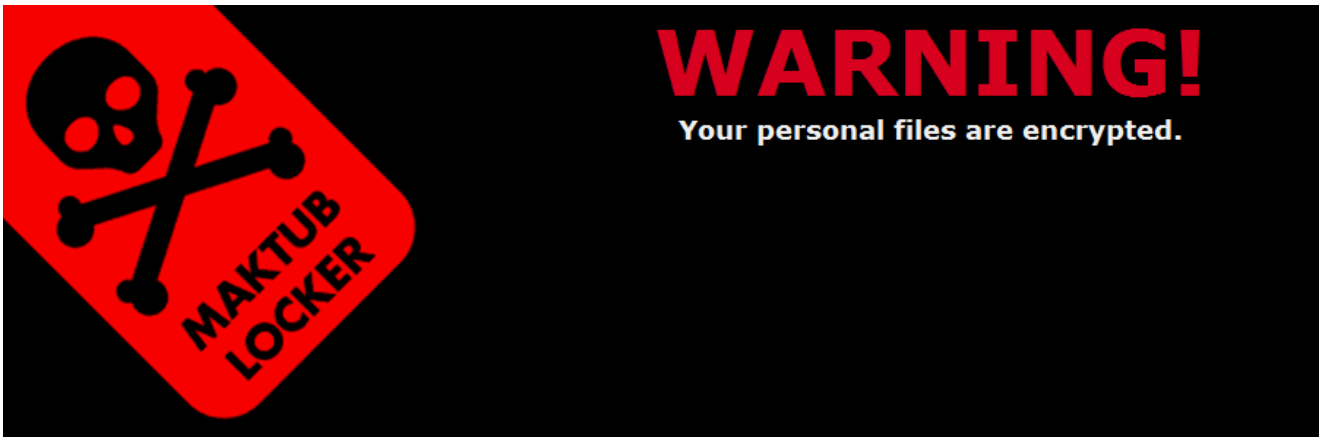
Maktub Locker – Beautiful And Dangerous

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

hasherezade March 24, 2016



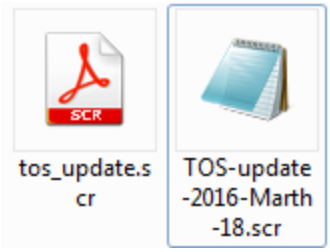
Maktub Locker is another ransomware that comes with a beautifully designed GUI and few interesting features. Its name originates from the Arabic word maktub which means “this is written” or “this is fate”. The authors were probably trying to make a joke by referencing the act of getting infected with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

Analyzed samples

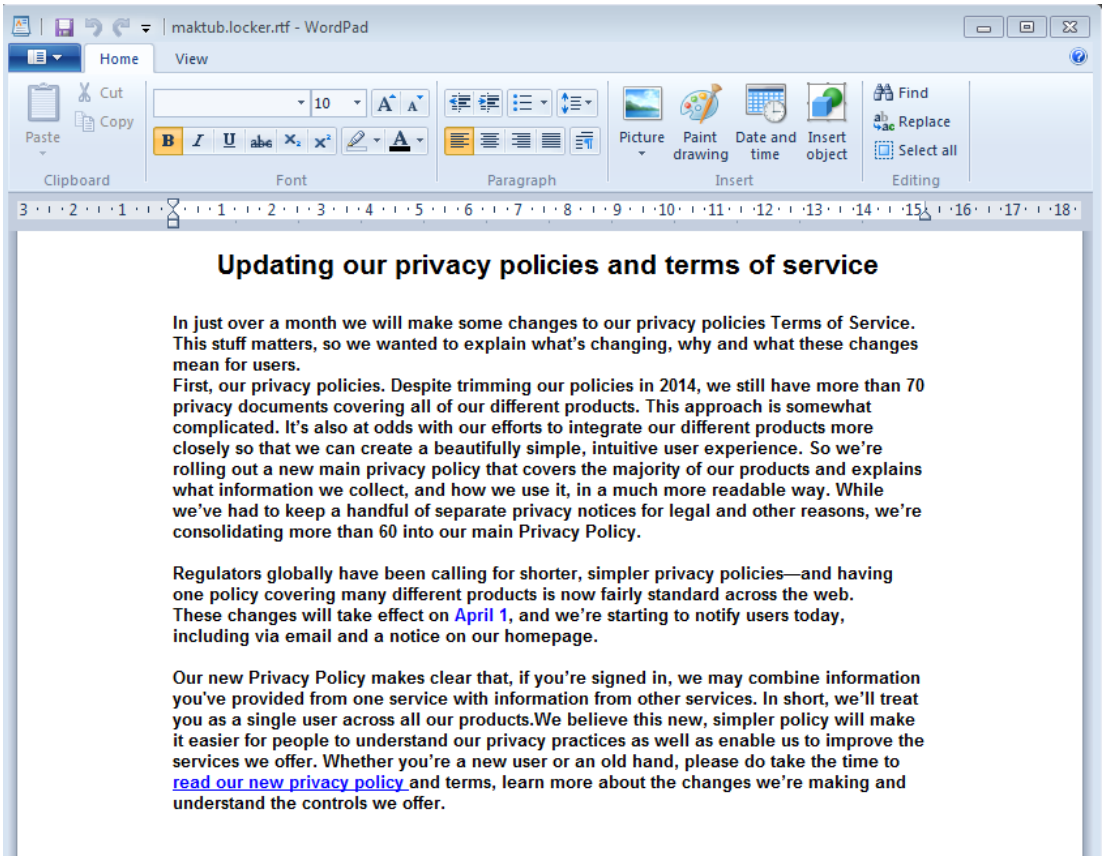
Special thanks to MalwareHunterTeam and Yonathan Klijnsma for sharing the samples.

