94359412DA4039D51955692E33CB5125C020C2273BC4C64DFBBF644752CBCA16233BFD35700AE614E90876774AD50C66C13468:
D868E5D0C7C1A98F276951B9BE529745ACF47E8130A05A8FE63788692A788DE68811C3AE0EA4F3442A7287DDB11786B05CC4EB1731BFFA2881FDCB76CD6CA8B5:
66D201FF2D88C9C83C964E591AAF00005AF1FBB53D330B9C00F2CCA82E682DA8C5F8327365AFE6AE37FB8E3411AC4AFD0F2FEB43017C6C1F95F0C36C52B3E0FF:
0828813B1F12D4B41FA8A6089FCF6232C8C76C8ED3EBE8E6AFFCA59199D0A7EC54D5C0899AE59513BA0A24E08D5AFDD6B6E8E512CDFCD1829F995E4791963348:
868CEA99ADF4B14ADA29796826BC4704771E004ACDD557460FDD2B9F13F0902CBACA1DA7B0D67A5190A384E202D3A94A5351E9DD4C242AA8F481D4F3567216B1D:
E52601E433379FB0C3B618CD106E653E6652CA10682D1DCD4BB5BD9056A4D3D2986E7EC3F86B6F8897F21C2B08C227B2093F778908BFD7016BB75055213519A:
```

*To be precise: material referred as a set of private keys (in the announcement from the person who leaked them, as well as in the code of original decryptor), in reality is a set of keypairs. Each pair is distributed in a continuous blob that is 0x483 bytes long. First 0x103 bytes contains the public key, and the next 0x380 – the private key. The API of the used DLL expects this full blob to be passed as a ‘private key’ – but thanks to having both of them, it can automatically validate the result of the decryption.*

## Finding the proper key

As we can see, most of the work is already done. The only remaining thing is to search if in the leaked set there is a key that we needed to decrypt our files.

The algorithm that we must use in this case will look similar to a dictionary attack. Our “dictionary” is a set of the leaked keys. As a verification we will try to decrypt one of the encrypted files. General idea is described in the pseudocode below:

```
while ((privateKey = getNextFromSet()) != NULL) {
    if (DecryptFileWrapper(encryptedFile, privateKey, privateKeyLen) == true) {
        printf("Hurray, key found!");
