

Maktub Locker – Beautiful And Dangerous

blog.malwarebytes.com/threat-analysis/2016/03/maktub-locker-beautiful-and-dangerous/

hasherezade

March 24, 2016



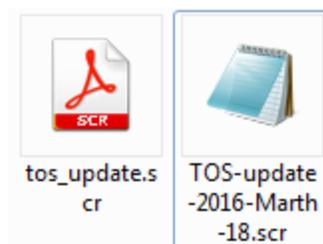
Maktub Locker is another ransomware that comes with a beautifully designed GUI and few interesting features. Its name originates from the Arabic word maktub which means “this is written” or “this is fate”. The authors were probably trying to make a joke by referencing the act of getting infected with ransomware, hinting that it is uninvited and unavoidable, just like fate.

Analyzed samples

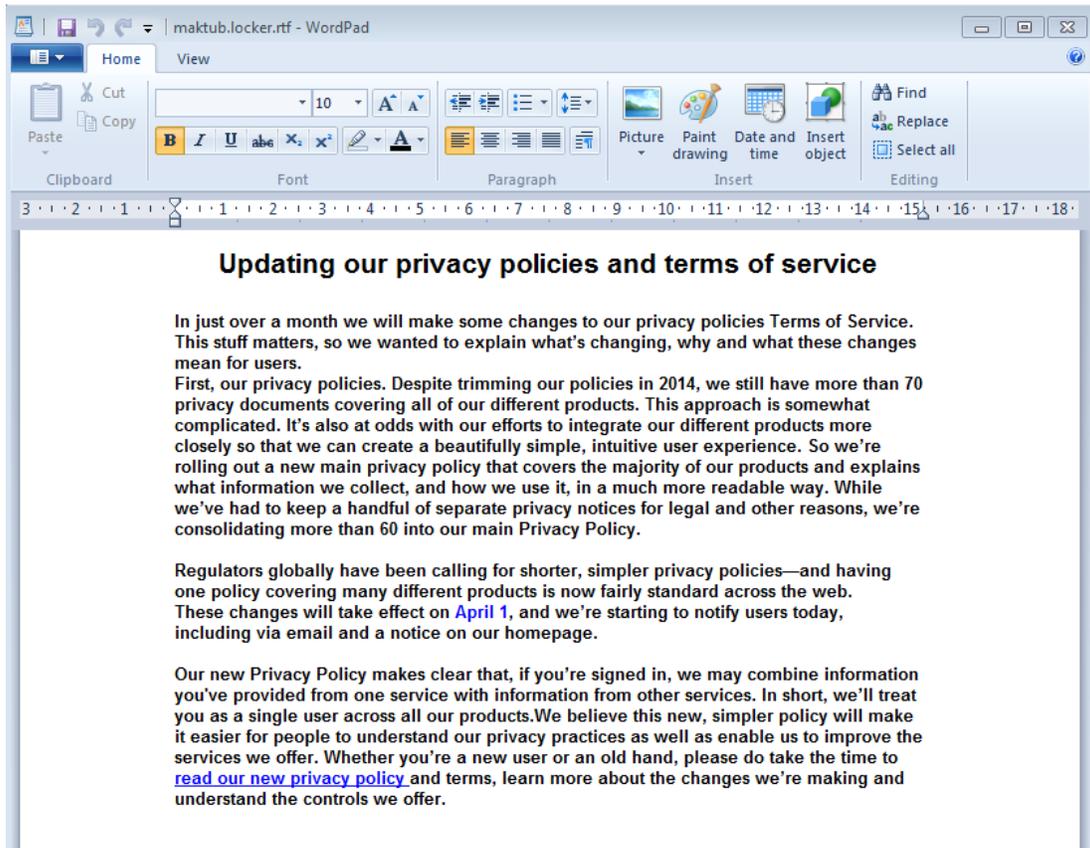
Special thanks to [MalwareHunterTeam](#) and [Yonathan Klijnsma](#) for sharing the samples.

Behavioral analysis

This ransomware comes in a spam campaign, pretending to be a document with a Terms-Of-Service update. This time full packing have a consistent theme: name of the attachment is made to resemble a document (examples: “TOS-update-[...].scr”, “20160321_tos.scr”), also it has a a document-like icon:



An interesting trick used by this ransomware to spoof legitimate behavior is that it really displays a document! Specifically, a fake TOS update in **.rtf** format:



While the user is busy reading the document, the malicious program runs in the background and encrypts his/her files.

Encryption process

Maktub Locker does not need to download a key from the CnC server – data can be encrypted offline as well. Extensions given to the encrypted files are random, generated at runtime – their pattern is: **[a-z]{4,6}**

The new and surprising thing is that encrypted files are much smaller than the original ones. It seems this ransomware not only encrypts but also compresses files.

Original files and their sizes:

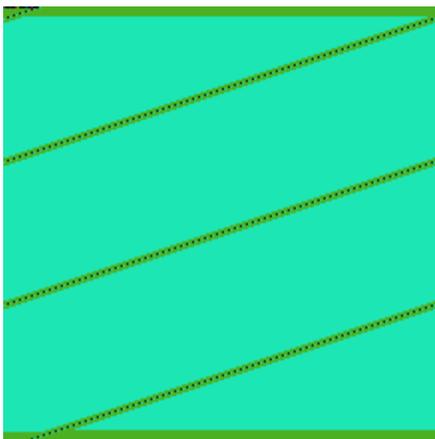
 square.gif	2016-02-22 01:14	14 KB
 square.jpg	2016-02-22 01:14	5 KB
 square.png	2016-01-20 19:19	2 KB
 square1.bmp	2016-01-20 19:21	140 KB
 square2.bmp	2016-02-22 01:15	48 KB
 square3.bmp	2016-02-22 01:15	24 KB
 square4.bmp	2016-02-22 01:15	7 KB
 tekst.txt	2016-01-31 16:50	1 KB

The same files after encryption:

 _DECRYPT_INFO_jkhnhu.html	2016-03-21 18:57	6 KB
 square.gif	2016-02-22 01:14	14 KB
 square.jpg.jkhnhu	2016-03-21 18:57	3 KB
 square.png.jkhnhu	2016-03-21 18:57	1 KB
 square1.bmp.jkhnhu	2016-03-21 18:57	1 KB
 square2.bmp.jkhnhu	2016-03-21 18:57	7 KB
 square3.bmp.jkhnhu	2016-03-21 18:57	1 KB
 square4.bmp.jkhnhu	2016-03-21 18:57	1 KB
 tekst.txt.jkhnhu	2016-03-21 18:57	1 KB

See below a visualization of bytes.

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



^– the bitmap is compressed very well, so the encrypted

file is tiny

A possible reason of compressing files first is to speed up the encryption process.

Encrypted content is different on each run of the sample. However, in a single run, files with the same content will give the same output. We can conclude that the random key is generated only once – at program's start. After that, every file is encrypted using the same key.

After the encryption is finished, the following GUI pops up:



WARNING!

Your personal files are encrypted!

11:54:16

Your documents, photos, databases and other important files have been encrypted with strongest encryption and unique key, generated for this computer. Private decryption key is stored on a secret Internet server and nobody can decrypt your files until you pay and obtain the private key. The server will eliminate the key after a time period specified in this window.