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

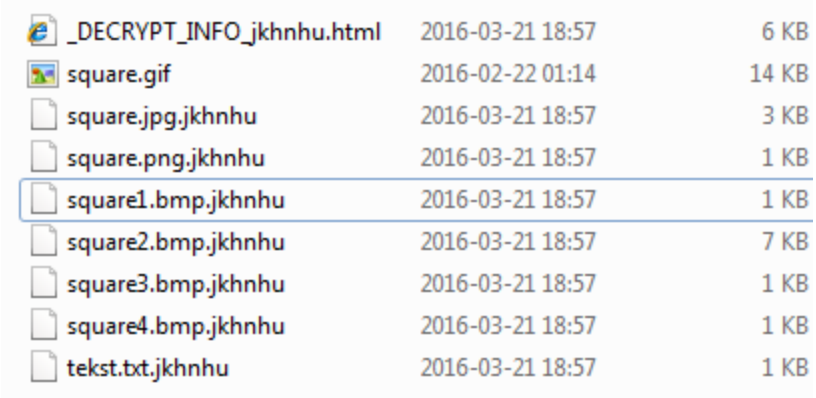
Maktub Locker does not need to download a key from the CnC server – data can be encrypted offline as well. Extensions given to the encrypted files are random, generated at runtime – their pattern is: **[a-z]{4,6}**

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:

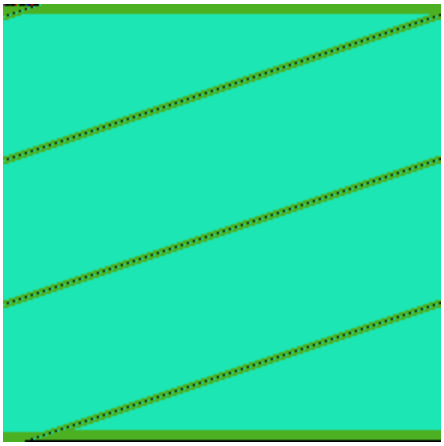
square.gif	2016-02-22 01:14	14 KB
square.jpg	2016-02-22 01:14	5 KB
square.png	2016-01-20 19:19	2 KB
square1.bmp	2016-01-20 19:21	140 KB
square2.bmp	2016-02-22 01:15	48 KB
square3.bmp	2016-02-22 01:15	24 KB
square4.bmp	2016-02-22 01:15	7 KB
tekst.txt	2016-01-31 16:50	1 KB

The same files after encryption:



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.

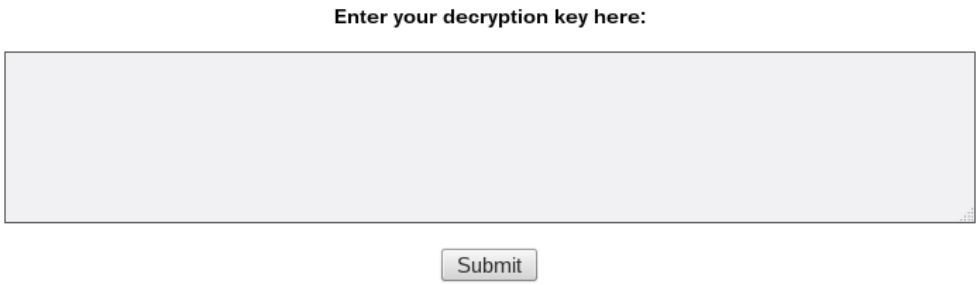
After the encryption is finished, the following GUI pops up:

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.

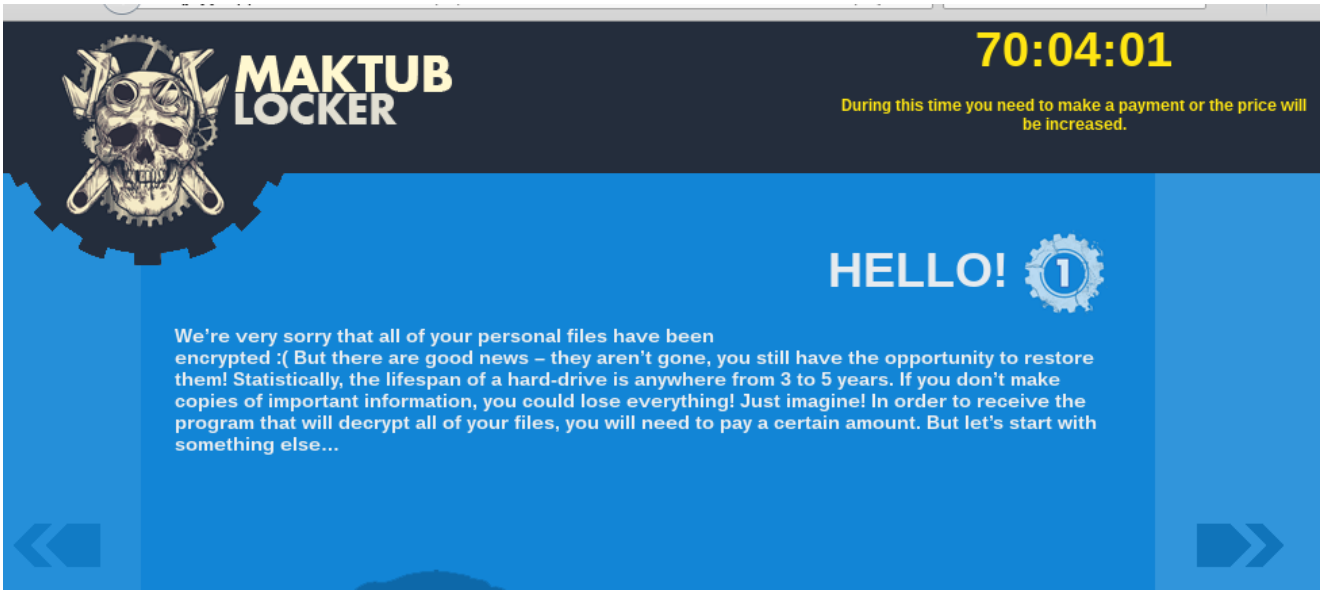
The same information (and layout) can be found in an HTML file (**_DECRYPT_INFO_[$EXTENSION].html**), dropped in each encrypted directory.