        storeTheFounKey(privateKey);
        return true;
    }
}
printf ("Sorry, your key is not in the leaked set!");
return false;
```

Full implementation you can find [here](#).

After finding the matching key, we can decrypt rest of the files with it’s help, using the same DLL.

## Testing

Test 1:

Chimera generates a unique, random keypair at the beginning of each execution. Then, sends it to the C&C via bitmessage (along with other data collected about the victim).

→*	03550280	56				push esi	
*	03550281	E8 72 13 00 00				call chimera1.35515F8	collect data and make key pair
*	03550286	8B F0				mov esi, eax	
*	03550288	56				push esi	
*	03550289	E8 39 0C 00 00				call chimera1.3550EC7	encrypt available resources
*	0355028E	56				push esi	
*	0355028F	E8 22 00 00 00				call chimera1.35502B6	display ransom note
*	03550294	8B 86 14 02 00 00				mov eax, dword ptr ds:[esi+214]	

For the test purpose I used a key generated by the original Chimera ransomware sample and dumped it from the memory. The key is visible clearly just before being passed to the function which task is to send the prepared material via bitmessage (you can see it's beginning selected on the screenshot below):

\* 0FCC1718 call chimera1.FCC1510  
 \* 0FCC171D push dword ptr ss:[ebp-C]  
 EIP → 0FCC1720 push dword ptr ss:[ebp-10]  
 \* 0FCC1723 call chimera1.FCBF5C2

buffer\_length = 0x523  
 buffer  
 send data via btmessage

dword [ebp-10]=274F70  
 .text:0FCC1720 chimera1.exe:\$11720 #10B20

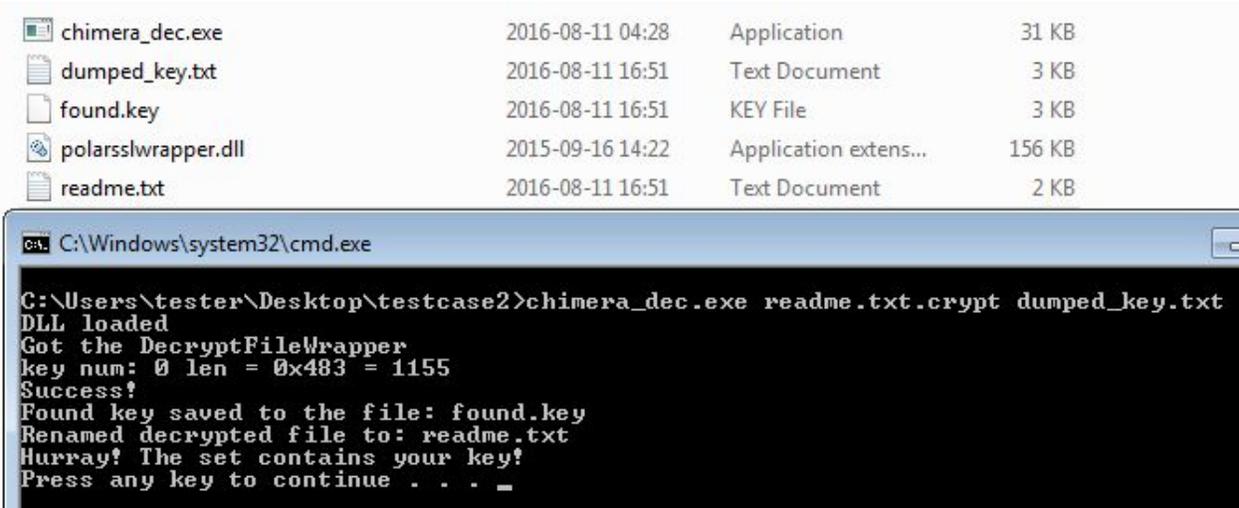
Dump 1    Dump 2    Dump 3    Dump 4    Dump 5    Watch 1

Address	Hex	ASCII
00274F70	20 37 38 37 46 42 45 39 39 33 31 46 41 39 35 43	787FBE9931FA95C
00274F80	35 45 43 44 33 46 39 39 31 35 42 36 32 33 38 38	5ECD3F9915B62388
00274F90	42 08 54 45 53 54 4D 41 43 48 49 4E 45 07 30 2E	B.TESTMACHINE.0.
00274FA0	30 2E 30 2E 30 00 01 00 01 00 00 00 00 57 AC 87	0.0.0.....W-.
