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.

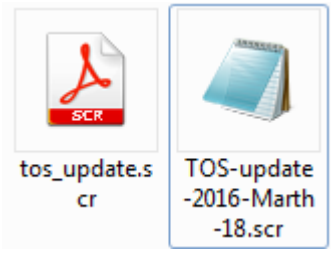
Analyzed samples

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

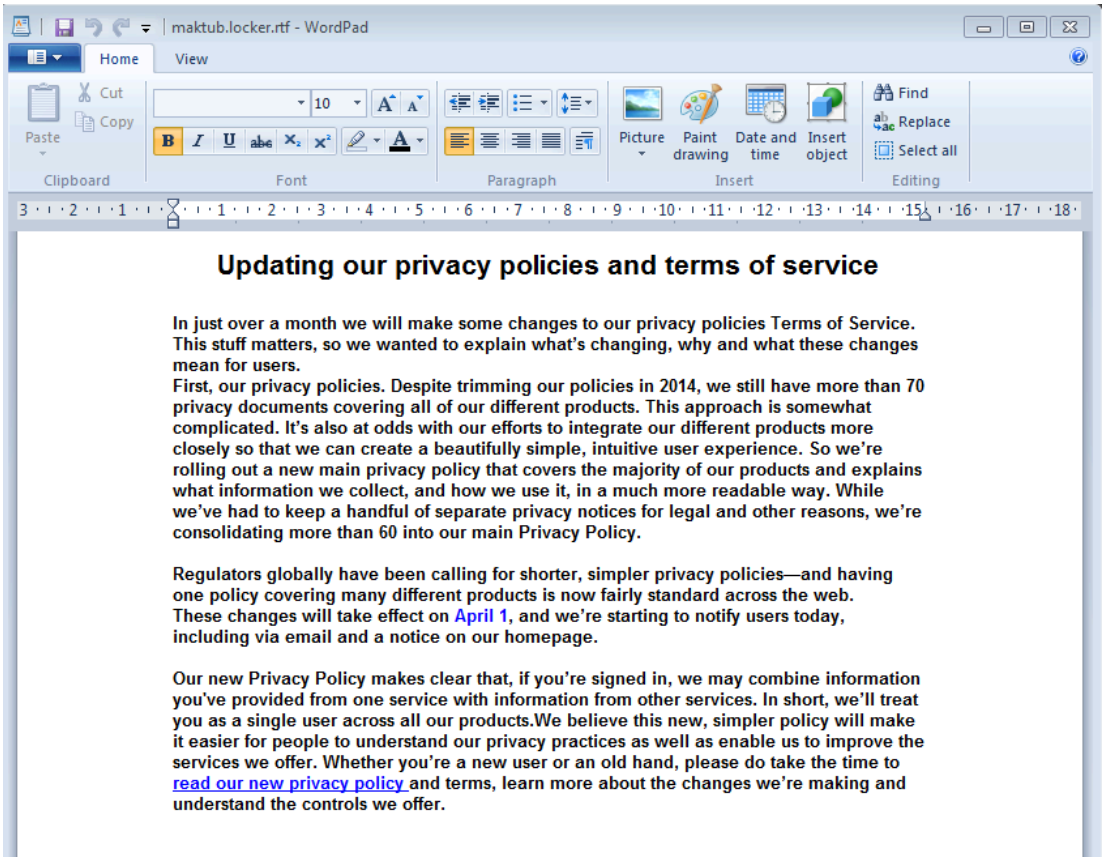
Special thanks to MalwareHunterTeam and Yonathan Klijnsma for sharing the samples.

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:



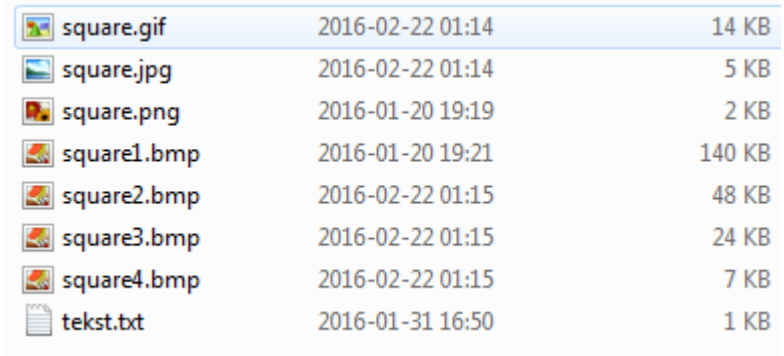
While the user is busy reading the document, the malicious program runs in the background and encrypts his/her files.