Website for the victim

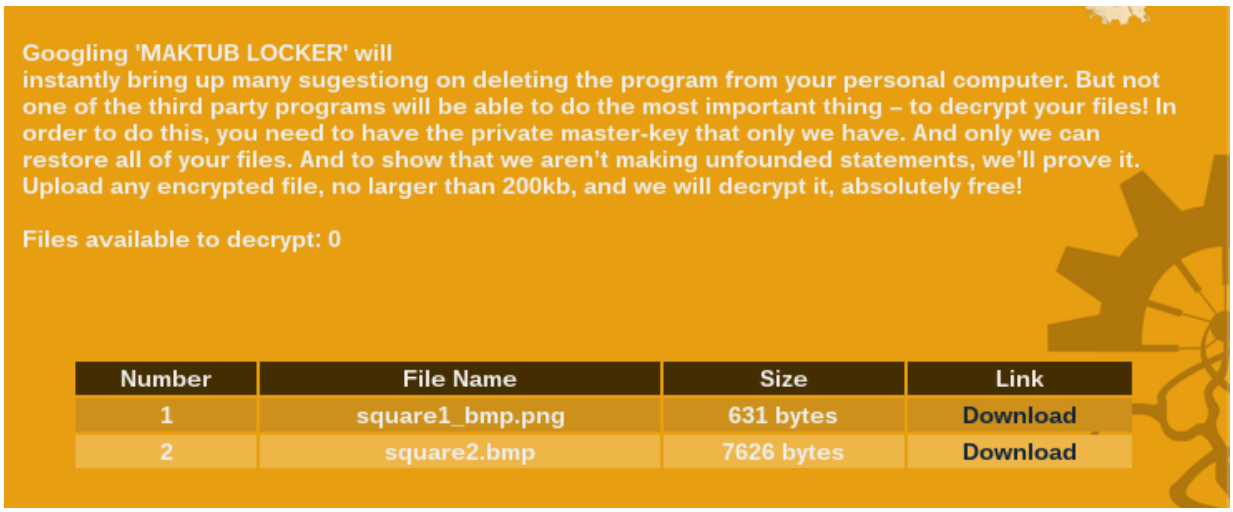
These days, it's a common feature of ransomware to provide a TOR-accessed website for the victim and Maktub Locker is no different. Similar to the ransom note, the website is only available in English. In order to access the individual page, the victim is supposed to paste his/her key (the one supplied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



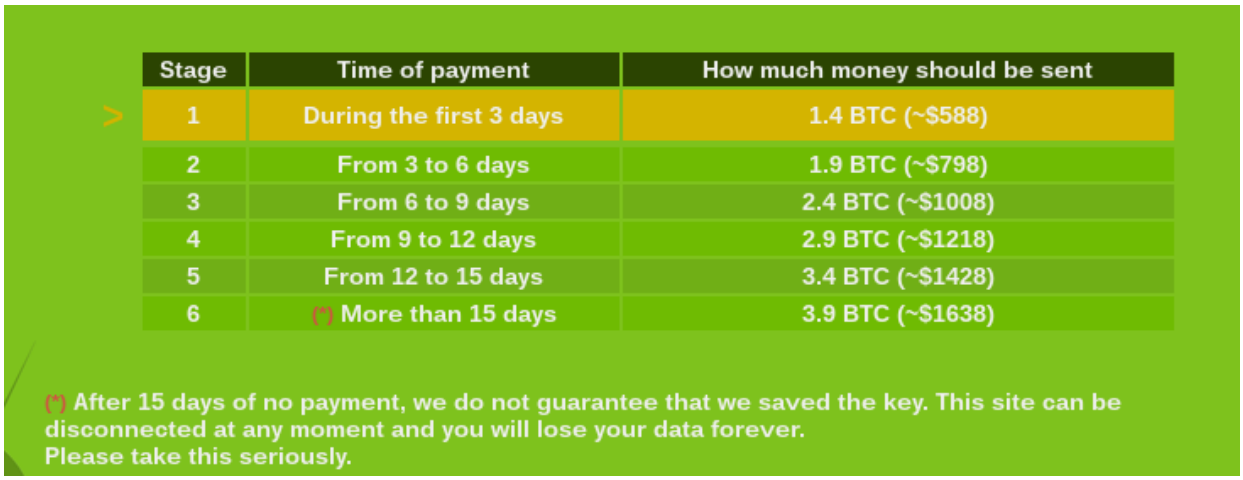
It then redirects to the main website. In comparison to other ransomware families, Maktub Locker actually has a very nicely designed website, including clean and polite language used.



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 "MtnISD 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):

```
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                             ┌IsShown = A (10.)
00401F78  .  PUSH ESI                            │DefDir
00401F79  .  PUSH ESI                            │Parameters
00401F7A  .  PUSH DWORD PTR DS:[402A28]          │FileName = "C:\Users\tester\AppData\Local\Temp\tos_update.rtf"
00401F80  .  PUSH ESI                            │Operation
00401F81  .  PUSH ESI                            │hWnd
00401F82  .  CALL DWORD PTR DS:[402A0C]          └ShellExecuteW
00401F88  .  LEA EAX,DWORD PTR SS:[EBP-20]
00401F8B  .  PUSH EAX                            ┌MutexName
00401F8C  .  PUSH ESI                            │InitialOwner
00401F8D  .  PUSH ESI                            │pSecurity
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                           ┌FreeType = MEM_RELEASE
00401FA6  .  PUSH ESI                            │Size
00401FA7  .  PUSH EBX                            │Address
00401FA8  .  CALL DWORD PTR DS:[402A74]          └VirtualFree
00401FAE  .  PUSH ESI                            ┌ExitCode
00401FAF  .  CALL DWORD PTR DS:[402B48]          └ExitProcess
```

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:

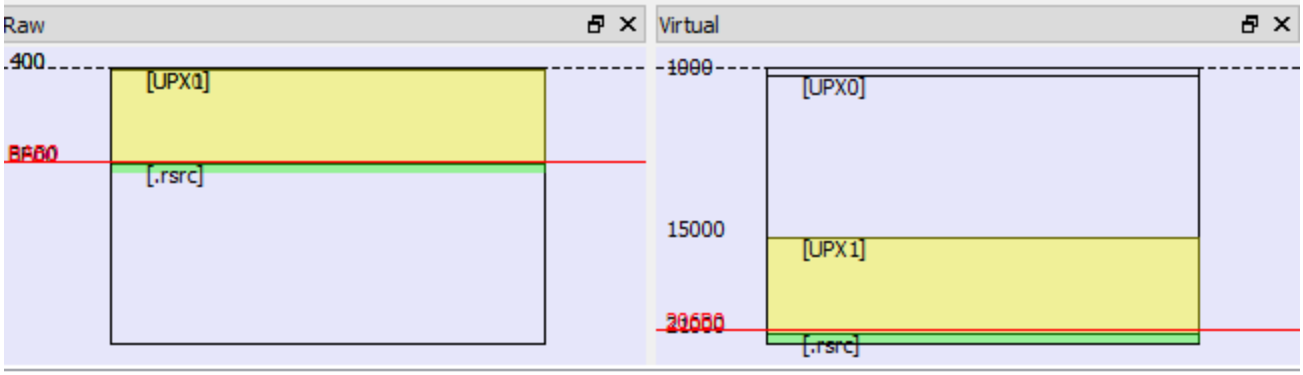
Dump - 10000000..10021FFF

```
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 S.......@.......
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 ............ó...
10000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ♫▼‖♂.┤.=↑$ØL=↑Th
10000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
10000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
10000070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.......
10000080 CD 48 B2 50 89 29 DC 03 89 29 DC 03 89 29 DC 03 =H▒Pë)▄♥ë)▄♥ë)▄♥
10000090 80 51 4F 03 98 29 DC 03 89 29 DD 03 C5 29 DC 03 ÇQO♥ś)▄♥ë)▌♥┼)▄♥
100000A0 3C B7 39 03 8E 29 DC 03 3C B7 03 03 8B 29 DC 03 <╖9♥Ä)▄♥<╖♥♥ö)▄♥
100000B0 3C B7 00 03 88 29 DC 03 84 7B 07 03 88 29 DC 03 <╖.♥ł)▄♥ä{·♥ł)▄♥
100000C0 3C B7 02 03 88 29 DC 03 52 69 63 68 89 29 DC 03 <╖☻♥ł)▄♥Richë)▄♥
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 00 PE..L☺♥.h=ŪV....
100000F0 00 00 00 00 E0 00 02 21 0B 01 0C 00 00 C0 00 00 ....ó.☻↑♂☺...└..
10000100 00 10 00 00 00 40 01 00 B0 06 02 00 00 50 01 00 .►...@☺.▒♠☻..P☺.
10000110 00 10 02 00 00 00 00 10 00 10 00 00 00 02 00 00 .►☻.....►.►...☻..
10000120 05 00 01 00 00 00 00 00 05 00 01 00 00 00 00 00 ♣.☺.....♣.☺.....
10000130 00 20 02 00 00 10 00 00 00 00 00 00 02 00 40 01 . ☻..►.......@.@☺
```

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-		



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):

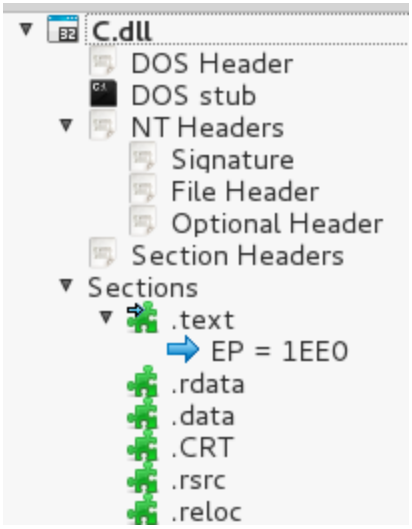
Offset	Name	Value	Meaning
CD68	Characterist...	0	
CD6C	TimeDateSt...	56EBCD67	
CD70	MajorVersion	0	
CD72	MinorVersion	0	
CD74	Name	21FA4	C.dll
CD78	Base	1	
CD7C	NumberOfF...	2	
CD80	NumberOfN...	2	
CD84	AddressOfF...	21F90	

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:

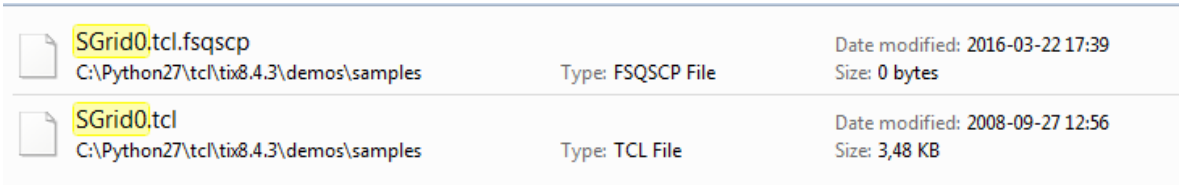
```
10004FB8  85C0            TEST EAX,EAX
10004FBA  0F84 70030000   JE 10005330
10004FC0  8B47 54         MOV EAX,DWORD PTR DS:[EDI+54]
10004FC3  6A 00           PUSH 0
10004FC5  68 00000008     PUSH 8000000
10004FCA  6A 03           PUSH 3
10004FCC  6A 00           PUSH 0
10004FCE  6A 01           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  E00 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:

```
1001277A  FFD3            CALL EBX
10012773  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 38310110   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 DDE9FEFF     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
100127BC  FF15 A4300110   CALL DWORD PTR DS:[100130A4]      USER32.DialogBoxParamA
100127C2  FF75 F8         PUSH DWORD PTR SS:[EBP-8]
100127C5  FF15 30310110   CALL DWORD PTR DS:[10013130]      gdiplus.GdiplusShutdown
100127CB  5F              POP EDI
100127CC  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                                  pLocaleId = 0030E5D8
0F90E079  .   PUSH [LOCAL.1]                            nLocaleId = 0x3
0F90E07C  .   CALL DWORD PTR DS:[<&USER32.GetKeyboardLa GetKeyboardLayoutList
0F90E082  .   MOV ECX,EAX
0F90E084  .   XOR EAX,EAX
0F90E086  .   TEST ECX,ECX
0F90E088  .v  JLE SHORT one.0F90E0A2
0F90E08A  .   MOV EDX,0x419                             locale_id = 1049 -> Russia
0F90E08F  .   NOP
0F90E090  >   CMP WORD PTR DS:[ESI+EAX*4],DX
0F90E094  .v  JE SHORT one.0F90E09D
0F90E096  .   INC EAX
0F90E097  .   CMP EAX,ECX
0F90E099  .^  JL SHORT one.0F90E090
0F90E09B  .v  JMP SHORT one.0F90E0A2
0F90E09D  >   MOV EDI,0x1
```

Excluded from the attack are also some predefined folders:

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

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