00274FB0	AC 00 00 00 00 05 D1 29 F5 22 31 50 72 42 75 35	~.....Ñ)ð"iPrBu5
00274FC0	4C 53 6F 63 6F 75 67 46 77 33 76 43 37 38 4E 35	LSocougFw3vC78N5
00274FD0	41 33 41 71 51 4C 37 42 43 43 47 58 33 35 4B 57	A3AqQL7BCCGX35KW
00274FE0	73 48 44 42 72 56 4C 4D 72 55 65 4E 45 47 6A 4E	sHDBrVLMrUeNEGjN
00274FF0	32 64 37 75 43 52 59 48 6E 58 62 38 34 36 5A 33	2d7uCRYKnXb846Z3
00275000	35 4C 66 52 68 4A 78 79 5A 6D 4E 33 6E 45 58 32	5LFRhJxyZmN3nEX2
00275010	96 61 91 81 C0 50 8A 31 18 AE 50 DA B3 54 DB 45	.a..AP.1.ºPÚT0E
00275020	6A 39 77 FC 53 D5 6F BA BB A8 C5 4F 18 22 42 A2	j9wüS0oº»"Ao."Be
00275030	49 2F 86 2C F7 18 07 B1 CC E0 D1 CF 6D 51 38 7B	I/.,+.±IaImQ8{
00275040	8B 75 1A 26 83 66 06 22 1D 6D 12 03 3E 55 0A D0	.u.&.f.".m..>U.D
00275050	13 2E 5D 6E 55 49 62 C7 AB 77 3C 83 6D 92 6A 32	..]nUIbÇ«w<.m.j2
00275060	73 94 C8 80 F1 E5 15 16 7F 00 E6 C1 99 55 E5 CC	s.E'ñà....æA.UaI
00275070	BD 14 00 0E 00 EE 55 DF 12 BE 4F 36 97 C8 4C 79	%....îUB.%06.ËLy
00275080	A4 5F 8D 88 98 C1 16 6F CB D0 53 25 AD 21 E5 A1	µ...A.oED5%.!âi
00275090	FF 7A 5A 4A 7E 10 AF 18 30 4D 28 DD E8 07 F1 86	yzZJ~...OM+Yè.ñ.
002750A0	BD 75 45 56 79 38 D6 C7 81 BC 78 C6 49 18 31 66	%uEVy8ÖÇ.%{4I.1f
002750B0	04 3F 88 D0 44 3D 13 58 A1 23 6E 69 39 DE BD B1	.?.DD=.Xi#ni9p%±
002750C0	13 55 28 46 A2 EC 25 19 EA C9 DD 02 BE 08 D8 10	.U(Fçì%.éÉY.%0.
002750D0	DC 25 54 5E 22 E2 D3 48 AC 2D 88 27 91 91 5C 60	Ü%T^"áOH~-.'.\
002750E0	71 01 CA F5 9D FA 91 45 6E A4 86 3E 5C 98 33 7F	q.Ëð.ú.Enµ.>\.3.
002750F0	8F 73 F5 F5 11 1E 9D FC DC 6D BE F9 34 EF 78 FA	.sôö...üÜm%ú4ixú
00275100	EA 24 76 76 26 28 AF 39 89 CC 41 68 34 32 00 8D	è\$vv&(9.IAh42..
00275110	01 00 01 0D 9D 25 9C 16 79 F4 18 BF C0 40 27 9C	....%.yð.¿A@'.
00275120	66 89 37 3E 87 97 5A 28 88 BE 05 AC A9 79 8B 50	f.7>..Z+.%-@y»P
00275130	57 01 A8 49 6E D8 F9 C7 20 D0 C9 82 DF 83 7E D0	w.«InòüÇ DÉ.ß*~D
00275140	C9 B2 90 54 AE B4 40 5A 52 E9 29 3B 7A BD 4F 87	É*.Tº@ZRé);z%0.
00275150	DB E1 0F 87 3C D2 4E 83 C0 12 06 3B 7A 9E EA D3	0á..<0N.A...;z.é0
00275160	07 97 3F 9C 01 13 2E 19 A8 00 81 B9 DB AE 08 46	..?.....'0º.F
00275170	C6 64 F8 65 01 A3 83 A1 07 F2 94 30 51 34 40 9F	4døe.f.j.ð.0Q4@.
00275180	FB 0D 19 52 0D 28 75 FE 46 34 AE C8 47 C2 CF 3D	ú..R.(upF4ºÉGAI=
00275190	5E 3D 38 89 DC F2 94 8D C9 E7 2F 55 CB 07 83 8B	^=8.Üð..Éç/UE...
002751A0	7B 86 01 5B 8F 70 7B 14 8B 21 14 3E DE 16 5C 9B	f.Yuxv.1%h.\

Command:  
 Paused    Dump: 00275010 -> 00275492 (0x0000483 bytes)

I converted it to same format as the keys that leaked (continuous hexadecimal string).