Open <http://qjuyyhqqzfeluxe7.onion.link>
 or <http://qjuyyhqqzfeluxe7.torstorm.org>
 or <http://qjuyyhqqzfeluxe7.tor2web.org>

in your browser. They are public gates to the secret server.

If you have problems with gates, use direct connection:

1) Download TOR Browser from <http://torproject.org>
 2) In the Tor Browser open the <http://qjuyyhqqzfeluxe7.onion>

(Note that this server is available via Tor Browser only. Retry in 1 hour if site is not reachable).

Write in the following public key in the input form on server:

```

0TJ71-B3T5U-C2RQ5-UPK0D-CRBJ7-7EEW4-SAFY0-EQ4FA-XCTUQ-S6MDE-UQF5D-8AJCQ-32X2E-YRXDK
HSFX2-6823X-YH0JR-EYP05-CXGDY-WVJXJ-FMKUS-XYMJJ-48RHF-6QTFQ-HD2TE-RAM1Y-6HENW-KT8PG
MA5W6-1AZHK-7FD5Y-KK5JD-WPQC1-CRVNM-VP846-SUM4A-X58MW-HUJUJ-QRP28-4TJ2R-RKGC0-0SUDQ
4VQEG-1EANX-CT507-HSJGC-UZGGG-22YN4-UUP8D-K7PPM-S8NC1-US74N-U46BW-DRAC5-UJZU0-8FTDA
DUTEB-ZVTPA-W7MAV-3WJ7X-0AF5F-Y7C7E-QYU2U-V82JH-4C2D7-2R6YT-T2671-HCUSB-VECE6-D343J
BTR2F-G35NS-KAE6E-5TFWK-RTVFN-KGUG2-CDN43-FMUJG-W25MY-2DN55-IJMSM-6V1AQ
  
```

Copy Public Key to Clipboard 

It provides a victim a custom-formatted key: 82 chunks, each 5 character long (chunk format: [A-Z0-9]{5}). Each time the sample runs, this key is newly generated.

The same information (and layout) can be found in an HTML file (`_DECRYPT_INFO_[$\$$ EXTENSION].html`), dropped in each encrypted directory.

Website for the victim

These days, it's a common feature of ransomware to provide a TOR-accessed website for the victim and Maktub Locker is no different. Similar to the ransom note, the website is only available in English. In order to access the individual page, the victim is supposed to paste his/her key (the one supplied in the ransom note) into the input box provided on the website.

Enter your decryption key here:

Submit

It then redirects to the main website. In comparison to other ransomware families, Maktub Locker actually has a very nicely designed website, including clean and polite language used.

The screenshot shows the Maktub Locker ransomware website. At the top left is a logo featuring a skull with gears and a wrench. To its right, the text "MAKTUB LOCKER" is displayed. In the top right corner, a yellow timer shows "70:04:01" with the warning "During this time you need to make a payment or the price will be increased." Below the timer, the word "HELLO!" is followed by a gear icon with the number "1". A paragraph of text explains the situation: "We're very sorry that all of your personal files have been encrypted :(But there are good news – they aren't gone, you still have the opportunity to restore them! Statistically, the lifespan of a hard-drive is anywhere from 3 to 5 years. If you don't make copies of important information, you could lose everything! Just imagine! In order to receive the program that will decrypt all of your files, you will need to pay a certain amount. But let's start with something else..." Navigation arrows are visible at the bottom of the page.

It comes with a demo, allowing the decryption of 2 selected files:

The screenshot shows a demo interface with a yellow background and gear icons. The text reads: "Googling 'MAKTUB LOCKER' will instantly bring up many suggestions on deleting the program from your personal computer. But not one of the third party programs will be able to do the most important thing – to decrypt your files! In order to do this, you need to have the private master-key that only we have. And only we can restore all of your files. And to show that we aren't making unfounded statements, we'll prove it. Upload any encrypted file, no larger than 200kb, and we will decrypt it, absolutely free!" Below this, it says "Files available to decrypt: 0" and displays a table with two files for decryption.

Number	File Name	Size	Link
1	square1_bmp.png	631 bytes	Download
2	square2.bmp	7626 bytes	Download

The price of decrypting files starts with 1.4 BTC and increases with time. The distributors warn that the website can be taken down and then it would not be possible to recover encrypted files:

Stage	Time of payment	How much money should be sent
> 1	During the first 3 days	1.4 BTC (~\$588)
2	From 3 to 6 days	1.9 BTC (~\$798)
3	From 6 to 9 days	2.4 BTC (~\$1008)
4	From 9 to 12 days	2.9 BTC (~\$1218)
5	From 12 to 15 days	3.4 BTC (~\$1428)
6	More than 15 days	3.9 BTC (~\$1638)

After 15 days of no payment, we do not guarantee that we saved the key. This site can be disconnected at any moment and you will lose your data forever. Please take this seriously.

Inside

Maktub Locker comes packed in a well-written crypter/FUD, so the code is not readable at first. Also, due to the FUD's functions, detection is problematic and samples have a low detection ratio in the first hours/days after the campaign starts.

Unpacking

Execution starts in the FUD's code. At first we can see many harmless-looking (and completely useless) API calls and random strings.

```

004016E4 . C78424 4C0100 MOV DWORD PTR SS:[ESP+14C],tos_upda.004 ASCII "ntn180 names Uline prompt "
004016F5 . C78424 500100 MOV DWORD PTR SS:[ESP+150],tos_upda.004
00401700 . 85C0 TEST EAX,EAX
00401702 . 7E 0E JLE SHORT tos_upda.00401712
00401704 . 0FB6D1 MOVZX EDX,CL
00401707 . 0FAFD0 IMUL EDX,EAX
0040170A . 03FA ADD EDI,EDX
0040170C . 893D C8AA4100 MOV DWORD PTR DS:[41AA08],EDI
00401712 > 3935 D0AA4100 CMP DWORD PTR DS:[41AA08],ESI
00401718 . C78424 540100 MOV DWORD PTR SS:[ESP+154],tos_upda.004 ASCII "Associates macromolecules "
00401723 . 7E 0B JLE SHORT tos_upda.00401730
00401725 . C78424 530100 MOV DWORD PTR SS:[ESP+158],tos_upda.004 ASCII "Trashing PRINCE configures TuxTween "
00401730 > 41 INC ECX
00401731 . 0FAFCF IMUL ECX,EDI
00401734 . 3BCF CMP ECX,EDI
00401736 . C78424 5C0100 MOV DWORD PTR SS:[ESP+15C],tos_upda.004 ASCII "Intel reasonably Adic damp "
00401741 . 890D C4AA4100 MOV DWORD PTR DS:[41AA04],ECX
00401747 . C78424 600100 MOV DWORD PTR SS:[ESP+160],tos_upda.004 ASCII "scheme Facility XHTML "
00401752 . C78424 640100 MOV DWORD PTR SS:[ESP+164],tos_upda.004 ASCII "Hubbed IXSLProcessor adjust CryptoAPI participant "
0040175D . 7E 08 JLE SHORT tos_upda.00401767
00401763 . 8B4424 3C MOV EAX,DWORD PTR SS:[ESP+3C]
00401767 > C78424 630100 MOV DWORD PTR SS:[ESP+168],tos_upda.004 ASCII "costsVou MSAs LineOne acquire "
00401772 . C78424 6C0100 MOV DWORD PTR SS:[ESP+16C],tos_upda.004 ASCII "awareness APJ monstrously "
0040177D . 3ACB CMP CL,BL
0040177F . 75 0E JNZ SHORT tos_upda.0040178F
00401781 . 8B5424 24 MOV EDX,DWORD PTR SS:[ESP+24]
00401785 . 35D1 CMP EBX,EDX