```
03AB2F4 70 61 6B 00 7F 55 EE 70 pak.ΔUεp
03AB2FC 68 6D 00 88 03 00 00 00 hm.ł♥...
03AB304 03 00 00 00 01 00 00 00 ♥...☺...
03AB30C 70 64 64 00 42 55 EE 70 pdd.BUεp
03AB314 75 70 00 88 03 00 00 00 up.ł♥...
03AB31C 03 00 00 00 01 00 00 00 ♥...☺...
03AB324 70 64 66 00 45 55 EE 70 pdf.EUεp
03AB32C 2C 76 00 88 03 00 00 00 ,v.ł♥...
03AB334 03 00 00 00 01 00 00 00 ♥...☺...
03AB33C 70 65 66 00 48 55 EE 70 pef.HUεp
03AB344 65 74 00 88 03 00 00 00 et.ł♥...
03AB34C 03 00 00 00 01 00 00 00 ♥...☺...
03AB354 70 65 6D 00 4B 55 EE 70 pem.KUεp
03AB35C 70 6C 00 88 03 00 00 00 pl.ł♥...
03AB364 03 00 00 00 01 00 00 00 ♥...☺...
03AB36C 70 66 78 00 4E 55 EE 70 pfx.NUεp
03AB374 78 6C 00 88 03 00 00 00 xl.ł♥...
03AB37C 03 00 00 00 01 00 00 00 ♥...☺...
03AB384 70 67 70 00 51 55 EE 70 pgp.QUεp
03AB38C 74 6D 00 88 03 00 00 00 tm.ł♥...
03AB394 03 00 00 00 01 00 00 00 ♥...☺...
03AB39C 70 6E 67 00 54 55 EE 70 png.TUεp
03AB3A4 22 58 00 88 03 00 00 00 "X.ł♥...
03AB3AC 03 00 00 00 01 00 00 00 ♥...☺...
03AB3B4 70 70 74 00 57 55 EE 70 ppt.WUεp
03AB3BC 00 8C 00 88 03 00 00 00 .î.ł♥...
03AB3C4 03 00 00 00 01 00 00 00 ♥...☺...
03AB3CC 70 73 64 00 5A 55 EE 70 psd.ZUεp
03AB3D4 00 8C 00 88 03 00 00 00 .î.ł♥...
03AB3DC 03 00 00 00 01 00 00 00 ♥...☺...
03AB3E4 70 73 6B 00 5D 55 EE 70 psk.]Uεp
03AB3EC 5B 58 00 88 03 00 00 00 [X.ł♥...
03AB3F4 03 00 00 00 01 00 00 00 ♥...☺...
03AB3FC 70 73 74 00 A0 55 EE 70 pst.áUεp
03AB404 00 8C 00 88 03 00 00 00 .î.ł♥...
03AB40C 03 00 00 00 01 00 00 00 ♥...☺...
03AB414 70 74 78 00 A3 55 EE 70 ptx.úUεp
```

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?

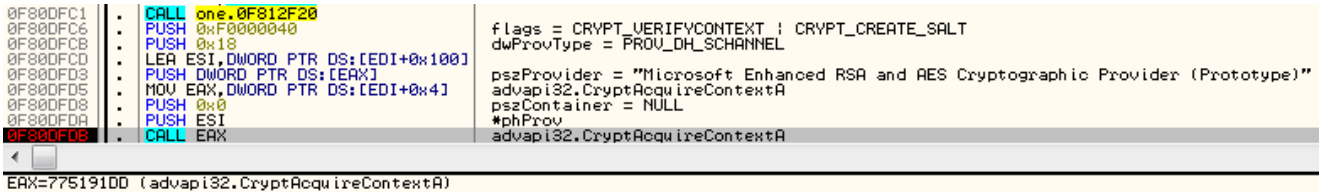
Maktub Locker uses Window Crypto API. But, as we concluded from the analysis, it uses only one key for all files (does not generate a random key per file). Let's see what technique it uses to obtain keys…

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

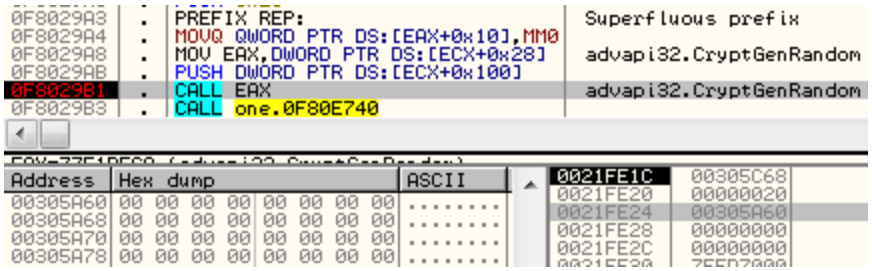
```
X25HE-J53ZU-QERDZ-ZNUJ3-SERJ6-J617E-UUASZ-AFG2G-83B08-2SHC1-AUYFZ-GJHF2-W7321-144TM
VKFKR-6TKRV-STG4B-CE5MZ-TAH4W-MP541-GD3SB-HE43J-ZF4TK-ZNZTG-R7ZBZ-AKM2U-T6TYN-53J7H
