

# Cerber ransomware: new, but mature

By hasherezade

Published: 2016-03-10 · Archived: 2026-04-05 20:32:16 UTC

Ransomware authors seem to love mythological creatures. We have seen [Chimera](#), now we will take a look at Cerber. Both are named after powerful beasts and both are prepared in a professional way. As [SenseCy](#) states ([source](#)), Cerber is sold to distributors on underground Russian forums.

This [malware](#) is often distributed via Exploit Kits (read more [here](#)).

**UPDATE:** [Checkpoint released a decryption tool working for some cases of Cerber](#)

## Analyzed samples

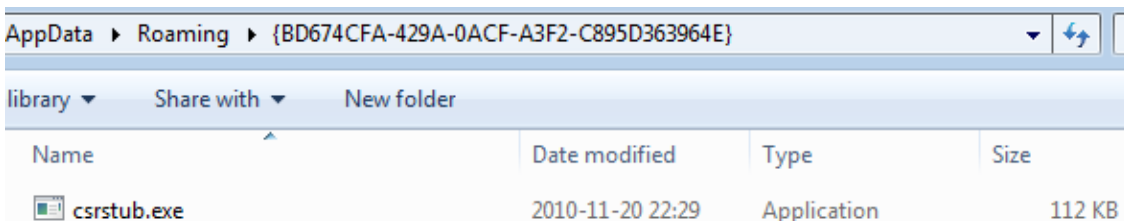
- [f5146a3bbe6c71e5a0ef2f04f955b1a1](#)
- [2f7059d7b1dda3080e391d99788fff18](#)
  - payload: [9a7f87c91bf7e602055a5503e80e2313](#) <- main focus of this analysis

## Behavioral analysis

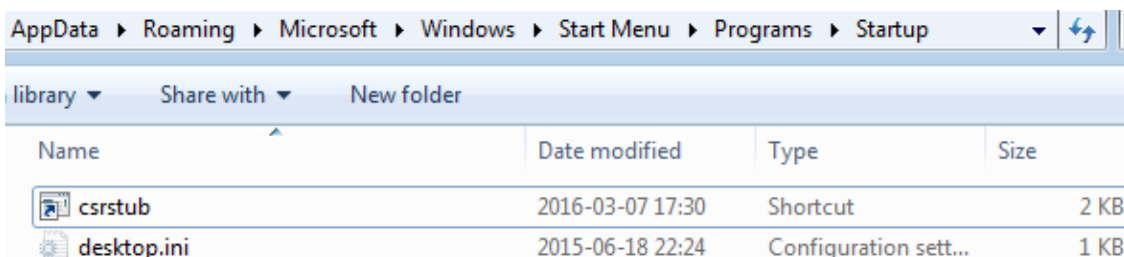
After being deployed it disappears and runs its dropped copy (renamed to [a random word].exe from the hidden folder created in %APPDATA%. Name of the folder is specific to a particular sample – in the analyzed one it is: **{BD674CFA-429A-0ACF-A3F2-C895D363964E}**).

Some observed file names: *csrsubst.exe*, *dinotify.exe*, *ndadmin.exe*, *setx.exe*, *rasdial.exe*, *RelPost.exe*, *ntkrnlpa.exe*

The dropped file has an edited creation timestamp.



It also creates a link to the dropped malware in: %APPDATA%/Microsoft/Windows/Start Menu/Programs/Startup:



Looking via Process Explorer we can see the dropped sample deploying new instances (it is used in order to divide the work of encrypting files).

ntkmlpa.exe	0.47	1 896 K	6 448 K	5868
ntkmlpa.exe	48.34	2 276 K	7 716 K	4828
ntkmlpa.exe	1.67	752 K	2 064 K	4704
ntkmlpa.exe	11.63	1 972 K	7 292 K	3768

