

# Maktub Locker - Beautiful And Dangerous | Malwarebytes Labs

By hasherezade

Published: 2016-03-23 · Archived: 2026-04-05 16:48:45 UTC

*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.

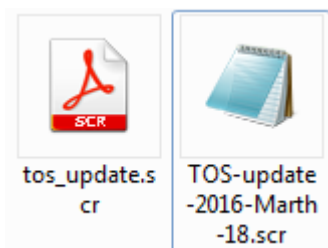
## Analyzed samples

- [74add6536cdcfb8b77d10a1e7be6b9ef](#)
- [b24952857ff5cb26b2e97331800fa142](#) <- main focus of this analysis
  - [38eff2f7c6c8810a055ca14628a378e7](#) – payload (C.dll)

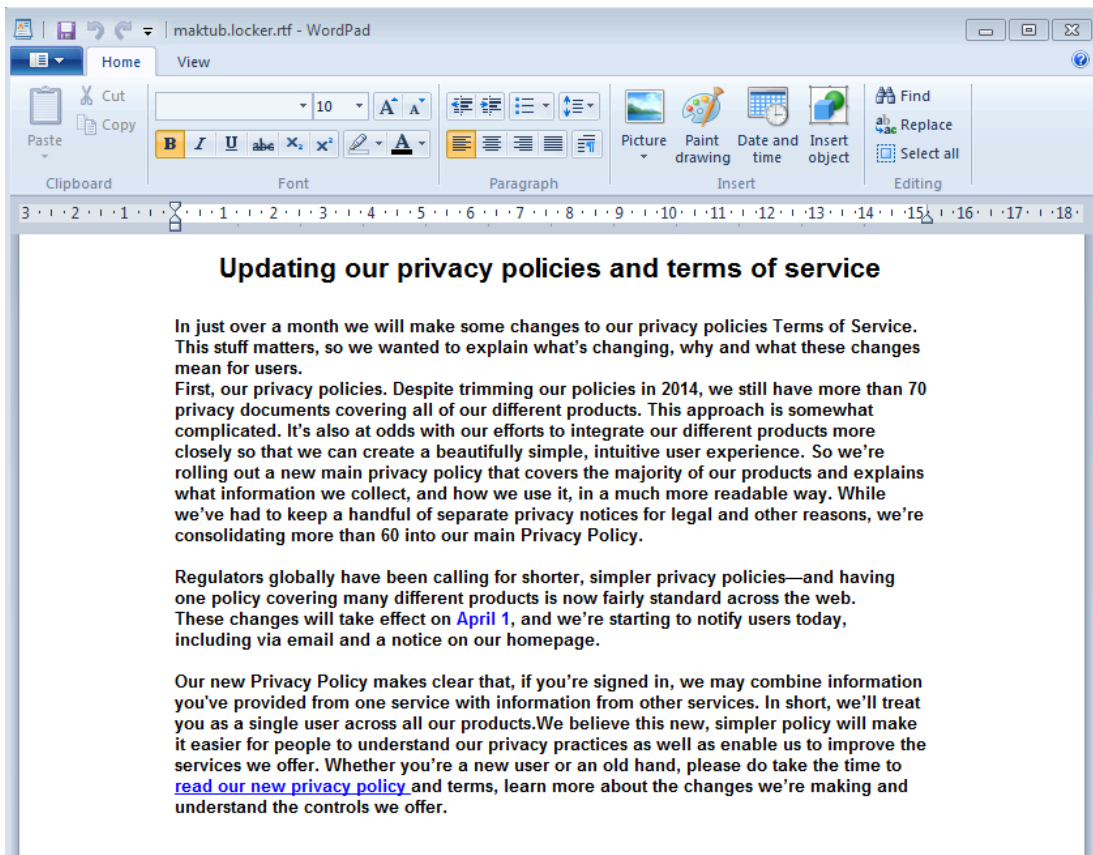
Special thanks to [MalwareHunterTeam](#) and [Yonathan Klijnsma](#) for sharing the samples.