MU6J6-BTSJC-FQVQR-EH755-C1WCJ-7SNPT-MHFBS-Q638V-MASEB-R16HW-P84P2-7EEX8-KXAHB-D10F7
GF071-U37K3-GJ5Q5-WD0PD-2EG16-KMC5R-RPCBX-R8EV3-ZPXQV-TDVXM-SEEFX-XK23J-FCH4Z-RNBPN
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
```

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.

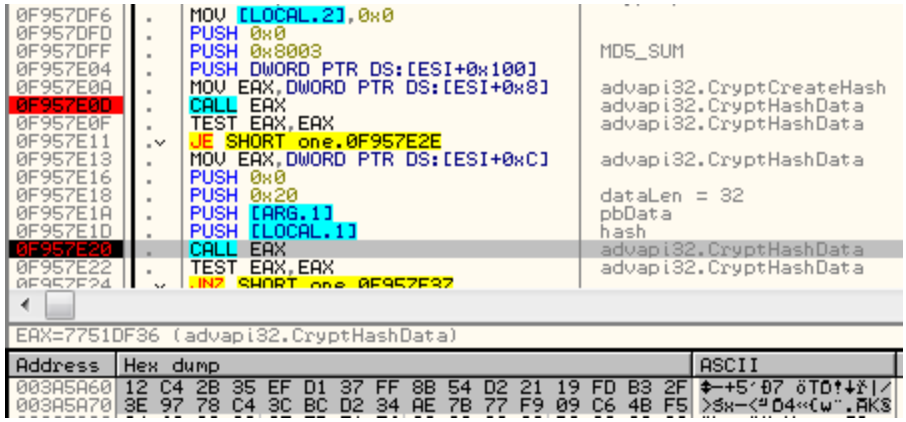
First it initialized two crypto contexts – both with the same settings, using provider type: PROV_DH_SCHANNEL



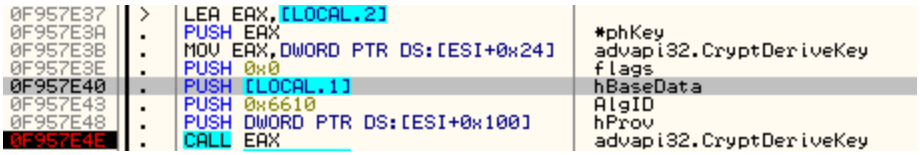
Gets 32 random bytes, using function CryptGenRandom



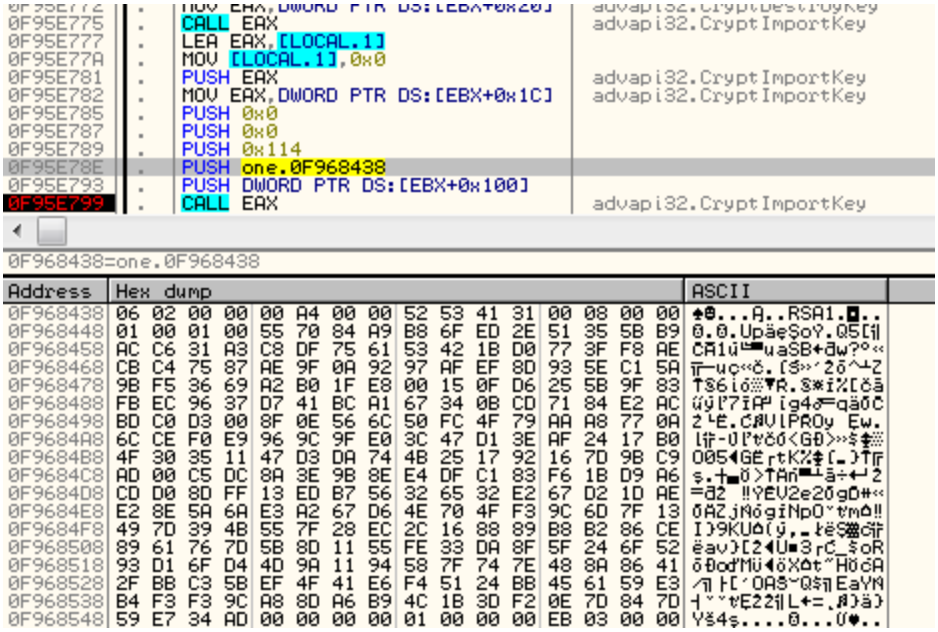
Creates MD5 sum of this random data (using: CryptCreateHash, CryptHashData)



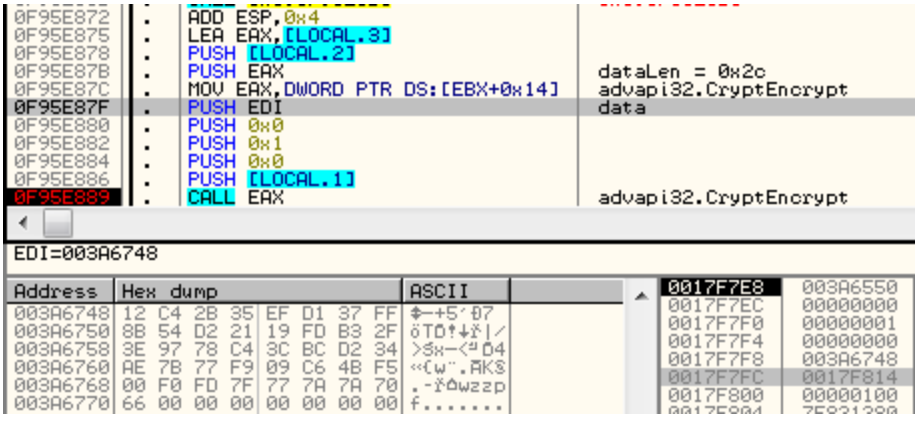
Then, using function CryptDeriveKey it converts the MD5 hash into a 256 bit AES key (AlgID = 0x6610 -> CALG_AES_256).



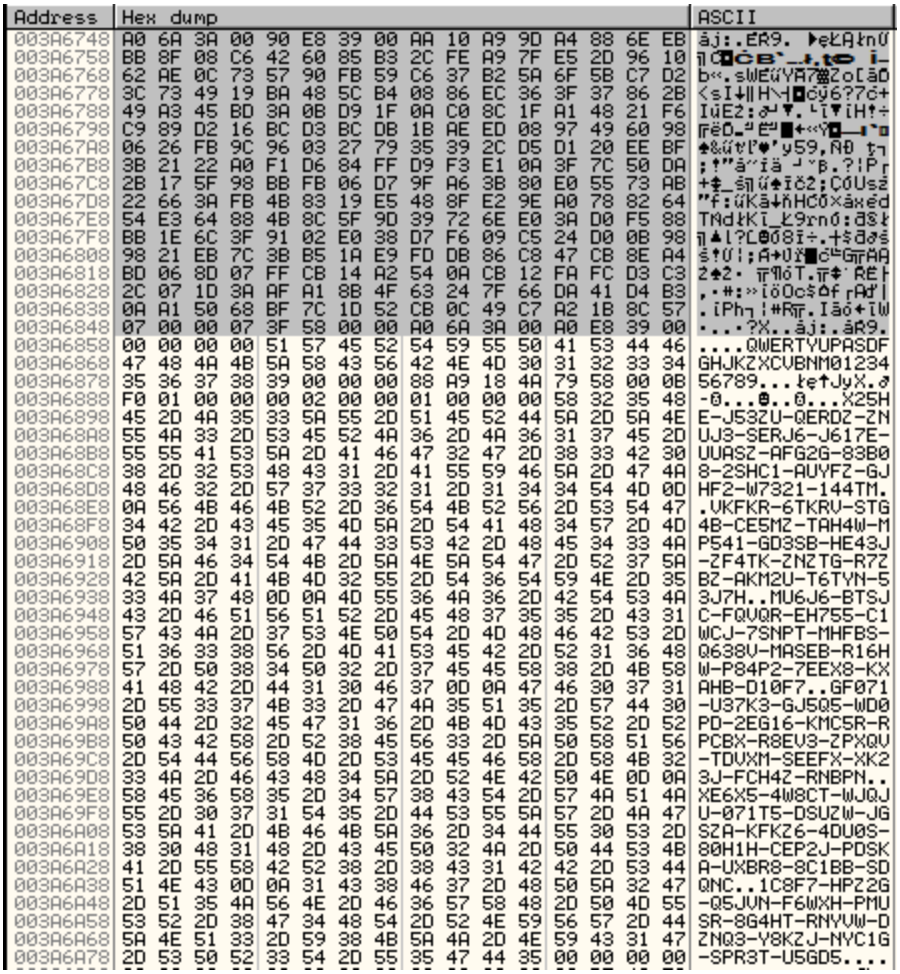
It also imports RSA public key (2048 bit). This key is hardcoded in the binary.



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:

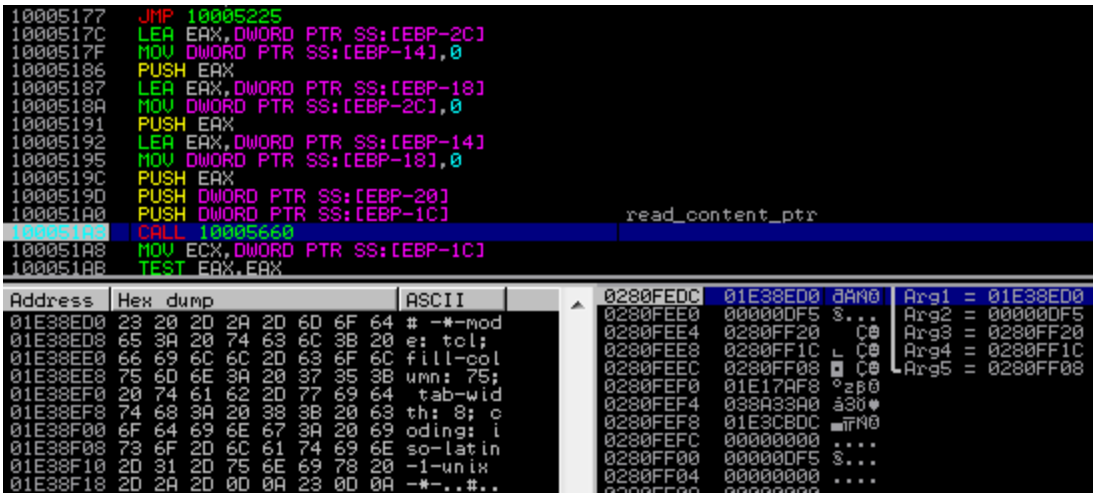


Address	Hex dump	ASCII
003A6748	A0 6A 3A 00 90 E8 39 00 AA 10 A9 9D A4 88 6E EB	áj:.ÉR9. ▶ęŁĄłnŰ
003A6758	BB 8F 08 C6 42 60 85 B3 2C FE A9 7F E5 2D 96 10	╗Ć◘ćB`…ł,t© ĺ–
003A6768	62 AE 0C 73 57 90 FB 59 C6 37 B2 5A 6F 5B C7 D2	b«.sWÉűYĂ7▒Zo[ăÔ
003A6778	3C 73 49 19 BA 48 5C B4 08 86 EC 36 3F 37 86 2B	<sI↓╢H\┤◘ćý6?7ć+
003A6788	49 A3 45 BD 3A 0B D9 1F 0A C0 8C 1F A1 48 21 F6	IúE2:♂┘▼.└Î▼ĩH!÷
003A6798	C9 89 D2 16 BC D3 BC DB 1B AE ED 08 97 49 60 98	╔ëÒ▬╜Ë╜ ■←«Ý◘—ı`¤
003A67A8	06 26 FB 9C 96 03 27 79 35 39 2C D5 D1 20 EE BF	♠&űŕP♥'y59,ĂĐ ţŗ
003A67B8	3B 21 22 A0 F1 D6 84 FF D9 F3 E1 0A 3F 7C 50 DA	;!"á˝íä ┘ˇß.?¦Pr
003A67C8	2B 17 5F 98 BB FB 06 D7 9F A6 3B 80 E0 55 73 AB	+↨_śŋŭ◆ĭčž;ĆóUsź
003A67D8	22 66 3A FB 4B 83 19 E5 48 8F E2 9E A0 78 82 64	"f:ŭKă↓ĺĥHĆő×áxéd
003A67E8	54 E3 64 88 4B 8C 5F 9D 39 72 6E E0 3A D0 F5 88	TMdłKĩ_Ł9rnó:đŝł
003A67F8	BB 1E 6C 3F 91 02 E0 38 D7 F6 09 C5 24 D0 0B 98	ŋ▲l?L@ó8ĭ÷.┼$đ♂ś
003A6808	98 21 EB 7C 3B B5 1A E9 FD DB 86 C8 47 CB 8E A4	ś↑Ú¦;Á→Ú▼■ć╚GŗĂĄ