```

This code is executed first, to deceive tools used to detect malicious behavior. Then it is completely overwritten by new code. However, this is also not the malware code, but just another layer of deception techniques. Below, you can see a fragment of the code responsible for unpacking and executing the bogus TOS update (it is first unpacked from the resources and dropped into the %TEMP% folder as a cabinet file):

```

00401F64 . PUSH ESI
00401F65 . LEA EAX, DWORD PTR SS:[EBP-628]
00401F6B . PUSH EAX
00401F6C . CALL DWORD PTR DS:[402A40]          SETUPAPI.SetupIterateCabinetW
00401F72 . TEST EAX, EAX
00401F74 . JE SHORT tos_upda.00401FB5
00401F76 . PUSH 0A
00401F78 . PUSH ESI
00401F79 . PUSH ESI
00401F7A . PUSH DWORD PTR DS:[402A28]
00401F80 . PUSH ESI
00401F81 . PUSH ESI
00401F82 . CALL DWORD PTR DS:[402A0C]          ShellExecuteW
00401F88 . LEA EAX, DWORD PTR SS:[EBP-20]
00401F8B . PUSH EAX
00401F8C . PUSH ESI
00401F8D . PUSH ESI
00401F8E . CALL DWORD PTR DS:[402A18]          CreateMutexA
00401F94 . CALL DWORD PTR DS:[402A20]          GetLastError
00401F9A . CMP EAX, 0B7
00401F9F . JNZ SHORT tos_upda.00401FBA
00401FA1 . PUSH 8000
00401FA6 . PUSH ESI
00401FA7 . PUSH EBX
00401FA8 . CALL DWORD PTR DS:[402A74]          VirtualFree
00401FAE . PUSH ESI
00401FAF . CALL DWORD PTR DS:[402B48]          ExitProcess

```

The real malicious code starts in another module that is unpacked into dynamically allocated memory.

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
0000044C	10001230	7FFD9000	ERROR_SUCCESS (00)	Active	32 + 0	20.9200 s	6.4592 s
000007FC	01C357A4	7FFDD000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0100 s
000009DC	00406D13	7FFDF000	ERROR_RESOURCE_TY	Active	32 + 0	2.2732 s	0.1201 s
00000AE4	773AFD0F	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000B18	10001230	7FFD8000	ERROR_SUCCESS (00)	Active	32 + 0	21.4107 s	6.0286 s
00000C94	773B03E7	7FFDB000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s

You can see above 2 threads with entry: 0x10001230. They belong to this malicious module. If we try to dump this memory area, we obtain a new PE file:

```

D Dump - 10000000..10021FFF
10000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZE...+...
10000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 $......@.....
10000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
10000030 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00 .....
10000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 A\|A.|.?!$QL=Th
10000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program cannot
10000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
10000070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode...$.
10000080 CD 48 B2 50 89 29 DC 03 89 29 DC 03 89 29 DC 03 =HPPe)w)w)
10000090 80 51 4F 03 98 29 DC 03 89 29 DD 03 C5 29 DC 03 C00s)w)T+)
100000A0 3C B7 39 03 8E 29 DC 03 3C B7 03 03 88 29 DC 03 <E9wA)w)w)
100000B0 3C B7 00 03 88 29 DC 03 84 7B 07 03 88 29 DC 03 <E.w)w)w)
100000C0 3C B7 02 03 88 29 DC 03 52 69 63 68 89 29 DC 03 <E.w)w)w)Rich)
100000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100000E0 50 45 00 00 4C 01 03 00 68 CD EB 56 00 00 00 PE..L0.w.h=0U...
100000F0 00 00 00 00 E0 00 02 21 08 01 0C 00 00 C0 00 00 .0.w)w)w)w)
10000100 00 10 00 00 00 40 01 00 B0 06 02 00 00 50 01 00 .w)w)w)w)w)w)
10000110 00 10 02 00 00 00 00 10 00 10 00 00 00 02 00 00 .w)w)w)w)w)w)
10000120 05 00 01 00 00 00 00 05 00 01 00 00 00 00 00 .w)w)w)w)w)w)
10000130 00 20 02 00 00 10 00 00 00 00 00 02 00 40 01 .w)w)w)w)w)w)

```

This PE file is loaded in a continuous area of dynamically allocated memory and used as a new virtual section.

Unfortunately this time, dumping it will not give us the independent payload – unpacked content has invalid headers, i.e:

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.
UPX0	400	0	1000	10001000	60000080	0	0
>	400	^	10002000	^	r-x		
UPX1	400	BA00	15000	10015000	60000040	0	0
>	BE00	^	1002A000	^	r-x		
.rsrc	BE00	1000	21000	10021000	C0000040	0	0
>	CE00	^	10042000	^	rw-		

The image shows a memory dump comparison between 'Raw' and 'Virtual' addresses. In the 'Raw' view, the UPX0 section is at address 400, UPX1 is at 400 with size BA00, and .rsrc is at BE00 with size 1000. In the 'Virtual' view, UPX0 is at 1000, UPX1 is at 15000, and .rsrc is at 21000. Red lines indicate the mapping between raw and virtual addresses.

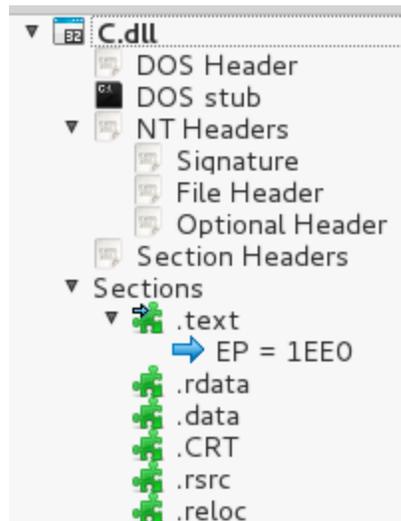
This trick is used by the crypter in order to protect the payload from automated dumping tools. However, if we capture the unpacking at the right moment, before the headers are overwritten, we still can recover the original payload. It turns out to be a DLL (packed with UPX):

Offset	Name	Value	Meaning
CD68	Characterist...	0	
CD6C	TimeDateSt...	56EBCD67	
CD70	MajorVersion	0	
CD72	MinorVersion	0	
CD74	Name	21FA4	C.dll
CD78	Base	1	
CD7C	NumberOff...	2	
CD80	NumberOfN...	2	
CD84	AddressOff...	21F90	

Details					
Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
CD90	1	2890	21FAA	one	
CD94	2	27B0	21FAE	two	

The code responsible for encrypting files is located in the function “one”.