## Registry keys

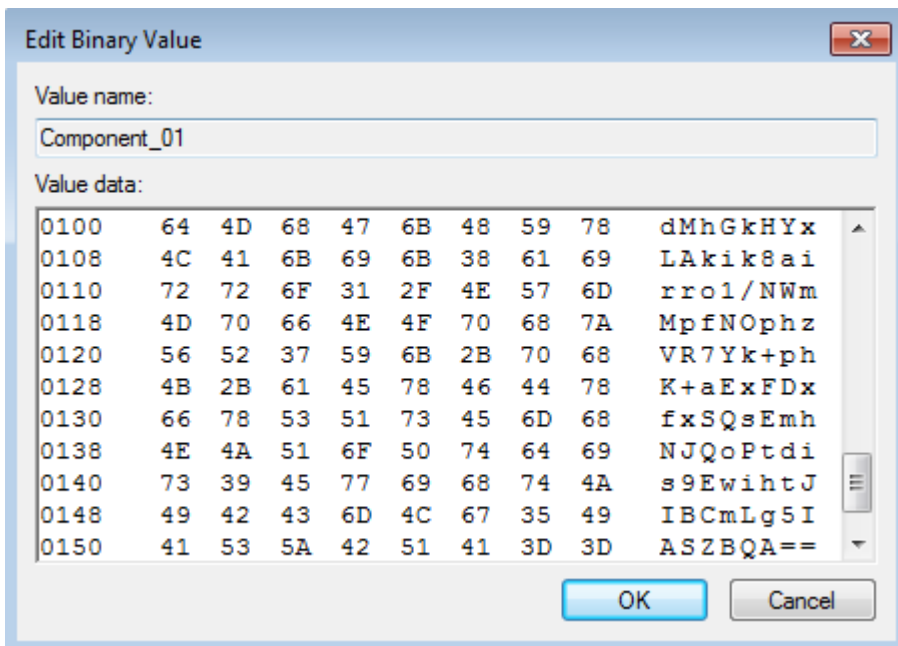
The malware makes changes in the Windows registry.

Two entries (Component\_00, Component\_01) are dropped in PrintersDefaults:

Name	Type	Data
(Default)	REG_SZ	(value not set)
Component_00	REG_BINARY	9a 04 01 01 82 b9 93 f9 40 02 00 00 01 00 00 00 db 5...
Component_01	REG_BINARY	42 45 74 4b 77 76 79 6b 6c 34 50 31 70 39 47 31 38 6...

Computer\HKEY\_CURRENT\_USER\Printers\Defaults\{67F80680-E78D-2436-5FCA-81CBA043DAE0}

Component\_01 contains some binary data in base64:



Registry keys for the persistence are added in various places, i.e:

HKEY\_USERS -> [current user's SID]:

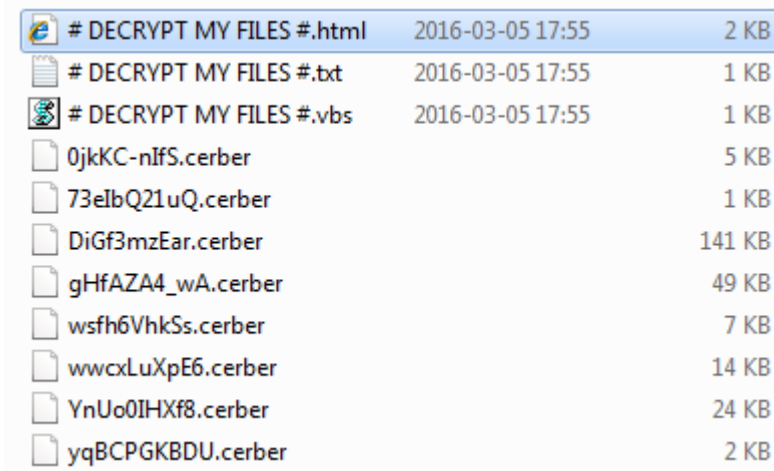
- "Software\Microsoft\Windows\CurrentVersion\Run"
- "Software\Microsoft\Windows\CurrentVersion\RunOnce"
- "Software\Microsoft\Windows\CurrentVersion\Policies\Explorer" -> "Run"
- "Software\Microsoft\Command Processor" -> "AutoRun"

However, when the encryption finishes successfully, the dropped sample is deleted.

## Encryption process

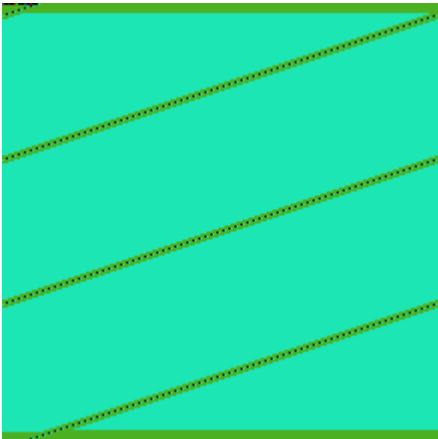
Cerber can encrypt files in offline mode – it means it doesn't need to fetch the key from the CnC server. Files that have been encrypted are fully renamed and appended with the extension typical for this ransomware: **.cerber**.