003A6818	BD 06 8D 07 FF CB 14 A2 54 0A CB 12 FA FC D3 C3	ż♠2· ŗ¶ółT.ŗ╤♦ ŔĘ┼
003A6828	2C 07 1D 3A AF A1 8B 4F 63 24 7F 66 DA 41 D4 B3	,•#:»ĺöOc$ôf┌Aď|
003A6838	0A A1 50 68 BF 7C 1D 52 CB 0C 49 C7 A2 1B 8C 57	.¡Phŋ¦#R╦.Iăó+ĺW
003A6848	07 00 00 07 3F 58 00 00 A0 6A 3A 00 A0 E8 39 00	•...?X..áj:.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	GHJKZXCVBNM01234
003A6878	35 36 37 38 39 00 00 00 88 A9 18 4A 79 58 00 0B	56789...ęţJyX.♂
003A6888	F0 01 00 00 00 02 00 00 01 00 00 00 58 32 35 48	-☺...☻..☺...X25H
003A6898	45 2D 4A 35 33 5A 55 2D 51 45 52 44 5A 2D 5A 4E	E-J53ZU-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	UUASZ-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	.VKFKR-6TKRV-STG
003A68F8	34 42 2D 43 45 35 4D 5A 2D 54 41 48 34 57 2D 4D	4B-CE5MZ-TAH4W-M
003A6908	50 35 34 31 2D 47 44 33 53 42 2D 48 45 34 33 4A	P541-GD3SB-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 32 55 2D 54 36 54 59 4E 2D 35	BZ-AKM2U-T6TYN-5
003A6938	33 4A 37 48 0D 0A 4D 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-FQVQR-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	Q638V-MASEB-R16H
003A6978	57 2D 50 38 34 50 32 2D 37 45 45 58 38 2D 4B 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-KMC5R-R
003A69B8	50 43 42 58 2D 52 38 45 56 33 2D 5A 50 58 51 56	PCBX-R8EV3-ZPXQV
003A69C8	2D 54 44 56 58 4D 2D 53 45 45 46 58 2D 58 4B 32	-TDVXM-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-KFKZ6-4DU0S-
003A6A18	38 30 48 31 48 2D 43 45 50 32 4A 2D 50 44 53 4B	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	-Q5JVN-F6WXH-PMU
003A6A58	53 52 2D 38 47 34 48 54 2D 52 4E 59 56 57 2D 44	SR-8G4HT-RNYVW-D
003A6A68	5A 4E 51 33 2D 59 38 4B 5A 4A 2D 4E 59 43 31 47	ZNQ3-Y8KZJ-NYC1G
003A6A78	2D 53 50 52 33 54 2D 55 35 47 44 35 00 00 00 00	-SPR3T-U5GD5....

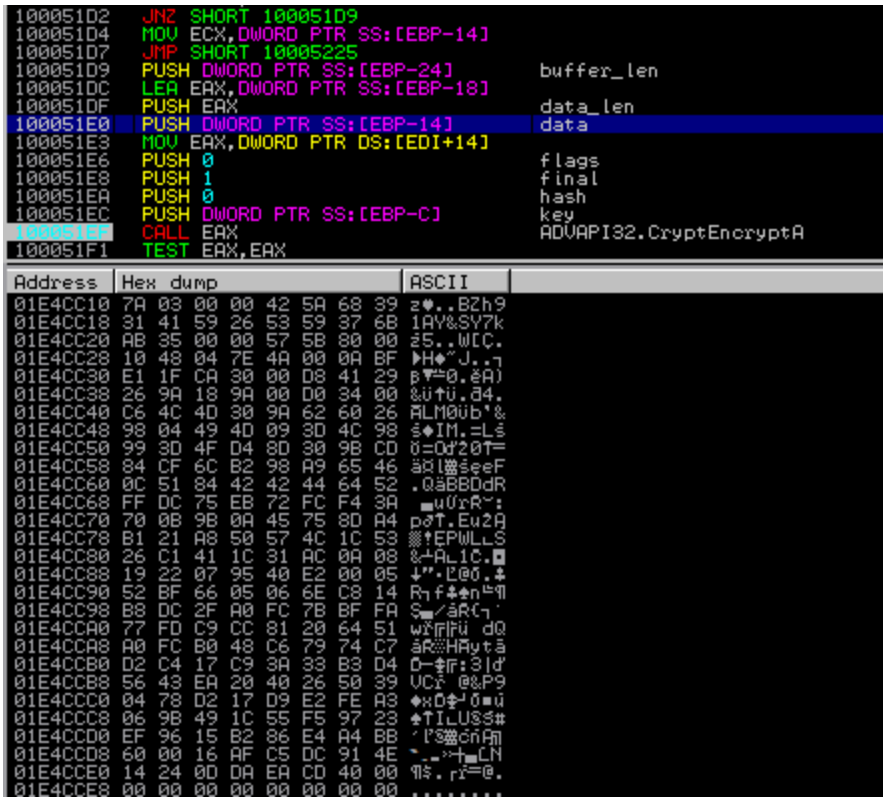
That’s why, when the user submit his/her individual ID, the attackers, having the appropriate private key, can decrypt the original data and easily recover the random AES key.

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)