The DLL is packed with genuine version of UPX, so we can easily unpack it, getting an deobfuscated DLL as result with the following sections layout (unpacked **C.dll** : 38eff2f7c6c8810a055ca14628a378e7):



However, we will still not see valid strings. Imports also seems irrelevant to the functionality (we will not find there, for example, any reference to the windows Crypto API). It is due to the fact that real imports are resolved dynamically. At the beginning of execution, the function “one” loads them on it’s own – first, decrypting their names:

```
1000D75C mov     edx, offset aPKUisMbiJgod ; "Ć:*Ů+óĚ[-číĚ--üžú<<Ŧ"
1000D761 lea     ecx, [ebp+lpMem]
1000D764 call    decrypt_name
1000D769 push   dword ptr [eax] ; lpProcName
1000D76B push   dword ptr [edi] ; hModule
1000D76D call    ebx ; GetProcAddress
1000D76F mov     esi, eax
1000D771 lea     ecx, [ebp+lpMem]
1000D774 mov     [edi+38h], esi ; store the handle
```

Then, they are accessed via dynamically loaded handles.

Execution flow

This malware first makes a list of all the files, and then processes them one by one. It also unpacks a built-in configuration with list of restricted paths and attacked executables. Each processed path is first checked against this list.

Below you can see a fragment of code opening file that is chosen to be encrypted. Call to the function CreateFileA is performed via handle and dynamically loaded into the EAX register:

```

10004FB8 85C0 TEST EAX,EAX
10004FBA 0F84 70030000 JE 10005330
10004FC0 8B47 54 MOV EAX,DWORD PTR DS:[EDI+54]
10004FC3 6A 00 PUSH 0
10004FC5 68 00000000 PUSH 00000000
10004FCA 6A 03 PUSH 3
10004FCC 6A 00 PUSH 0
10004FCE 6A 01 PUSH 1
10004FD0 68 000000C0 PUSH C0000000
10004FD5 FF75 08 PUSH DWORD PTR SS:[EBP+8]
10004FD8 FFD0 CALL EAX kernel32.CreateFileA
10004FDA 8BD8 MOV EBX,EAX
10004FDB 83F8 FF CMP EBX,-1

```

Then, a new file is created – with an extension added:

```

100050FF 8B75 F0 MOV ESI,DWORD PTR SS:[EBP-10]
10005102 8B47 54 MOV EAX,DWORD PTR DS:[EDI+54]
10005105 6A 00 PUSH 0
10005107 6A 00 PUSH 0
10005109 6A 02 PUSH 2
1000510B 6A 00 PUSH 0
1000510D 6A 00 PUSH 0
10005110 6A 00 PUSH 0
10005112 68 00000040 PUSH 40000000
10005114 56 PUSH ESI
10005115 FFD0 CALL EAX kernel32.CreateFileA
10005117 8945 F0 MOV DWORD PTR SS:[EBP-10],EAX
1000511A 83F8 FF CMP EAX,-1
1000511D 75 0D JNZ SHORT 1000512C
1000511F 8B47 54 MOV EAX,DWORD PTR DS:[EDI+54]
EAX=7581CEE8 (kernel32.CreateFileA)
0280FED4 01E2451C LE00 ASCII "C:\Python27\tcl\tix8.4.3\demos\samples\SGrid0.tcl.fsqscp"
0280FED8 40000000 ...@
0280FEDC 00000000 ...
0280FEF0 00000000 ...

```

At first both files coexist in the system – the newly created file has 0 size. After it is filled by the encrypted content, the original file gets deleted.

	SGrid0.tcl.fsqscp C:\Python27\tcl\tix8.4.3\demos\samples	Type: FSQSCP File	Date modified: 2016-03-22 17:39 Size: 0 bytes
	SGrid0.tcl C:\Python27\tcl\tix8.4.3\demos\samples	Type: TCL File	Date modified: 2008-09-27 12:56 Size: 3,48 KB

After the process of encryption finished, the malware creates and pops up the dialog box.

Below – code responsible for popping up the GUI with a ransom note:

```

10012773 FFD3 CALL EBX
10012775 8D45 FC LEA EAX,DWORD PTR SS:[EBP-4]
10012776 C745 FC 00000000 MOV DWORD PTR SS:[EBP-4],0
1001277D 50 PUSH EAX
1001277E 6A 01 PUSH 1
10012780 57 PUSH EDI
10012781 FF15 30310110 CALL DWORD PTR DS:[10013138] ole32.CreateStreamOnHGlobal
10012787 85C0 TEST EAX,EAX
10012789 78 15 JS SHORT 100127A0
1001278B 8B4D FC MOV ECX,DWORD PTR SS:[EBP-4]
1001278E E8 D0E9FEFF CALL 10001170
10012793 8B4D FC MOV ECX,DWORD PTR SS:[EBP-4]
10012796 8BF0 MOV ESI,EAX
10012798 51 PUSH ECX
10012799 8B11 MOV EDX,DWORD PTR DS:[ECX]
1001279B FF52 08 CALL DWORD PTR DS:[EDX+8]
1001279E EB 02 JMP SHORT 100127A2
100127A0 33F6 XOR ESI,ESI
100127A2 6A 00 PUSH 0
100127A4 68 A0320010 PUSH 100032A0
100127A9 6A 00 PUSH 0
100127AB 68 81000000 PUSH 81
100127B0 FF35 88860110 PUSH DWORD PTR DS:[10018688]
100127B6 8935 BC860110 MOV DWORD PTR DS:[100186BC],ESI USER32.DialogBoxParamA
100127BC FF15 A4300110 CALL DWORD PTR DS:[100130A4]
100127C2 FF75 F8 PUSH DWORD PTR SS:[EBP-8]
100127C5 FF15 30310110 CALL DWORD PTR DS:[10013130] gdiplus.GdiplusShutdown
100127CB 5F POP EDI
100127CD 5F POP ESI

```

What is attacked?

It is common practice to exclude some chosen countries from the attack. In this case, before deploying the malicious actions, the application fetches the keyboard locale list. If it finds Russian (value 0x419 = 1049) among them, the malware exits without infecting files:

```
0F90E076 . MOV ESI,EAX
0F90E078 . PUSH ESI
0F90E079 . PUSH [LOCAL_1]
0F90E07C . CALL DWORD PTR DS:[&USER32.GetKeyboardL
0F90E082 . MOV ECX,EAX
0F90E084 . XOR EAX,EAX
0F90E086 . TEST ECX,ECX
0F90E088 . JLE SHORT one.0F90E0A2
0F90E08A . MOV EDX,0x419
0F90E08F . NOP
0F90E090 . CMP WORD PTR DS:[ESI+EAX*4],DX
0F90E094 . JE SHORT one.0F90E09D
0F90E096 . INC EAX
0F90E097 . CMP EAX,ECX
0F90E099 . JLE SHORT one.0F90E090
0F90E09B . JMP SHORT one.0F90E0A2
0F90E09D . MOV EDI,0x1
```

pLocaleId = 0030E5D8
nLocaleId = 0x3
GetKeyboardLayoutList

locale_id = 1049 -> Russia

Excluded from the attack are also some predefined folders:

```
"\\internet explorer\\;\\history\\;\\mozilla\\;\\chrome\\;\\temp\\;\\program files\\;\\program files (x86)\\;\\microsoft\\;\\cache\\;\\chaches\\;\\appdata\\;"
```