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:

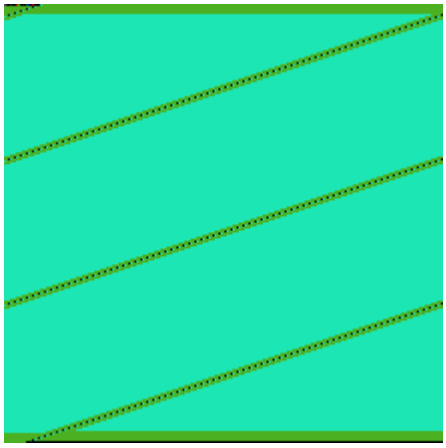


The same files after encryption:

_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:

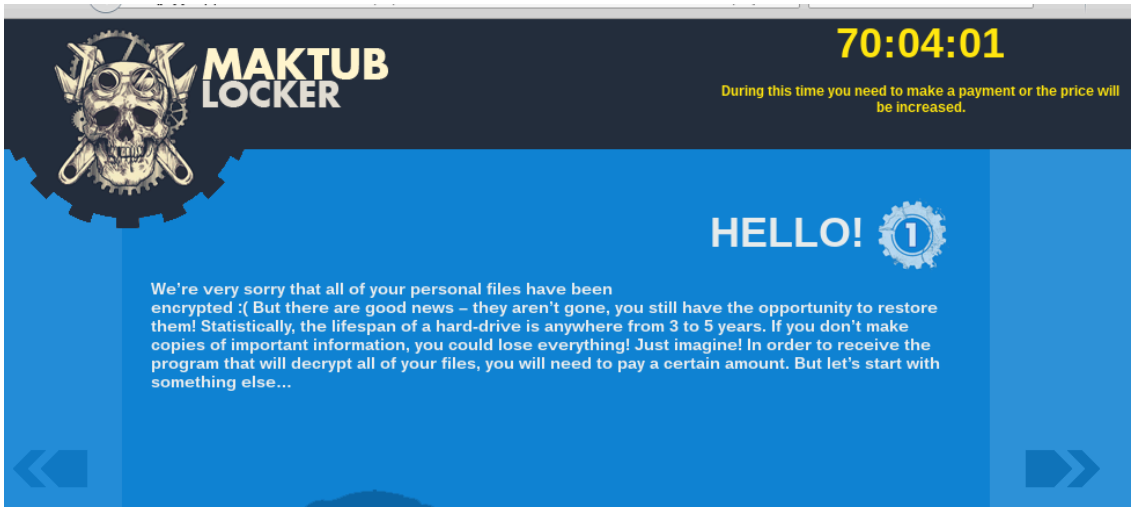
It provides a victim a custom-formatted key: 82 chunks, each 5 character long (chunk format: [A-Z0-9]{5}). Each time the sample runs, this key is newly generated.

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

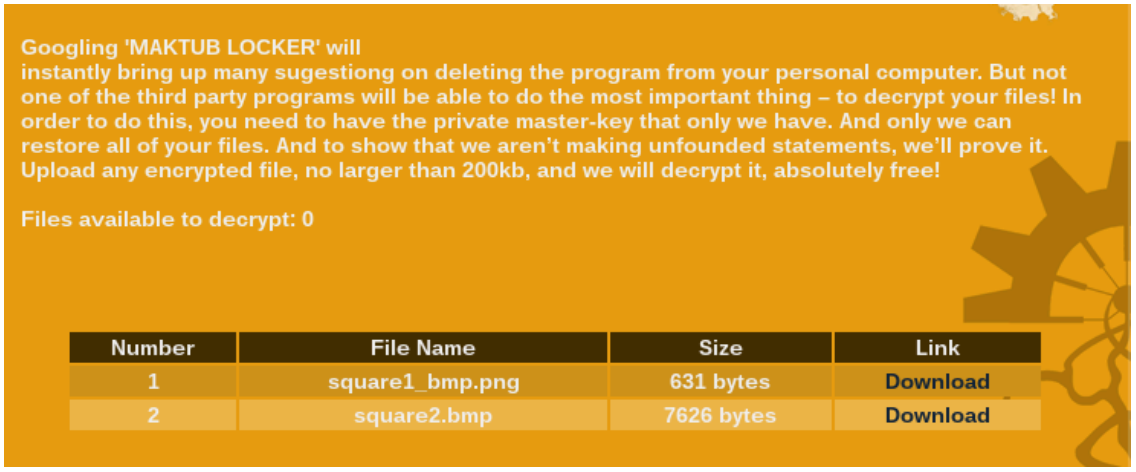
Website for the victim

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.