## 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:

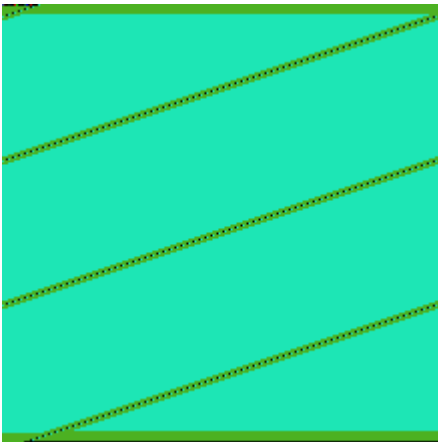
square.gif	2016-02-22 01:14	14 KB
square.jpg	2016-02-22 01:14	5 KB
square.png	2016-01-20 19:19	2 KB
square1.bmp	2016-01-20 19:21	140 KB
square2.bmp	2016-02-22 01:15	48 KB
square3.bmp	2016-02-22 01:15	24 KB
square4.bmp	2016-02-22 01:15	7 KB
tekst.txt	2016-01-31 16:50	1 KB

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-CRB7J-7EEN4-SAFY0-EQ4FA-XCTUQ-S6MDE-UQF5D-8AJCQ-32X2E-YRXDK  
HSFX2-6823X-YH0JR-EYP05-CXGDY-WVJXJ-PMKUS-XYMJJ-48RHF-6QTPQ-HD2TE-RAM1Y-6HBNW-KT8PG  
MA5W6-1AZHK-7FD5Y-KK5JD-WPQC1-CRVNM-VP846-SUM4A-X58MW-HUJUJ-QRP28-4TJ2R-RKGC0-0SUDQ  
4VQEG-1EANX-CT507-HSJGC-UZWGG-2ZYN4-UUP8D-K7PPM-S8NC1-US74N-U46BW-DRAC5-UJZU0-8FTDA  
DUTEB-ZVTPA-W7MAV-3WJ7X-0AF5F-Y7C7E-QYU2U-V82JH-4C2D7-2R6YT-T2671-HCU5B-VECE6-D343J  
BTR2F-G35NS-KAE6E-5TFWK-RTVEN-KGUG2-CDN43-FMUJG-W25MY-2DN55-TJMSM-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.

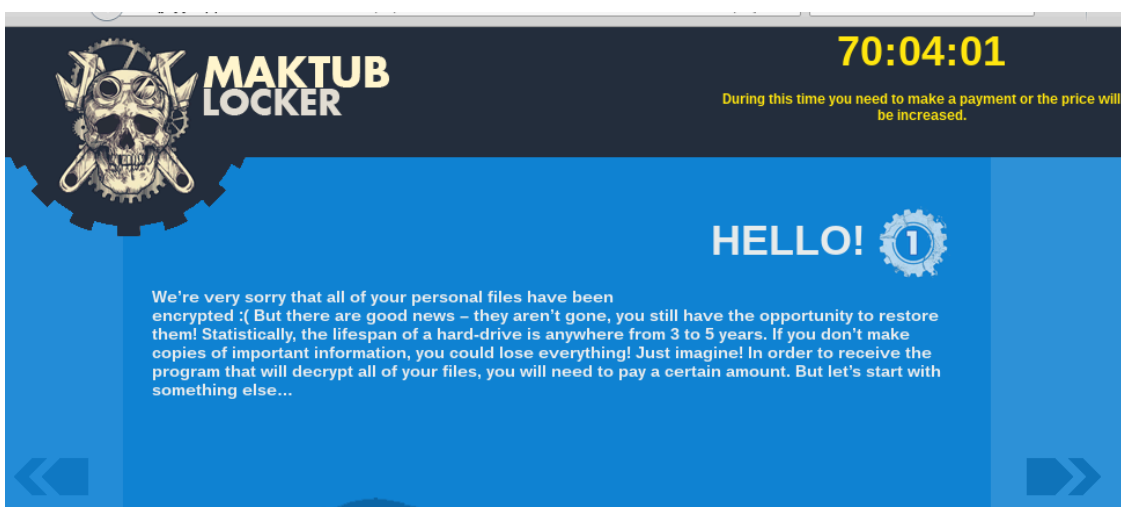
### Website for the victim

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.