The built-in configuration also specifies what are the extensions to attack:

```
03AB2F4 70 61 68 00 7F 55 EE 70 pak.0Utp
03AB2FC 68 6D 00 88 03 00 00 00 hm.k*...
03AB304 03 00 00 00 01 00 00 00 *...0...
03AB30C 70 64 64 00 42 55 EE 70 pdd.BUtp
03AB314 75 70 00 88 03 00 00 00 up.k*...
03AB31C 03 00 00 00 01 00 00 00 *...0...
03AB324 70 64 66 00 45 55 EE 70 pdf.EUtp
03AB32C 2C 76 00 88 03 00 00 00 ,v.k*...
03AB334 03 00 00 00 01 00 00 00 *...0...
03AB33C 70 65 66 00 48 55 EE 70 pdf.HUtp
03AB344 65 74 00 88 03 00 00 00 et.k*...
03AB34C 03 00 00 00 01 00 00 00 *...0...
03AB354 70 65 6D 00 48 55 EE 70 pem.KUtp
03AB35C 70 6C 00 88 03 00 00 00 pl.k*...
03AB364 03 00 00 00 01 00 00 00 *...0...
03AB36C 70 66 78 00 4E 55 EE 70 pfx.NUtp
03AB374 78 6C 00 88 03 00 00 00 xl.k*...
03AB37C 03 00 00 00 01 00 00 00 *...0...
03AB384 70 67 70 00 51 55 EE 70 pgp.0Utp
03AB38C 74 6D 00 88 03 00 00 00 tm.k*...
03AB394 03 00 00 00 01 00 00 00 *...0...
03AB39C 70 6E 67 00 54 55 EE 70 png.TUtp
03AB3A4 22 58 00 88 03 00 00 00 "%k*...
03AB3AC 03 00 00 00 01 00 00 00 *...0...
03AB3B4 70 70 74 00 57 55 EE 70 ppt.WUtp
03AB3BC 00 8C 00 88 03 00 00 00 .l.k*...
03AB3C4 03 00 00 00 01 00 00 00 *...0...
03AB3CC 70 73 64 00 5A 55 EE 70 psd.ZUtp
03AB3D4 00 8C 00 88 03 00 00 00 .l.k*...
03AB3DC 03 00 00 00 01 00 00 00 *...0...
03AB3E4 70 73 68 00 5D 55 EE 70 psk.JUtp
03AB3EC 58 58 00 88 03 00 00 00 [X.k*...
03AB3F4 03 00 00 00 01 00 00 00 *...0...
03AB3FC 70 73 74 00 40 55 EE 70 pst.sUtp
03AB404 00 8C 00 88 03 00 00 00 .l.k*...
03AB40C 03 00 00 00 01 00 00 00 *...0...
03AB414 70 74 78 00 A3 55 EE 70 ptx.uUtp
```

Like other ransomware families, it attacks not only the local disk but also network shares and disks mounted by virtual environments, including external hard drives.

How does the encryption work?

Maktub Locker uses Window Crypto API. But, as we concluded from the analysis, it uses only one key for all files (does not generate a random key per file). Let's see what technique it uses to obtain keys...

In this run, the key supplied to a user was:

X25HE - J53ZU - QERDZ - ZNUJ3 - SERJ6 - J617E - UUSAZ - AFG2G - 83B08 - 2SHC1 - AUYPZ - GJHF2 - W7321 - 144TM
 VKFKR - 6TKRV - STG4B - CE5M2 - TAH4W - MP541 - GD3SB - HE43J - ZF4TK - ZNZTG - R7ZBZ - AKM2U - T6TYN - 53J7H
 MU6J6 - BTSJC - FVQQR - EH755 - C1WCJ - 7SNPT - MHFBS - Q638V - MASEB - R16HW - P84P2 - 7EEX8 - KXAHB - D10F7
 GF071 - U37K3 - GJ5Q5 - WD0PD - 2EG16 - KMC5R - RPCBX - R8EV3 - ZPXQV - TDVXM - SEEFX - XK23J - FCH4Z - RNBPN
 XE6X5 - 4W8CT - WJQJU - 071T5 - DSUZW - JGSZA - KFKZ6 - 4DU0S - 80H1H - CEP2J - PDSKA - UXBR8 - 8C1BB - SDQNC
 1C8F7 - HPZ2G - Q5JVN - F6WXH - PMUSR - 8G4HT - RNYVM - DZNQ3 - Y8KZJ - NYC1G - SPR3T - U5GD5

Let's investigate what is the relationship between this key and the key used to encrypt files. So far we know that it must be generated locally.

First it initialized two crypto contexts - both with the same settings, using provider type: PROV_DH_SCHANNEL

```

0F80DFC1 | . | CALL one.0F812F20
0F80DFC6 | . | PUSH 0xF000040
0F80DFCB | . | PUSH 0x18
0F80DFCD | . | LEA ESI,DWORD PTR DS:[EDI+0x100]
0F80DFD3 | . | PUSH DWORD PTR DS:[EAX]
0F80DFD5 | . | MOV EAX,DWORD PTR DS:[EDI+0x4]
0F80DFD8 | . | PUSH 0x0
0F80DFDA | . | PUSH ESI
0F80DFDB | . | CALL EAX
  
```

flags = CRYPT_VERIFYCONTEXT | CRYPT_CREATE_SALT
 dwProvType = PROV_DH_SCHANNEL
 pszProvider = "Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)"
 advapi32.CryptAcquireContextA
 pszContainer = NULL
 *phProv
 advapi32.CryptAcquireContextA

EAX=775191DD (advapi32.CryptAcquireContextA)

Gets 32 random bytes, using function CryptGenRandom

```

0F8029A3 | . | PREFIX REP:
0F8029A4 | . | MOVQ QWORD PTR DS:[EAX+0x10],MM0
0F8029A8 | . | MOV EAX,DWORD PTR DS:[ECX+0x28]
0F8029AB | . | PUSH DWORD PTR DS:[ECX+0x100]
0F8029B1 | . | CALL EAX
0F8029B3 | . | CALL one.0F80E740
  
```

Superfluous prefix
 advapi32.CryptGenRandom
 advapi32.CryptGenRandom

Address	Hex dump	ASCII
00305A60	00 00 00 00 00 00 00 00
00305A68	00 00 00 00 00 00 00 00
00305A70	00 00 00 00 00 00 00 00
00305A78	00 00 00 00 00 00 00 00