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:



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.

```
004016EA . C78424 4C0100 MOV DWORD PTR SS:[ESP+14C],tos_upda.004  ASCII "MiniSD names Vine 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:[41AAC8],EDI
00401712 > 3935 D0AA4100 CMP DWORD PTR DS:[41AAD0],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 Adic damp "
00401741 . 890D C4AA4100 MOV DWORD PTR DS:[41AAC4],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
0040175F . 8B4424 3C     MOV EAX,DWORD PTR SS:[ESP+3C]
00401763 . 894424 24     MOV DWORD PTR SS:[ESP+24],EAX
00401767 > C78424 680100 MOV DWORD PTR SS:[ESP+168],tos_upda.004  ASCII "costsYou 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]
```

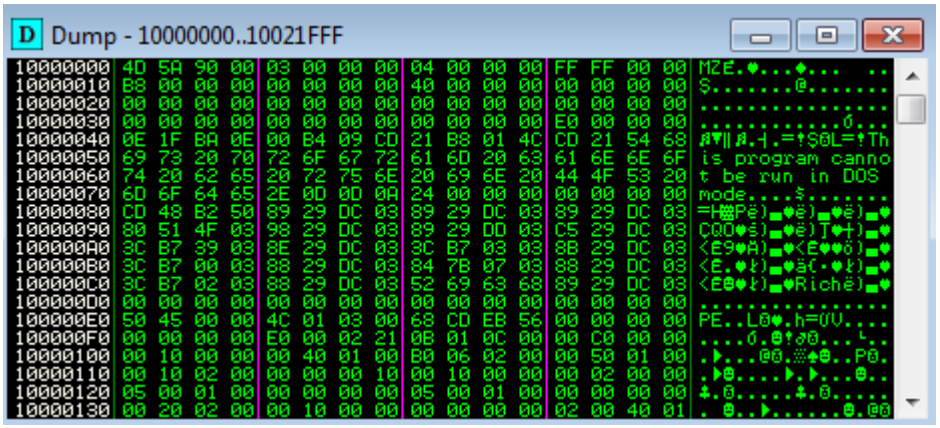
This code is executed first, to deceive tools used to detect malicious behavior. Then it is completely overwritten by new code. However, this is also not the malware code, but just another layer of deception techniques. Below, you can see a fragment of the code responsible for unpacking and executing 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.

Threads

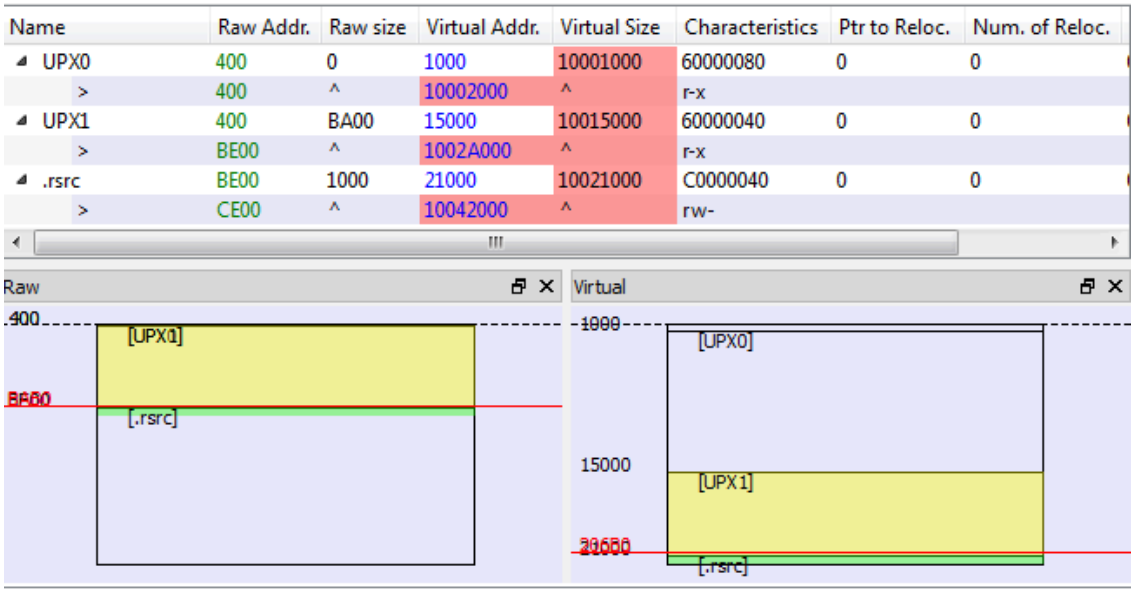
Ident	Entry	Data block	Last error	Status	Priority	User time	System time
0000044C	10001230	7FFD9000	ERROR_SUCCESS (00	Active	32 + 0	20.9200 s	6.4592 s
000007FC	01C357A4	7FFDD000	ERROR_SUCCESS (00	Active	32 + 0	0.0000 s	0.0100 s
000009DC	00406D13	7FFDF000	ERROR_RESOURCE_TY	Active	32 + 0	2.2732 s	0.1201 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:



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):

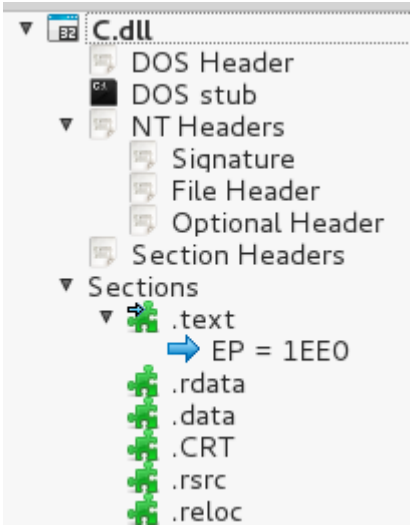
Offset	Name	Value	Meaning
CD68	Characterist...	0	
CD6C	TimeDateSt...	56EBCD67	
CD70	MajorVersion	0	
CD72	MinorVersion	0	
CD74	Name	21FA4	C.dll
CD78	Base	1	
CD7C	NumberOfF...	2	
CD80	NumberOfN...	2	
CD84	AddressOfF...	21F90	

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 aPKVisMbiJgod ; "Ć:*Ő+óĘĹ-ČíĘ¬-ÜŽú«Ą"
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.

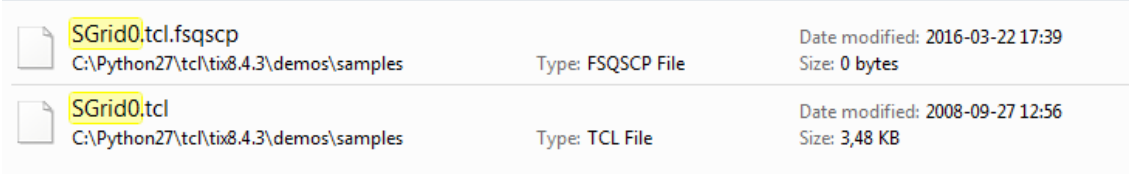
Below you can see a fragment of code opening file that is chosen to be encrypted. Call to the function CreateFileA is performed via handle and dynamically loaded into the EAX register:

```
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 00000008     PUSH 8000000
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
```