Pattern of the name: **[0-9a-zA-Z\_-]{10}.cerber**



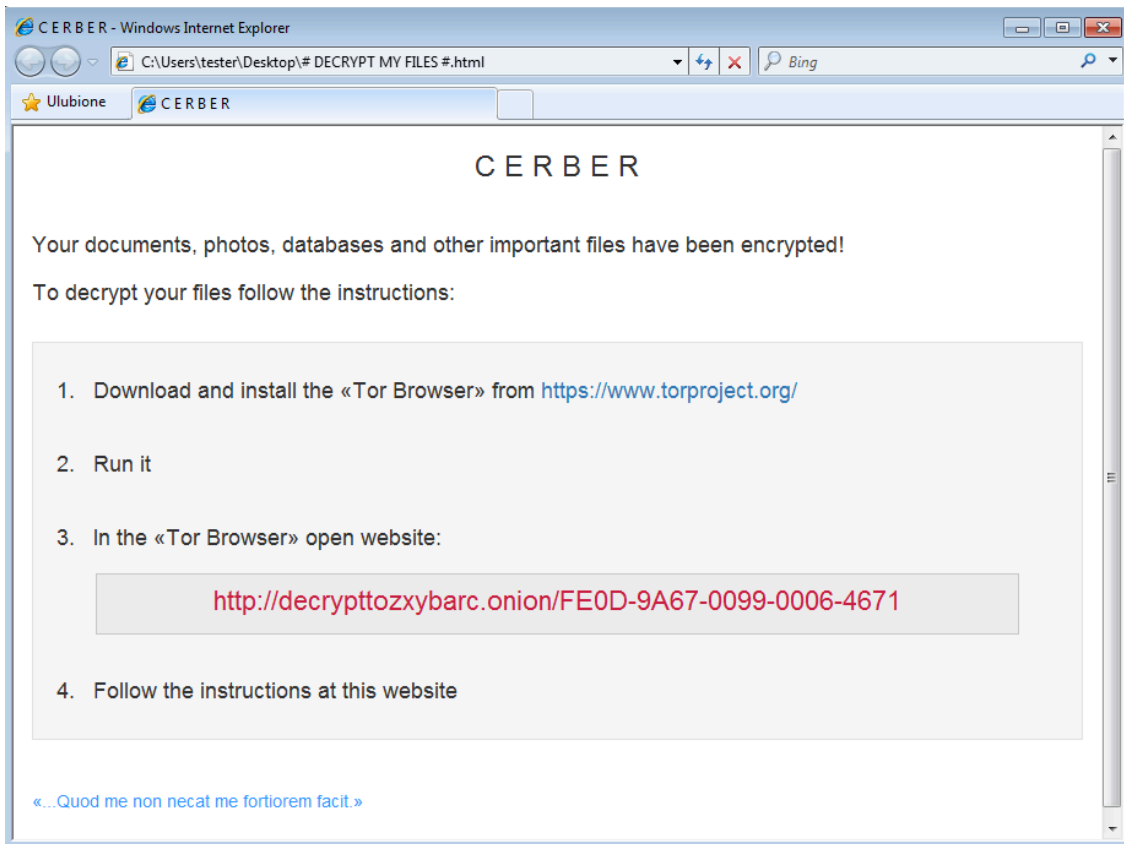
File Name	Modified	Size
# DECRYPT MY FILES #.html	2016-03-05 17:55	2 KB
# DECRYPT MY FILES #.txt	2016-03-05 17:55	1 KB
# DECRYPT MY FILES #.vbs	2016-03-05 17:55	1 KB
0jkkC-nlfS.cerber		5 KB
73eIbQ21uQ.cerber		1 KB
DiGf3mzEar.cerber		141 KB
gHfAZA4_wA.cerber		49 KB
wsfh6VhkSs.cerber		7 KB
wwcxLuXpE6.cerber		14 KB
YnUo0IHxf8.cerber		24 KB
yqBCPGKBDU.cerber		2 KB

The encrypted content has a high level of entropy and no patterns are visible. Below: visualization of bytes of *square.bmp* : left – original, right encrypted with Cerber:



Content of the encrypted file is different on every encryption – probably keys are dynamically generated. After encryption size of the file content is increased about 384 bytes\* – it may suggest, that the RSA encrypted AES key is appended to the file (\*depending on the file this value may vary a bit, probably because of various padding).