Creates MD5 sum of this random data (using: CryptCreateHash, CryptHashData)

```

0F957DF6 | . | MOV [LOCAL.2],0x0
0F957DFD | . | PUSH 0x0
0F957DFF | . | PUSH 0x8003
0F957E04 | . | PUSH DWORD PTR DS:[ESI+0x100]
0F957E0A | . | MOV EAX,DWORD PTR DS:[ESI+0x8]
0F957E0C | . | CALL EAX
0F957E0F | . | TEST EAX,EAX
0F957E11 | . | JE SHORT one.0F957E2E
0F957E13 | . | MOV EAX,DWORD PTR DS:[ESI+0xC]
0F957E16 | . | PUSH 0x0
0F957E18 | . | PUSH 0x20
0F957E1A | . | PUSH [ARG.1]
0F957E1D | . | PUSH [LOCAL.1]
0F957E20 | . | CALL EAX
0F957E22 | . | TEST EAX,EAX
0F957E24 | . | INZ SHORT one.0F957E27
  
```

MD5_SUM
 advapi32.CryptCreateHash
 advapi32.CryptHashData
 advapi32.CryptHashData
 advapi32.CryptHashData
 dataLen = 32
 pbData
 hash
 advapi32.CryptHashData
 advapi32.CryptHashData

EAX=7751DF36 (advapi32.CryptHashData)

Address	Hex dump	ASCII
003A5A60	12 C4 2B 35 EF D1 37 FF 8B 54 D2 21 19 FD B3 2F	⚡+5*07 0T0+4x /
003A5A70	3E 97 78 C4 3C BC D2 34 AE 7B 77 F9 09 C6 4B F5	>Sw-<04«Cw".AKS

Then, using function `CryptDeriveKey` it converts the MD5 hash into a 256 bit AES key (AlgID = 0x6610 -> `CALG_AES_256`).

```

0F957E37 |> | LEA EAX, [LOCAL.2]
0F957E3A | . | PUSH EAX
0F957E3B | . | MOV EAX, DWORD PTR DS:[ESI+0x24]
0F957E3E | . | PUSH 0x0
0F957E40 | . | PUSH [LOCAL.1]
0F957E43 | . | PUSH 0x6610
0F957E48 | . | PUSH DWORD PTR DS:[ESI+0x100]
0F957E4E | . | CALL EAX

```

*phKey
 advapi32.CryptDeriveKey
 flags
 hBaseData
 AlgID
 hProv
 advapi32.CryptDeriveKey

It also imports RSA public key (2048 bit). This key is hardcoded in the binary.

```

0F95E774 | . | MOV EAX, DWORD PTR DS:[EBX+0x20]
0F95E775 | . | CALL EAX
0F95E777 | . | LEA EAX, [LOCAL.1]
0F95E77A | . | MOV [LOCAL.1], 0x0
0F95E781 | . | PUSH EAX
0F95E782 | . | MOV EAX, DWORD PTR DS:[EBX+0x1C]
0F95E785 | . | PUSH 0x0
0F95E787 | . | PUSH 0x0
0F95E789 | . | PUSH 0x114
0F95E78E | . | PUSH one.0F968438
0F95E793 | . | PUSH DWORD PTR DS:[EBX+0x100]
0F95E799 | . | CALL EAX

```

advapi32.CryptDeriveKey
 advapi32.CryptImportKey
 advapi32.CryptImportKey
 advapi32.CryptImportKey
 advapi32.CryptImportKey
 advapi32.CryptImportKey

0F968438=one.0F968438

Address	Hex dump	ASCII
0F968438	06 02 00 00 00 A4 00 00 52 53 41 31 00 08 00 00	*0...A..RSA1.0..
0F968448	01 00 01 00 55 70 84 A9 B8 6F ED 2E 51 35 5B B9	0.0.UpaeSoY.05[
0F968458	AC C6 31 A3 C8 DF 75 61 53 42 1B D0 77 3F F8 AE	CA1U...uaSB+dw?o<<
0F968468	CB C4 75 87 AE 9F 0A 92 97 AF EF 8D 93 5E C1 5A	ir-uc<<C. [3>^20^+Z
0F968478	9B F5 36 69 A2 B0 1F E8 00 15 0F D6 25 58 9F 83	T86i0...R.Swi%L0a
0F968488	FB EC 96 37 D7 41 BC A1 67 34 0B CD 71 84 E2 AC	UjP7iAP ig40=qã0C
0F968498	BD C0 03 00 8F 0E 56 6C 50 FC 4F 79 AA A8 77 0A	2^E.C8U1PR0y Ew.
0F9684A8	6C CE F0 E9 96 9C 9F E0 3C 47 D1 3E AF 24 17 B0	lf-0Pvc0<G0>>\$#
0F9684B8	4F 30 35 11 47 D3 DA 74 4B 25 17 92 16 7D 9B C9	0054GErtK2* (.)Tf
0F9684C8	AD 00 C5 DC 8A 3E 9B 8E E4 DF C1 83 F6 1B D9 A6	s.+>0>TAn...a+~2
0F9684D8	CD D0 8D FF 13 ED B7 56 32 65 32 E2 67 D2 1D AE	=ã2 !!YEU2e20g0+!!
0F9684E8	E2 8E 5A 6A E3 AD 62 D6 4E 70 4F F3 9C 6D 7F 13	0AZjN0giNp0^vm0!!
0F9684F8	49 7D 39 4B 55 7F 28 EC 2C 16 88 89 B8 B2 86 CE	I>9KU0(y. - k0000f
0F968508	89 61 76 7D 58 8D 11 55 FE 33 DA 8F 5F 24 6F 52	ëav)[24U#3rC_sor
0F968518	93 01 6F D4 40 9A 11 94 58 7F 74 7E 48 8A 86 41	00dMü40X0t^H00A
0F968528	2F B8 C3 5B EF 4F 41 E6 F4 51 24 BB 45 61 59 E3	^ [-0A\$~0\$EaYn
0F968538	B4 F3 F3 9C A8 8D A6 B9 4C 1B 3D F2 0E 7D 84 7D	+^vE22jL+.(#ã)
0F968548	59 E7 34 AD 00 00 00 00 01 00 00 00 EB 03 00 00	Y\$4s....0...0#..

The random 32 bytes (base of the AES key), along with the random extension, are concatenated together. Then, the prepared buffer is RSA encrypted:

```

0F95E872 | . | ADD ESP, 0x4
0F95E875 | . | LEA EAX, [LOCAL.3]
0F95E878 | . | PUSH [LOCAL.2]
0F95E87B | . | PUSH EAX
0F95E87C | . | MOV EAX, DWORD PTR DS:[EBX+0x14]
0F95E87F | . | PUSH EDI
0F95E880 | . | PUSH 0x0
0F95E882 | . | PUSH 0x1
0F95E884 | . | PUSH 0x0
0F95E886 | . | PUSH [LOCAL.1]
0F95E889 | . | CALL EAX

```

dataLen = 0x2c
 advapi32.CryptEncrypt
 data
 advapi32.CryptEncrypt

EDI=003A6748

Address	Hex dump	ASCII
003A6748	12 C4 2B 35 EF D1 37 FF	+-+5^07
003A6750	08 54 D2 21 19 FD B3 2F	0T0^+X /
003A6758	3E 97 78 C4 3C BC D2 34	>Sx-<^04
003A6760	AE 7B 77 F9 09 C6 4B F5	<<(w".AKS
003A6768	00 F0 FD 7F 77 7A 7A 70	.-Y0wz2p
003A6770	66 00 00 00 00 00 00 00	f.....