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

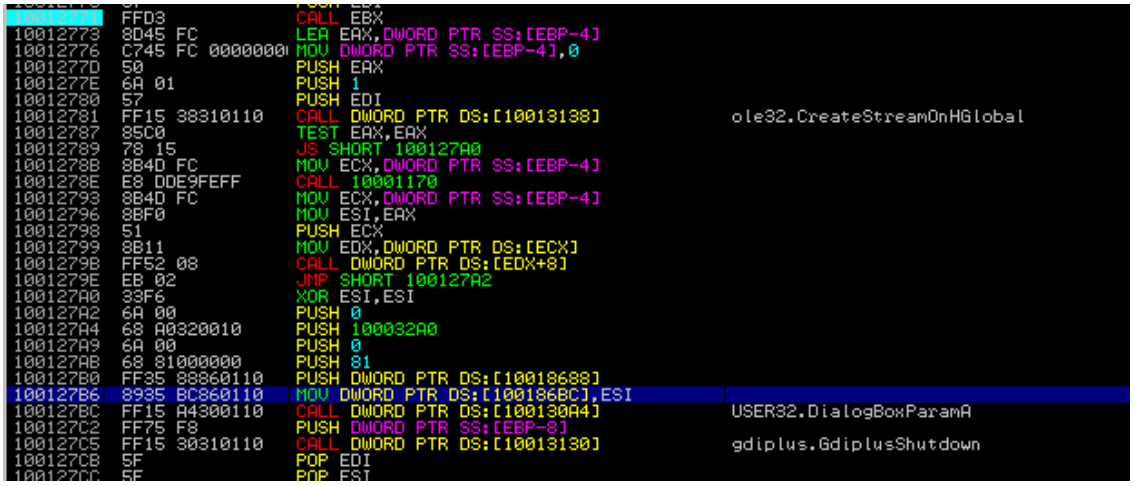
```
100050FF  8B75 F0         MOV ESI,DWORD PTR SS:[EBP-10]
10005102  8B47 54         MOV EAX,DWORD PTR DS:[EDI+54]
10005105  6A 00           PUSH 0
10005107  6A 00           PUSH 0
10005109  6A 02           PUSH 2
1000510B  6A 00           PUSH 0
1000510D  6A 00           PUSH 0
1000510F  68 00000040     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
EAX=7581CEE8 (kernel32.CreateFileA)
0280FED4  01E2451C LE00 ASCII "C:\Python27\tcl\tix8.4.3\demos\samples\SGrid0.tcl.fsqscp"
0280FED8  40000000 ...@
0280FEDC  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.



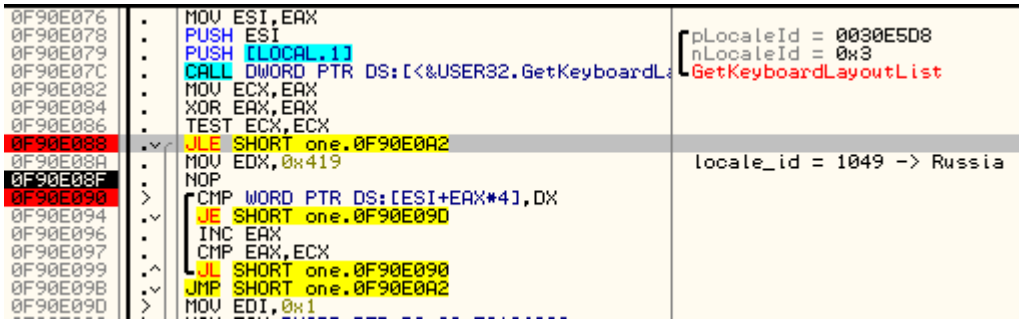
After the process of encryption finished, the malware creates and pops up the dialog box.

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



What is attacked?

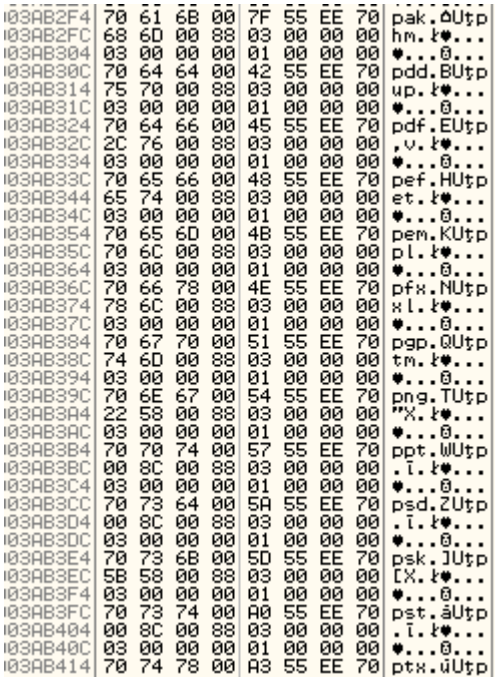
It is common practice to exclude some chosen countries from the attack. In this case, before deploying the malicious actions, the application fetches the keyboard locale list. If it finds [Russian (value 0x419 = 1049)] among them, the malware exits without infecting files:



Excluded from the attack are also some predefined folders:

```
"\internet explorer\;\history\;\mozilla\;\chrome\;\temp\;\program files\;\program files (x86)\;\micr
```

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



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?

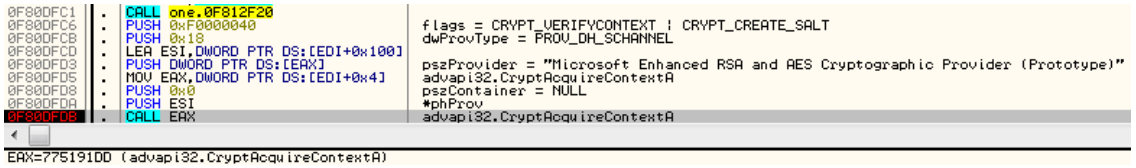
Maktub Locker uses Window Crypto API. But, as we concluded from the analysis, it uses only one key for all files (does not generate a random key per file). Let's see what technique it uses to obtain keys…

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

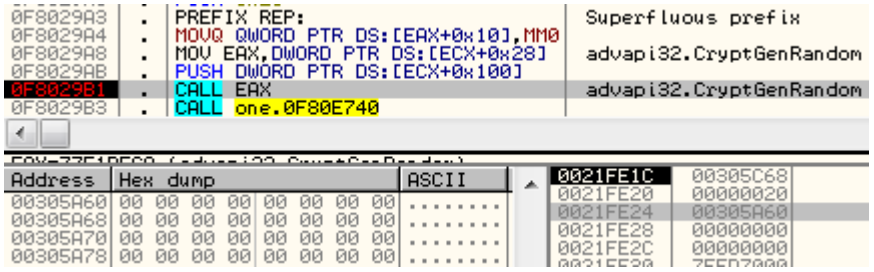
[code]X25HE-J53ZU-QERDZ-ZNUJ3-SERJ6-J617E-UUASZ-AFG2G-83B08-2SHC1-AUYFZ-GJHF2-W7321-144TM VKFKR-6TKRV-STG4B-CE5MZ-TAH4W-MP541-GD3SB-HE43J-ZF4TK-ZNZTG-R7ZBZ-AKM2U-T6TYN-53J7H MU6J6-BTSJC-FQVQR-EH755-C1WCJ-7SNPT-MHFBS-Q638V-MASEB-R16HW-P84P2-7EEX8-KXAHB-D10F7 