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

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!

Files available to decrypt: 0

Number	File Name	Size	Link
1	square1_bmp.png	631 bytes	<a href="#">Download</a>
2	square2.bmp	7626 bytes	<a href="#">Download</a>

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.

```

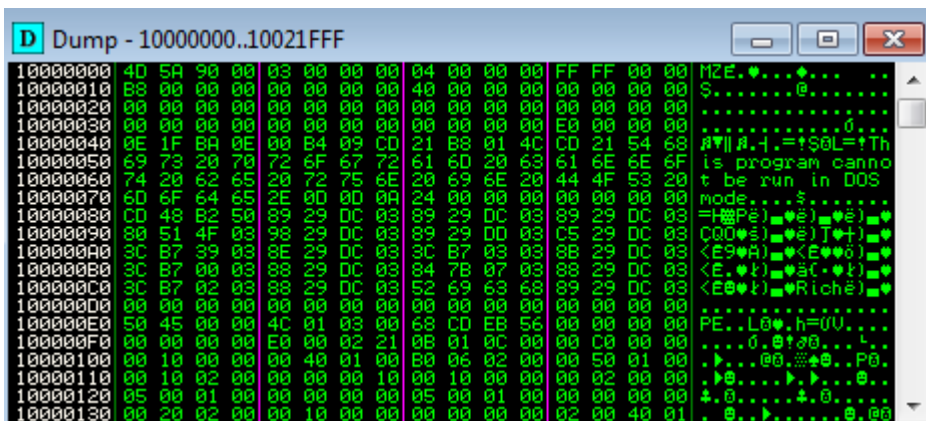
004016E4 . C78424 400100 MOV DWORD PTR SS:[ESP+14C],tos_upda.004 ASCII "AtInTSU names Uline prompt "
004016F6 . C78424 500100 MOV DWORD PTR SS:[ESP+150],tos_upda.004
00401700 . 85C9 TEST EAX,EAX
00401702 . 7E 0E JLE SHORT tos_upda.00401712
00401704 . 0FB6D1 MOVZX EDX,CL
00401707 . 0FAFD0 IMUL EDX,EAX
0040170A . 83FA ADD EDI,EDX
0040170C . 893D C8AA4100 MOV DWORD PTR DS:[41AAC0],EDI
00401712 > 3935 D0AA4100 CMP DWORD PTR DS:[41AA00],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 580100 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 Adio damp "
00401741 . 89D C4AA4100 MOV DWORD PTR DS:[41AAC4],EAX
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
0040175F . 8B424 3C MOV EAX,DWORD PTR SS:[ESP+3C]
00401763 . 8B424 24 MOV EDI,DWORD PTR SS:[ESP+24],EAX
00401767 > C78424 680100 MOV DWORD PTR SS:[ESP+168],tos_upda.004 ASCII "costsVow 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 JNE SHORT tos_upda.0040178F
00401781 . 8B5424 24 MOV EDX,DWORD PTR SS:[ESP+24]
00401785 . 3BD1 SUB EBX,EBX
    
```

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):