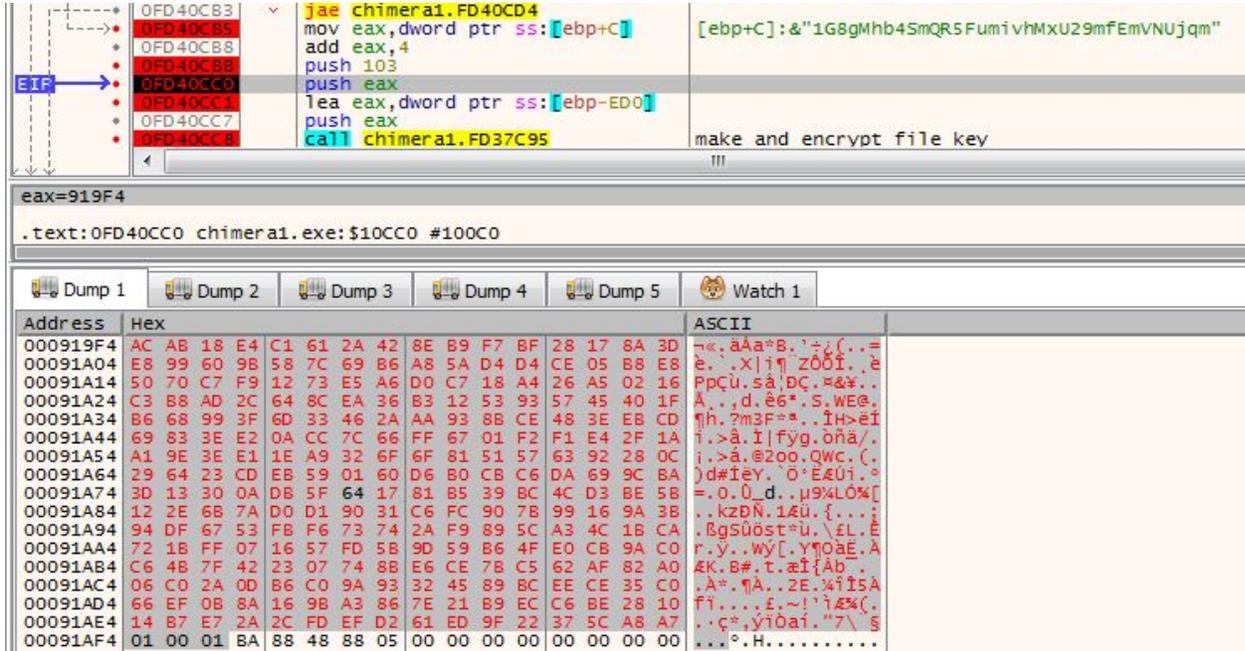
The prepared tool worked – test file has been decrypted successfully:



Test 2:

Due to the fact, that in the leaked material we have complete key pairs, we can use them for the testing purpose. In this experiment I cut out the public key from one of the leaked blobs. Then, I supplied it to the original Chimera sample and deployed. This trick allows to generate encrypted sample set imitating files that could belong to a victim who's key leaked.

Below – substituting the generated public key by the key fetched from the leaked set:



Now, we can attempt to decrypt the attacked file using the set of leaked keys and the prepared application:

chimera_dec.exe	2016-08-11 04:28	Application	31 KB
chimera-leak.txt	2016-08-10 21:12	Text Document	7 819 KB
found.key	2016-08-11 17:50	KEY File	3 KB
idata.txt	2016-08-11 17:50	Text Document	784 KB
polarsslwrapper.dll	2015-09-16 14:22	Application extens...	156 KB

```

C:\Windows\system32\cmd.exe
key num: 416 len = 0x483 = 1155
key num: 417 len = 0x483 = 1155
key num: 418 len = 0x483 = 1155
key num: 419 len = 0x483 = 1155
key num: 420 len = 0x483 = 1155
key num: 421 len = 0x483 = 1155
key num: 422 len = 0x483 = 1155
key num: 423 len = 0x483 = 1155
key num: 424 len = 0x483 = 1155
key num: 425 len = 0x483 = 1155
key num: 426 len = 0x483 = 1155
Success!
Found key saved to the file: found.key
Renamed decrypted file to: idata.txt
Hurray! The set contains your key!
Press any key to continue . . .

```

And it worked! This test have proven, that the supplied material contains real key pairs, compatible with the format used by Chimera (not just a random set of rubbish data).

[Some of the test cases](#), as well as the [compiled application](#), are available in the [github repository](#).

## Conclusion

Looking at the format of keys, we can expect that the leak contains legitimate data. Also, contacting with other researcher ([Fabian Wosar](#)) we got confirmation, that some of the keys from the leak are matching the sample of captured traffic. However, we didn't had an opportunity get evidence from any victim. Chimera is dead from some months, and probably most of the infected people already deleted their encrypted files.

If by any chance you are a victim of Chimera, capable to provide any of such evidence, please contact us.

## Appendix

---

*This was a guest post written by Hasherezade, an independent researcher and programmer with a strong interest in InfoSec. She loves going in details about malware and sharing threat*

information with the community. Check her out on Twitter @[hasherezade](#) and her personal blog: <https://hshrzd.wordpress.com>.