Output is converted using the predefined charset and given to a victim as the individual ID:

Address	Hex dump	ASCII
003A6748	A0 6A 3A 00 90 E8 39 00 AA 10 A9 9D A4 88 6E EB	aj:ER9. teKAknU
003A6758	BB 8F 08 C6 42 60 85 B3 2C FE A9 7F E5 2D 96 10	1C06e...te i-
003A6768	62 AE 0C 73 57 90 FB 59 06 37 B2 5A 6F 5B C7 D2	b<.swEUV7#ZolA0
003A6778	3C 73 49 19 BA 48 5C B4 08 86 EC 36 3F 37 86 2B	<SI+IH\ d9677c+
003A6788	49 A3 45 B0 3A 0B D9 1F 0A C0 8C 1F A1 48 21 F6	IuE2:aj.V.'iVlHf+
003A6798	C9 89 D2 16 BC D3 BC DB 1B AE ED 08 97 49 60 98	FE0.#E#<<V-
003A67A8	06 26 FB 9C 96 03 27 79 35 39 2C D5 01 20 EE BF	*#U#P#y59.RD t
003A67B8	3B 21 22 A0 F1 D6 84 FF D9 F3 E1 0A 3F 7C 50 DA	;f"á'ia'vB.?lPr
003A67C8	2B 17 5F 98 BB FB 06 D7 9F A6 38 80 E0 55 73 AB	+* _qú#i62;C0Us2
003A67D8	22 66 3A FB 4B 83 19 E5 48 3F E2 9E A0 78 82 64	*f:úKá+HHC0xáed
003A67E8	54 E3 64 88 4B 8C 5F 9D 39 72 6E E0 3A D0 F5 88	TAdkKl_L9rn0:8Sk
003A67F8	BB 1E 6C 3F 91 82 E0 38 D7 F6 09 C5 24 D0 08 98	ñ!l?C008i+.+58s
003A6808	9B 21 EB 7C 3B B5 1A E9 FD DB 86 C8 47 CB 8E A4	s!0!;A+0Y#c#GPA
003A6818	8D 06 8D 07 FF CB 14 A2 54 0A CB 12 FA FC D3 C3	2*2. #q0T.#+ REH
003A6828	2C 07 1D 3A AF A1 88 4F 63 24 7F 66 DA 41 D4 B3	.+*:>i0c\$of rAdl
003A6838	0A A1 50 68 BF 7C 10 52 CB 0C 49 C7 A2 1B 8C 57	.iPh!#Rr.Ia0+iW
003A6848	07 00 00 07 3F 58 00 00 A0 6A 3A 00 A0 E8 39 00	...?X..aj:áR9.
003A6858	00 00 00 00 51 57 45 52 54 59 55 50 41 53 44 46	...QWERTYUPASDF
003A6868	47 48 4A 4B 5A 58 43 56 42 4E 4D 30 31 32 33 34	GHJKZXCUBNM01234
003A6878	35 36 37 38 39 00 00 00 88 A9 18 4A 79 58 00 0B	56789...tefJyX.á
003A6888	F0 01 00 00 00 02 00 00 01 00 00 00 58 32 35 48	-0...0...X25H
003A6898	45 2D 4A 35 33 5A 55 2D 51 45 52 44 5A 2D 5A 4E	E-J532U-QERDZ-ZN
003A68A8	55 4A 33 2D 53 45 52 4A 36 2D 4A 36 31 37 45 2D	UJ3-SERJ6-J617E-
003A68B8	55 55 41 53 5A 2D 41 46 47 32 47 2D 38 33 42 30	UUA5Z-AFG2G-83B0
003A68C8	38 2D 32 53 48 43 31 2D 41 55 59 46 5A 2D 47 4A	8-2SHC1-AUYFZ-GJ
003A68D8	48 46 32 2D 57 37 33 32 31 2D 31 34 34 54 4D 0D	HF2-W7321-144TM.
003A68E8	0A 56 4B 46 4B 52 2D 36 54 4B 52 56 2D 53 54 47	.UKFKR-6TKRU-STG
003A68F8	34 42 2D 43 45 35 4D 5A 2D 54 41 48 34 57 2D 4D	4B-CESM2-TAH4W-M
003A6908	50 35 34 31 2D 47 44 33 53 42 2D 48 45 34 33 4A	P541-6D3SB-HE43J
003A6918	2D 5A 46 34 54 4B 2D 5A 4E 5A 54 47 2D 52 37 5A	-ZF4TK-ZNZTG-R7Z
003A6928	42 5A 2D 41 4B 4D 40 32 55 2D 54 36 54 59 4E 2D 35	BZ-AKM2U-T6TVN-5
003A6938	33 4A 37 48 0D 0A 4D 00 55 36 4A 36 2D 42 54 53 4A	3J7H..MU6J6-BTSJ
003A6948	43 2D 46 51 56 51 52 2D 45 48 37 35 35 2D 43 31	C-FQUGR-EH755-C1
003A6958	57 43 4A 2D 37 53 4E 50 54 2D 4D 48 46 42 53 2D	WCJ-7SNPT-MHFBS-
003A6968	51 36 33 38 56 2D 4D 41 53 45 42 2D 52 31 36 48	Q638U-MA5EB-R16H
003A6978	57 2D 50 38 34 50 32 2D 37 45 45 58 38 2D 48 58	W-P84P2-7EEX8-KX
003A6988	41 48 42 2D 44 31 30 46 37 0D 0A 47 46 30 37 31	AHB-D10F7..GF071
003A6998	2D 55 33 37 4B 33 2D 47 4A 35 51 35 2D 57 44 30	-U37K3-GJ5Q5-WD0
003A69A8	50 44 2D 32 45 47 31 36 2D 4B 4D 43 35 52 2D 52	PD-2EG16-KMCSR-R
003A69B8	50 43 42 58 2D 52 38 45 56 33 2D 5A 50 58 51 56	PCBX-R8EU3-ZPXQU
003A69C8	2D 54 44 56 58 4D 2D 53 45 45 46 58 2D 58 48 32	-TDUXM-SEEFX-XK2
003A69D8	33 4A 2D 46 43 48 34 5A 2D 52 4E 42 50 4E 0D 0A	3J-FCH4Z-RNBPN..
003A69E8	58 45 36 58 35 2D 34 57 38 43 54 2D 57 4A 51 4A	XE6X5-4W8CT-WJQJ
003A69F8	55 2D 30 37 31 54 35 2D 44 53 55 5A 57 2D 4A 47	U-071T5-DSUZW-JG
003A6A08	53 5A 41 2D 4B 46 4B 5A 36 2D 34 44 55 30 53 2D	SZA-KFK26-4DU0S-
003A6A18	38 30 48 31 48 2D 43 45 50 32 4A 2D 50 44 53 48	80H1H-CEP2J-PDSK
003A6A28	41 2D 55 58 42 52 38 2D 38 43 31 42 42 2D 53 44	A-UXBR8-8C1BB-SD
003A6A38	51 4E 43 0D 0A 31 43 38 46 37 2D 48 50 5A 32 47	QNC..1C8F7-HPZ2G
003A6A48	2D 51 35 4A 56 4E 2D 46 36 57 58 48 2D 50 4D 55	-Q5JUN-F6WXH-PMU
003A6A58	53 52 2D 38 47 34 48 54 2D 52 4E 59 56 57 2D 44	SR-8G4HT-RNVUW-D
003A6A68	5A 4E 51 33 2D 59 38 4B 5A 4A 2D 4E 59 43 31 47	ZNQ3-Y8KZJ-NVC1G
003A6A78	2D 53 50 52 33 54 2D 55 35 47 44 35 00 00 00 00	-SPR3T-USGDS...