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

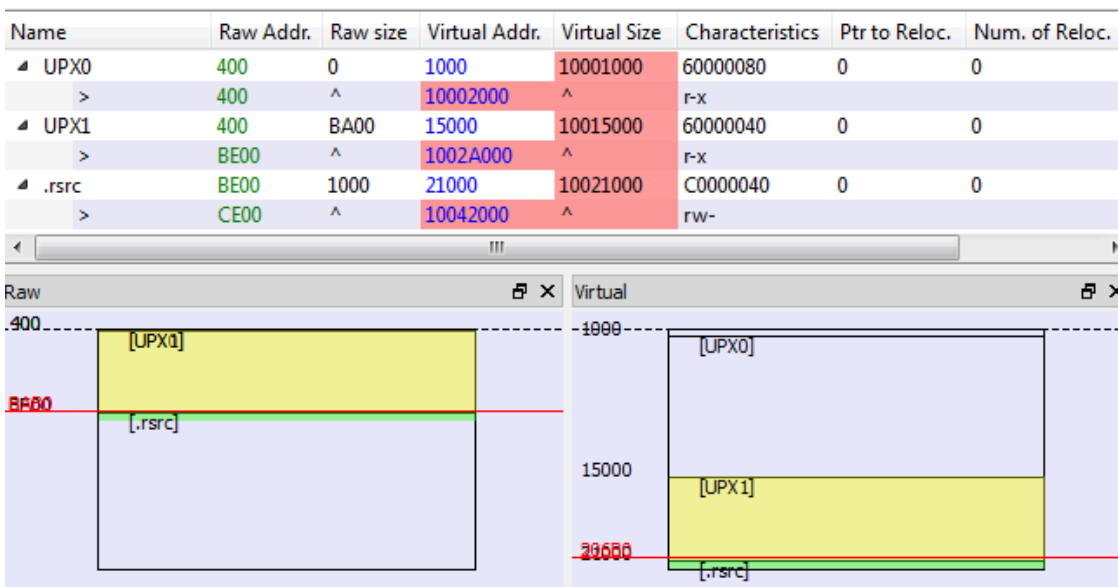
Ident	Entry	Data block	Last error	Status	Priority	User time	System time
0000044C	10001230	7FFD9000	ERROR_SUCCESS (00)	Active	32 + 0	20.9200 s	6.4592 s
000007FC	01C357A4	7FFD0000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0100 s
000009DC	00406D13	7FFDF000	ERROR_RESOURCE_TVI	Active	32 + 0	2.2732 s	0.1201 s
00000AE4	773AFD0F	7FFDE000	ERROR_SUCCESS (00)	Active	32 + 0	0.0000 s	0.0000 s
00000B18	10001230	7FFD9000	ERROR_SUCCESS (00)	Active	32 + 0	21.4187 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:



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:



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):

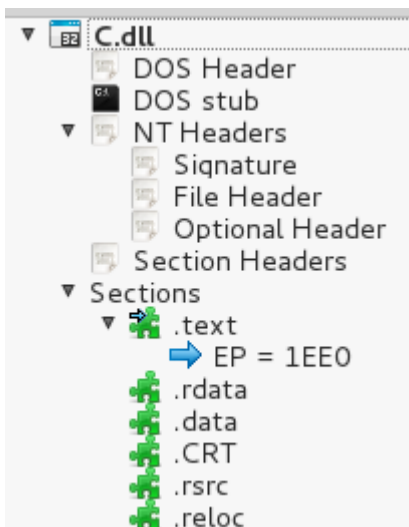
Offset	Name	Value	Meaning
CD68	Characterist...	0	
CD6C	TimeDateSt...	56EBCD67	
CD70	MajorVersion	0	
CD72	MinorVersion	0	
CD74	Name	21FA4	C.dll
CD78	Base	1	
CD7C	NumberOf...	2	
CD80	NumberOfN...	2	
CD84	AddressOff...	21F90	

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:*Ű+óĚĹ-ĈíĚ-~ŰŽú«A"
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 80000000
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
10004FDC 83F8 FF     CMP EBX,-1
    
```

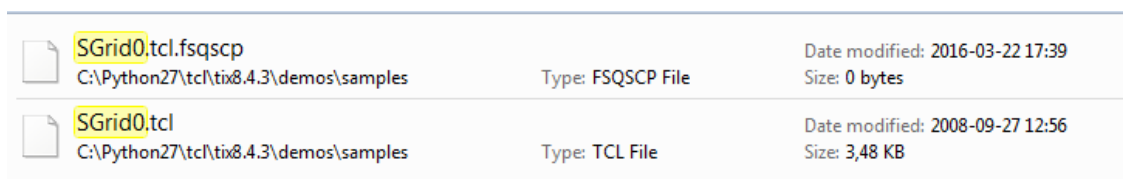
Then, a new file is created – with an extension added:

```

100050FA 89 70100000 MOV EDI,DWORD PTR DS:[EDI]
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
1000510F 68 00000040 PUSH 40000000
10005114 56         PUSH ESI
10005116 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
1000511E 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.