After executing it displays a ransom note in two forms: HTML and TXT. The note is available only in English. Example below:



C E R B E R

Your documents, photos, databases and other important files have been encrypted!

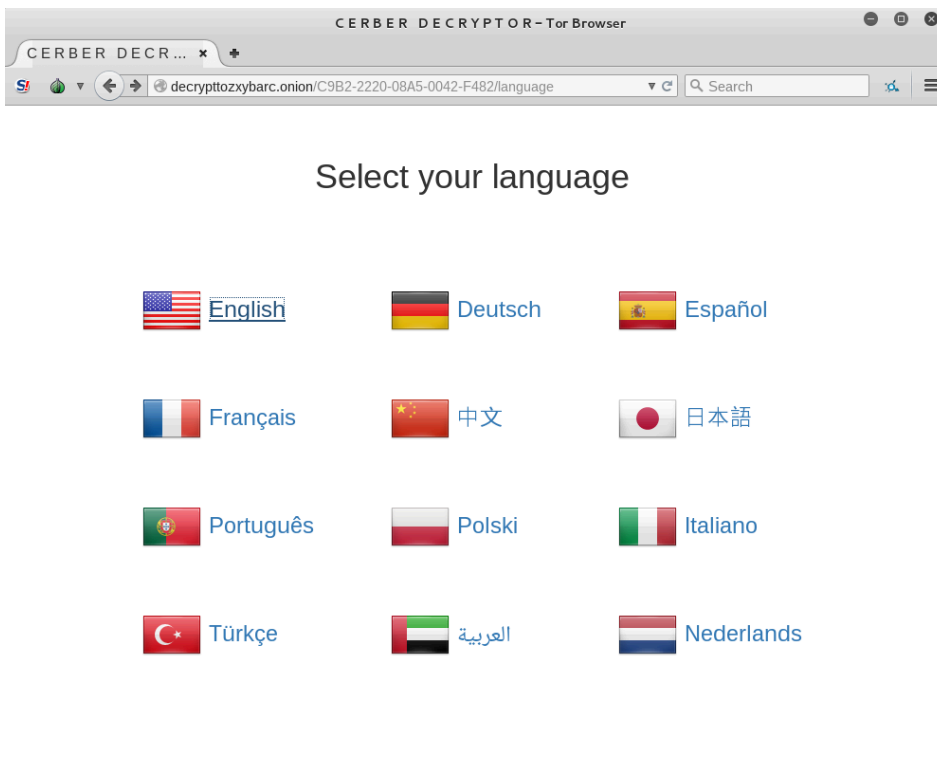
At the bottom of the ransom note attackers added a quote in Latin: «...Quod me non necat me fortioem facit.» (“What doesn’t kill me, makes me stronger”). We can only speculate what they wanted to convey – to share their own motto, or to console the victim of the attack?

It comes also with a VB macro that is supposed to speak up the message with the help of a local text-to-speech emulator:

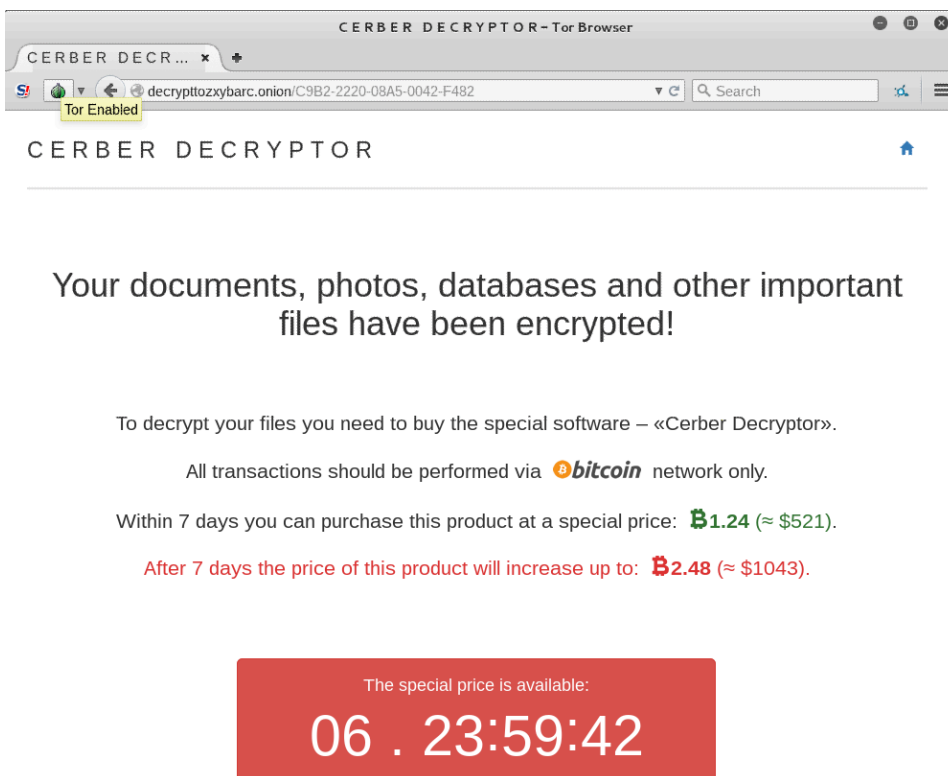
```
Set SAPI = CreateObject("SAPI.SpVoice") SAPI.Speak "Attention! Attention! Attention!" For i = 1 to 5
```