That's why, when the user submit his/her individual ID, the attackers, having the appropriate private key, can decrypt the original data and easily recover the random AES key.

After this operation, the previously generated AES key is used to encrypt files.

First, file content is compressed by a dedicated function (BZip2):

```

10005177 JMP 10005225
1000517C LEA EAX,DWORD PTR SS:[EBP-2C]
1000517F MOV DWORD PTR SS:[EBP-14],0
10005186 PUSH EAX
10005187 LEA EAX,DWORD PTR SS:[EBP-18]
1000518A MOV DWORD PTR SS:[EBP-2C],0
10005191 PUSH EAX
10005192 LEA EAX,DWORD PTR SS:[EBP-14]
10005195 MOV DWORD PTR SS:[EBP-18],0
1000519C PUSH EAX
1000519D PUSH DWORD PTR SS:[EBP-20]
100051A0 PUSH DWORD PTR SS:[EBP-1C]
100051A3 CALL 10005660
100051A8 MOV ECX,DWORD PTR SS:[EBP-1C]
100051AB TEST EAX,EAX

```

Address	Hex dump	ASCII	0280FEDC	01E38ED0	0A00	Arg1 = 01E38ED0
01E38ED0	23 20 2D 2A 2D 6D 6F 64	# -*-mod	0280FEE0	00000DF5	3...	Arg2 = 00000DF5
01E38ED8	65 3A 20 74 63 6C 3B 20	e: tol;	0280FEE4	0280FF20	C0	Arg3 = 0280FF20
01E38EE0	66 69 6C 6C 2D 63 6F 6C	fill-col	0280FEE8	0280FF1C	L C0	Arg4 = 0280FF1C
01E38EE8	75 6D 6E 3A 20 37 35 3B	umn: 75;	0280FEEC	0280FF08	C0	Arg5 = 0280FF08
01E38EF0	20 74 61 62 2D 77 69 64	tab-wid	0280FEF0	01E17AF8	230	
01E38EF8	74 68 3A 20 38 3B 20 63	th: 8; c	0280FEF4	039A33A0	330	
01E38F00	6F 64 69 6E 67 3A 20 69	oding: i	0280FEF8	01E3CBDC	TFN0	
01E38F08	73 6F 2D 6C 61 74 69 6E	so-latin	0280FEFC	00000000	
01E38F10	2D 31 2D 75 6E 69 78 2D	-1-unix	0280FF00	00000DF5	3...	
01E38F18	2D 2A 2D 0D 0A 23 0D 0A	-*-.#..	0280FF04	00000000	

Then, the buffer containing compressed data is AES encrypted - using [CryptEncrypt](#)

```

100051D2 JNZ SHORT 100051D9
100051D4 MOV ECX,DWORD PTR SS:[EBP-14]
100051D7 JMP SHORT 10005225
100051D9 PUSH DWORD PTR SS:[EBP-24]
100051DC LEA EAX,DWORD PTR SS:[EBP-18]
100051DF PUSH EAX
100051E0 PUSH DWORD PTR SS:[EBP-14]
100051E3 MOV EAX,DWORD PTR DS:[EDI+14]
100051E6 PUSH 0
100051E8 PUSH 1
100051EA PUSH 0
100051EC PUSH DWORD PTR SS:[EBP-C]
100051EF CALL ADVAPI32.CryptEncryptA
100051F1 TEST EAX,EAX

```

Address	Hex dump	ASCII
01E4CC10	7A 03 00 00 42 5A 68 39	z*.BZh9
01E4CC18	31 41 59 26 53 59 37 6B	1AV%SV7k
01E4CC20	A8 35 00 00 57 58 80 00	25..WCC.
01E4CC28	10 48 04 7E 4A 00 0A BF	H* J..
01E4CC30	E1 1F CA 30 00 08 41 29	pT*0.eA)
01E4CC38	26 9A 18 9A 00 00 34 00	&Utu.d4.
01E4CC40	C6 4C 4D 30 9A 62 60 26	ALM00b'&
01E4CC48	98 04 49 4D 09 3D 4C 98	sIM,=Ls
01E4CC50	99 3D 4F D4 8D 30 9B CD	0=0d'20T=
01E4CC58	84 CF 6C B2 98 A9 65 46	3R1#saeF
01E4CC60	0C 51 84 42 42 44 64 52	.0aBBdR
01E4CC68	FF DC 75 EB 73 FC F4 3A	u0r~:
01E4CC70	70 08 98 0A 45 75 8D A4	p0T.Eu2a
01E4CC78	B1 21 A8 50 57 4C 1C 53	EPWLLS
01E4CC80	26 C1 41 1C 31 AC 0A 08	&+AL1C.#
01E4CC88	19 22 07 95 40 E2 00 05	J* C00.#
01E4CC90	52 BF 66 05 06 6E C8 14	R1 f#an#q
01E4CC98	B8 DC 2F A0 FC 7B BF FA	S=/aRc1'
01E4CCA0	77 FD C9 CC 81 20 64 51	wRfH d0
01E4CCA8	A0 FC B0 48 C6 79 74 C7	aR:HAytã
01E4CCB0	D2 C4 17 C9 3A 33 B3 D4	D-#F:31d'
01E4CCB8	56 43 EA 20 40 26 50 39	UC# @&P9
01E4CCC0	04 78 D2 17 D9 E2 FE A3	*x0# 0=u
01E4CCC8	06 98 49 1C 55 F5 97 23	#TILUS3#
01E4CCD0	EF 96 15 B2 86 E4 A4 BB	'P330nAn
01E4CCD8	60 00 16 AF C5 DC 91 4E	'-#-CN
01E4CCE0	14 24 0D DA EA CD 40 00	q\$.ri=@.
01E4CCE8	00 00 00 00 00 00 00 00

The encrypted data is saved to the file with the generated extension added.

Conclusion

Maktub Locker has clearly been developed by professionals. The full product's complexity suggests that it is the work of a team of people with different areas of expertise. From the packing operations to the website, everything is well-polished. We are not sure if the crypter/FUD is designed by the same team - it could also be a commercial solution available on the black market. However, it is not the only level of defense - the core DLL is also obfuscated and for sure prepared by someone with experience in writing malware.

Malwarebytes Anti-Malware detects this threat as: **Ransom.Maktub**.

Appendix

<http://www.bleepingcomputer.com/news/security/the-art-of-the-maktub-locker-ransomware/> - "The Art of the Maktub Locker Ransomware" (detailed description of the graphical design)