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:

```

0012773 8D45 FC      CALL EBX
10012776 C745 FC 00000000 LEA EAX,DWORD PTR SS:[EBP-4]
1001277D 50          PUSH EAX
1001277E 6A 01       PUSH 1
10012780 57          PUSH EDI
10012781 FF15 30310110 CALL DWORD PTR DS:[10013130]      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:[10018680]
100127B6 8935 BC860110 MOV DWORD PTR DS:[100186BC],ESI
100127BC FF15 A4300110 CALL DWORD PTR DS:[100130A4]      USER32.ShowDialogParamA
100127C2 FF75 F8    PUSH DWORD PTR SS:[EBP-8]
100127C5 FF15 30310110 CALL DWORD PTR DS:[10013130]      gdiplus.GdiplusShutdown
100127CB 5F         POP EDI
100127CD 5E         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.GetKeyboardLayoutList] GetKeyboardLayoutList
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 . JLE 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)\;\micro
```

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 60 00 88 03 00 00 00 hm.k0...
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.k0...
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.k0...
03AB334 03 00 00 00 01 00 00 00 *.0...
03AB33C 70 65 66 00 48 55 EE 70 pef.HUtp
03AB344 65 74 00 88 03 00 00 00 et.k0...
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.k0...
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.k0...
03AB37C 03 00 00 00 01 00 00 00 *.0...
03AB384 70 67 70 00 51 55 EE 70 pgp.QUtp
03AB38C 74 6D 00 88 03 00 00 00 tm.k0...
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 *.k0...
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 .i.k0...
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 .i.k0...
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.k0...
03AB3F4 03 00 00 00 01 00 00 00 *.0...
03AB3FC 70 73 74 00 A0 55 EE 70 pst.aUtp
03AB404 00 8C 00 88 03 00 00 00 .i.k0...
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:

```

[code]X25HE-J53ZU-QERDZ-ZNUJ3-SERJ6-J617E-UUASZ-AFG2G-83B08-2SHC1-AUYFZ-GJHF2-W7321-
144TM VKFKR-6TKRV-STG4B-CE5MZ-TAH4W-MP541-GD3SB-HE43J-ZF4TK-ZNZTG-R7ZBZ-AKM2U-
T6TYN-53J7H MU6J6-BTSJC-FQVQR-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-WJQU-071T5-DSUZW-JGSZA-KFKZ6-4DU0S-80H1H-
CEP2J-PDSKA-UXBR8-8C1BB-SDQNC 1C8F7-HPZ2G-Q5JVN-F6WXH-PMUSR-8G4HT-RNYVW-DZLNQ3-
Y8KZJ-NYC1G-SPR3T-U5GD5 [/code]
    
```

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 | flags = CRYPT_VERIFYCONTEXT | CRYPT_CREATE_SALT
0F80DFC6 | . PUSH 0xF0000040 | dwProvType = PROV_DH_SCHANNEL
0F80DFCB | . PUSH 0x18 |
0F80DFCD | . LEA ESI,DWORD PTR DS:[EDI+0x100] |
0F80DFD3 | . PUSH DWORD PTR DS:[EAX] | pszProvider = "Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)"
0F80DFD5 | . MOV EAX,DWORD PTR DS:[EDI+0x4] | advapi32.CryptAcquireContextA
0F80DFD8 | . PUSH 0x0 | pszContainer = NULL
0F80DFDA | . PUSH ESI | *phProv
0F80DFDB | . CALL EAX | advapi32.CryptAcquireContextA
    
```

EAX=77519100 (advapi32.CryptAcquireContextA)

Gets 32 random bytes, using function [CryptGenRandom](#)

```

0F8029A3 . PREFIX REP: Superfluous prefix
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 advapi32.CryptGenRandom
0F8029B3 . CALL one.0F80E740
    