GF071-U37K3-GJ5Q5-WD0PD-2EG16-KMC5R-RPCBX-R8EV3-ZPXQV-TDVXM-SEEFX-XK23J-FCH4Z-RNBPN XE6X5-4W8CT-WJQJU-071T5-DSUZW-JGSZA-KFKZ6-4DU0S-80H1H-CEP2J-PDSKA-UXBR8-8C1BB-SDQNC 1C8F7-HPZ2G-Q5JVN-F6WXH-PMUSR-8G4HT-RNYVW-DZNQ3-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.

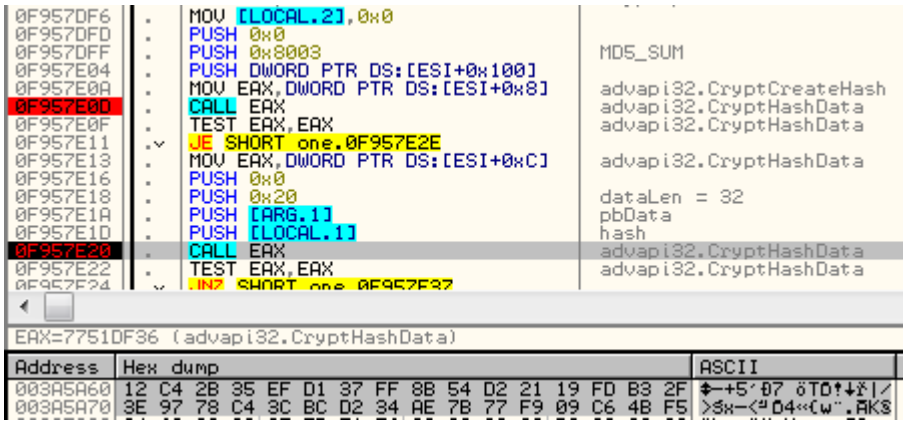
First it initialized two crypto contexts – both with the same settings, using provider type: PROV_DH_SCHANNEL



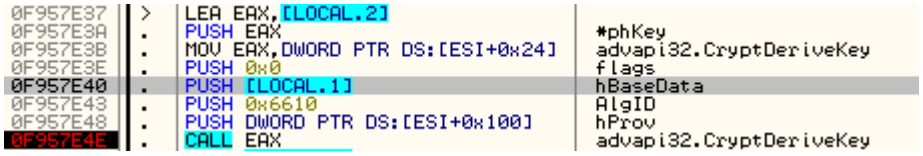
Gets 32 random bytes, using function CryptGenRandom



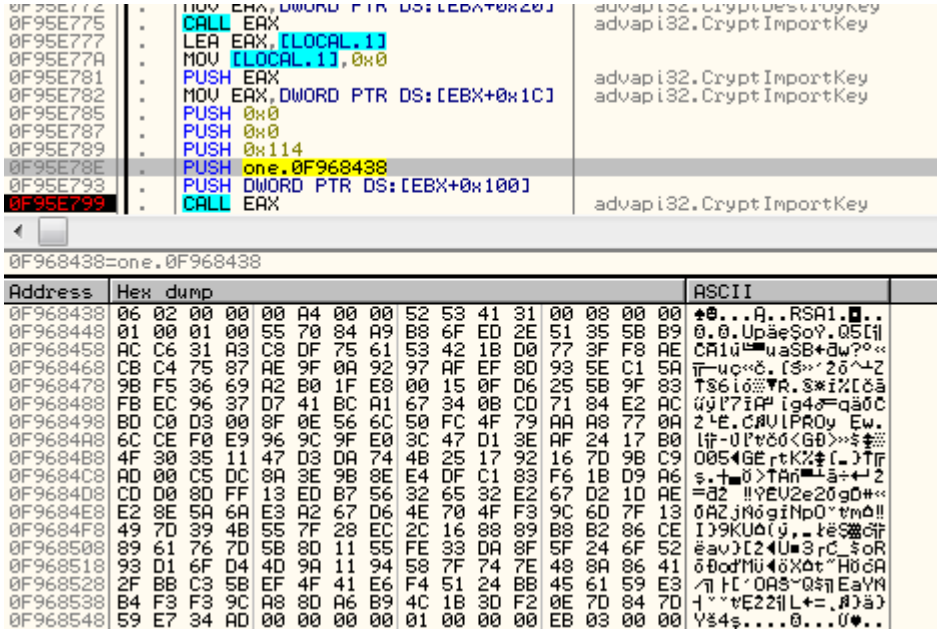
Creates MD5 sum of this random data (using: CryptCreateHash, CryptHashData)



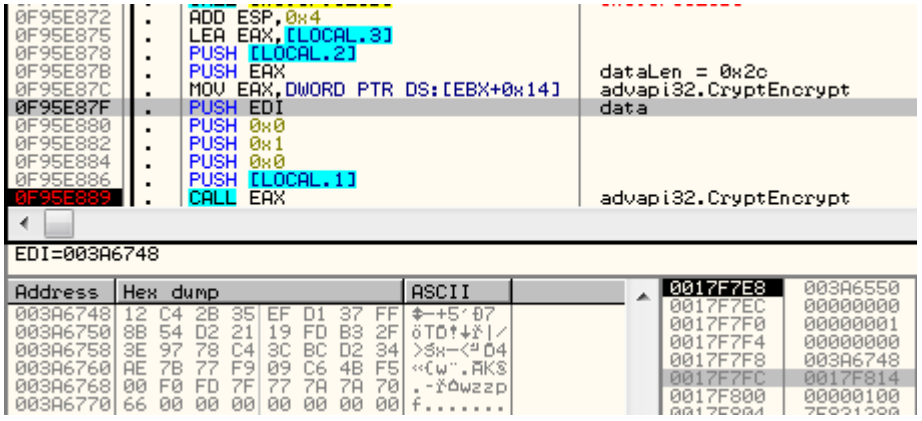
Then, using function CryptDeriveKey it converts the MD5 hash into a 256 bit AES key (AlgID = 0x6610 -> CALG_AES_256).