```

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
0F957DFE . PUSH 0x8003 MD5_SUM
0F957E04 . PUSH DWORD PTR DS:[ESI+0x100]
0F957E08 . MOV EAX,DWORD PTR DS:[ESI+0x8]
0F957E0D . CALL EAX advapi32.CryptCreateHash
0F957E0F . TEST EAX,EAX advapi32.CryptHashData
0F957E11 . JZ SHORT one.0F957E2E advapi32.CryptHashData
0F957E13 . MOV EAX,DWORD PTR DS:[ESI+0xC]
0F957E16 . PUSH 0x0
0F957E18 . PUSH 0x20 dataLen = 32
0F957E1A . PUSH [ARG.1] pbData
0F957E1D . PUSH [LOCAL.1] hash
0F957E20 . CALL EAX advapi32.CryptHashData
0F957E22 . TEST EAX,EAX advapi32.CryptHashData
0F957E24 . JNZ SHORT one.0F957E37
    
```

Address	Hex dump	ASCII
003A5A60	12 C4 2B 35 EF D1 37 FF 8B 54 D2 21 19 FD B3 2F	⚡+5'07 0T0!+x /
003A5A70	3E 97 78 C4 3C BC D2 34 AE 7B 77 F9 09 C6 4B F5	>3x-@04«w"._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 #phKey
0F957E3B . MOV EAX,DWORD PTR DS:[ESI+0x24] advapi32.CryptDeriveKey
0F957E3E . PUSH 0x0 flags
0F957E40 . PUSH [LOCAL.1] hBaseData
0F957E43 . PUSH 0x6610 AlgID
0F957E48 . PUSH DWORD PTR DS:[ESI+0x100] hProv
0F957E4E . CALL EAX advapi32.CryptDeriveKey
    
```

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

```

0F95E772 . MOV EAX,DWORD PTR DS:[EBX+0x20] advapi32.CryptDestroyKey
0F95E775 . CALL EAX advapi32.CryptImportKey
0F95E777 . LEA EAX,[LOCAL.1]
0F95E77A . MOV [LOCAL.1],0x0
0F95E781 . PUSH EAX advapi32.CryptImportKey
0F95E782 . MOV EAX,DWORD PTR DS:[EBX+0x1C] advapi32.CryptImportKey
0F95E785 . PUSH 0x0
0F95E787 . PUSH 0x0
0F95E789 . PUSH 0x114
0F95E78E . PUSH one.0F968438
0F95E793 . PUSH DWORD PTR DS:[EBX+0x100]
0F95E799 . CALL EAX 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	0.0.0.0.RSA1.0.
0F968448	01 00 01 00 55 70 84 A9 B8 6F ED 2E 51 35 5B B9	0.0.UpaeSo?,05[
0F968458	AC C6 31 A3 C8 DF 75 61 53 42 1B D0 77 3F F8 AE	CAIU=wasB+dw?<<
0F968468	CB C4 75 87 AE 9F 0A 92 97 AF EF 8D 93 5E C1 5A	[-uc<<C.[S>^20^z
0F968478	9B F5 36 69 A2 B0 1F E8 00 15 0F D6 25 5B 9F 83	T36i0:YR.S*%Lca
0F968488	FB EC 96 37 D7 41 BC A1 67 34 0B CD 71 84 E2 AC	0U!P!P!ig4=q00C
0F968498	BD C0 D3 00 8F 0E 56 6C 50 FC 4F 79 AA A8 77 0A	2!E.C#U!P!P!y Ew.
0F9684A8	6C CE F0 E9 96 9C 9F E0 3C 47 D1 3E AF 24 17 B0	l[-U!P!P!G0>>>S#
0F9684B8	4F 30 35 11 47 D3 DA 74 4B 25 17 92 16 7D 9B C9	0054GE!tK%#[-]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	=02 !!YEU2e20g0#
0F9684E8	E2 8E 5A 6A E3 A2 67 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.-teS#0f
0F968508	89 61 76 7D 5B 8D 11 55 FE 33 DA 8F 5F 24 6F 52	0av)l24U#3rC_0r
0F968518	93 D1 6F D4 4D 9A 11 94 58 7F 74 7E 48 8A 86 41	000MU!0X0t*H00A
0F968528	2F B8 C3 5B EF 4F 41 E6 F4 51 24 BB 45 61 59 E3	^l 'OAS~0\$EayN
0F968538	B4 F3 F3 9C A8 8D A6 B9 4C 1B 3D F2 0E 7D 84 7D	+^*eE22!L+=.0)0}
0F968548	59 E7 34 AD 00 00 00 00 01 00 00 00 EB 03 00 00	Y\$4s.....0...U#..

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] dataLen = 0x2c
0F95E87F . PUSH EDI data
0F95E880 . PUSH 0x0
0F95E882 . PUSH 0x1
0F95E884 . PUSH 0x0
0F95E886 . PUSH [LOCAL.1]
0F95E889 . CALL EAX advapi32.CryptEncrypt

```

EDI=003A6748

Address	Hex dump	ASCII
003A6748	12 C4 2B 35 EF D1 37 FF	0017F7E8 003A6550
003A6750	8B 54 D2 21 19 FD B3 2F	0017F7EC 00000000
003A6758	3E 97 78 C4 3C BC D2 34	0017F7F0 00000001
003A6760	AE 7B 77 F9 09 C6 4B F5	0017F7F4 00000000
003A6768	00 F0 FD 7F 77 7A 7A 70	0017F7F8 003A6748
003A6770	66 00 00 00 00 00 00 00	0017F7FC 0017F814
		0017F800 00000100
		0017F804 7E031000

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



```

100051D2 JNZ SHORT 100051D9
100051D4 MOV ECX,DWORD PTR SS:[EBP-14]
100051D7 JMP SHORT 10005225
100051D9 PUSH DWORD PTR SS:[EBP-24]          buffer_len
100051DC LEA EAX,DWORD PTR SS:[EBP-18]
100051DF PUSH EAX
100051E0 PUSH DWORD PTR SS:[EBP-14]        data_len
100051E3 MOV EAX,DWORD PTR DS:[EDI+14]      data
100051E6 PUSH 0                               flags
100051E8 PUSH 1                               final
100051EA PUSH 0                               hash
100051EC PUSH DWORD PTR SS:[EBP-C]         key
100051EF CALL EAX                             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 68	1AV&SY7k
01E4CC20	AB 35 00 00 57 5B 80 00	25..WDC.
01E4CC28	10 48 04 7E 4A 00 0A BF	▶H*J..7
01E4CC30	E1 1F CA 30 00 08 41 29	βT=0.8A)
01E4CC38	26 9A 18 9A 00 00 34 00	&ü†ü.84.
01E4CC40	06 4C 4D 30 9A 62 60 26	ALM0üb'&
01E4CC48	98 04 49 4D 09 3D 4C 98	šIM.=Lš
01E4CC50	99 3D 4F D4 8D 30 9B CD	0=0f20T=
01E4CC58	84 CF 6C B2 98 A9 65 46	šššššššššš
01E4CC60	0C 51 84 42 42 44 64 52	.Qšššššššš
01E4CC68	FF DC 75 EB 72 FC F4 3A	šššššššššš
01E4CC70	70 0B 9B 0A 45 75 8D A4	p0T.Eu2A
01E4CC78	B1 21 A8 50 57 4C 1C 53	šššššššššš
01E4CC80	26 C1 41 1C 31 AC 0A 08	&+A.L.C.š
01E4CC88	19 22 07 95 40 E2 00 05	↓".E00.š
01E4CC90	52 BF 66 05 06 6E C8 14	R7fšššššššš
01E4CC98	B8 DC 2F A0 FC 7B BF FA	Sššššššššš
01E4CCA0	77 FD C9 CC 81 20 64 51	wšššššššššš
01E4CCA8	A0 FC B0 48 C6 79 74 C7	šššššššššš
01E4CCB0	D2 C4 17 C9 3A 33 B3 D4	C-šššššššššš
01E4CCB8	56 43 EA 20 40 26 50 39	UCšššššššššš
01E4CCC0	04 78 D2 17 D9 E2 FE A3	šššššššššš
01E4CCC8	06 9B 49 1C 55 F5 97 23	šššššššššš
01E4CCD0	EF 96 15 B2 86 E4 A4 B8	'šššššššššš
01E4CCD8	60 00 16 AF C5 DC 91 4E	šššššššššš
01E4CCE0	14 24 0D DA EA CD 40 00	šššššššššš
01E4CCE8	00 00 00 00 00 00 00 00	šššššššššš

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)

## About the author

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