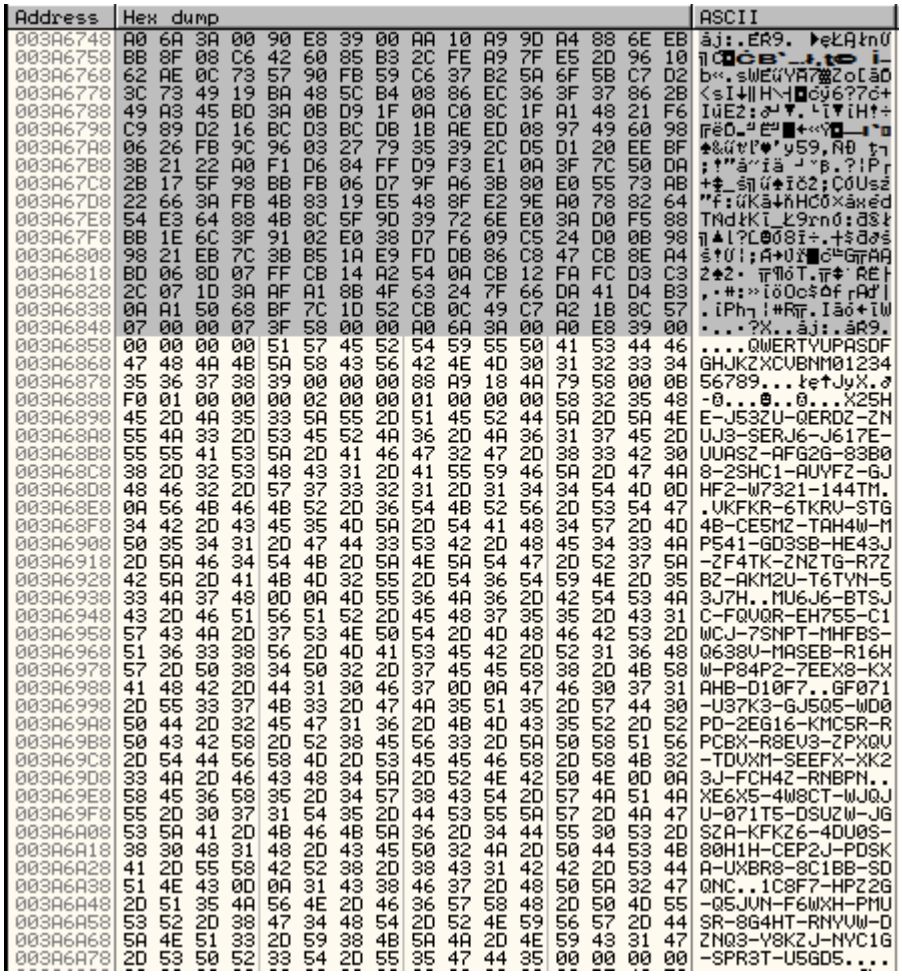
It also imports RSA public key (2048 bit). This key is hardcoded in the binary.



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



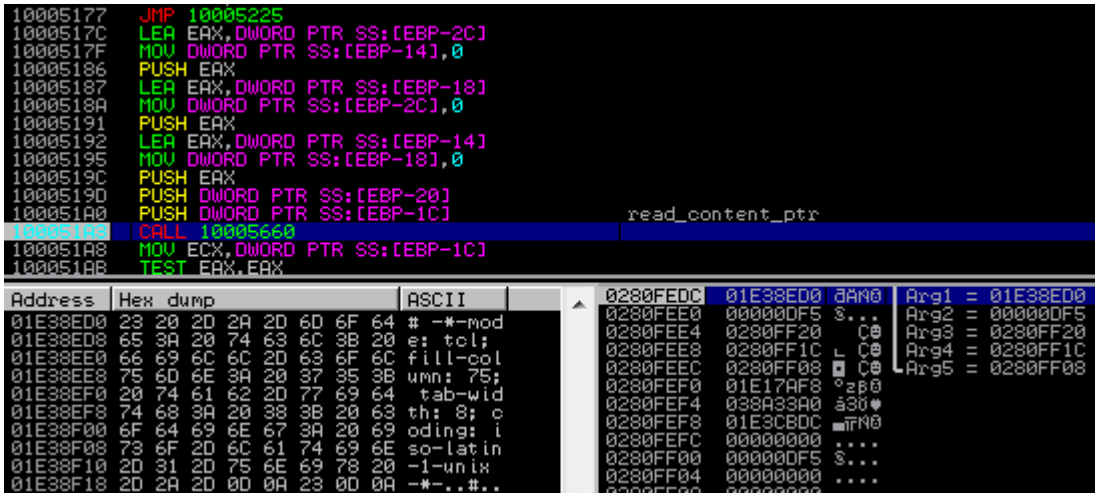
Output is converted using the predefined charset and given to a victim as the individual ID:



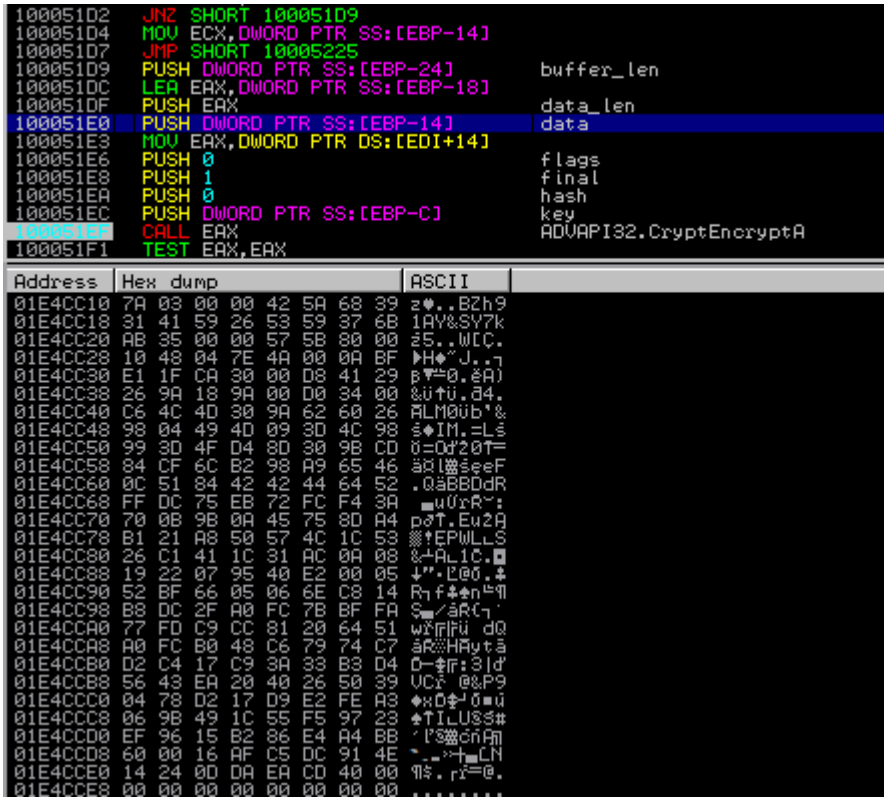
That's why, when the user submit his/her individual ID, the attackers, having the appropriate private key, can decrypt the original data and easily recover the random AES key.

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

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



Then, the buffer containing compressed data is AES encrypted – using CryptEncrypt



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.

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