## Website for the victim

Each victim has a Web page that can be accessed via Tor. Although the ransom note is available only in English, the Tor website can be customized to several languages:



These pages contain further instructions to the victim and support for managing payments. The time to an increase in the ransom price is counted from the first access to this website.



To decrypt your files you need to buy the special software - <>.

## Network communication

Cerber can manage well without CnC and accomplish its task offline. However, if given opportunity, it can communicate with CnC in order to send statistics from encryption process.

First, it fetches geolocation info (in JSON format) of the local computer by querying a genuine service:

<http://ipinfo.io/json>

Then, we can observe sending UDP requests to a predefined range of IP addresses:

Destination	Protocol	Length	Info
87.98.128.0	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.1	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.2	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.3	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.4	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.5	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.6	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.7	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.8	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.9	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.10	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.11	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.12	UDP	68	Source port: 33919 Destination port: 6891
87.98.128.13	UDP	68	Source port: 33919 Destination port: 6891

## Inside

Cerber samples come packed by some [crypters/FUDs](#), so the code is not readable at first. Even when we unpack the core (i.e. [9a7f87c91bf7e602055a5503e80e2313](#)), only a few strings are readable. It is caused by the fact that the authors decided to encrypt the strings and decrypt them just before the usage. Example:

```

00346BED  |  | PUSH EBP
00346BEE  |  | MOV EBP,ESP
00346BF0  |  | PUSH ECX          cerber_p.0034B21E
00346BF1  |  | PUSH ECX          cerber_p.0034B21E
00346BF2  |  | PUSH EBX
00346BF3  |  | PUSH ESI
00346BF4  |  | PUSH EDI
00346BF5  |  | PUSH 0x1
00346BF7  |  | PUSH 0xECB03F2D
00346BFC  |  | PUSH 0x6
00346BFE  |  | PUSH cerber_p.003565AC
0034C083  |  | CALL cerber_p.0034A12A
0034C088  |  | PUSH EAX          decrypt_string
                                L"CERBER"

```

---

```

EAX=0155B0E0, (UNICODE "CERBER")

```

Address	Hex dump	ASCII
003565AC	75 D1 A7 01 0E E5 00 00 81 51 AB FB B9 1A DD 47	u020A0..u0201+TG
003565BC	E8 33 1D 19 BE E2 9E 2C 18 50 A5 EB 2A 6E 2F CB	R3+20x,↑PaU*n/r
003565CC	9F A2 61 0C 5D 41 F2 E5 72 CB 00 00 EC B2 1F 6B	00a.JA,n,r,..u007k
003565DC	74 B7 36 41 B1 31 D7 CF C7 44 E4 EF 9A 1D 04 E2	tE6A01i03Dn'u#0
003565EC	EA EB 75 CA 1D 4A 51 07 4D A8 CA 66 00 00 00 00	r0u2d' Q·ME2f....

The decrypting function takes the following parameters:

```
decrypt_string(char* input_buffer, DWORD input_lenght, DWORD key, BOOL is_unicode)
```

One of the few strings that hasn't been encrypted was a check against anti-malware vendors (one of them is Malwarebytes). The list of vendors is in JSON – this format have been used extensively by Cerber.

```

01192674 . MOV EAX, LHKs.Zj
01192677 . CMP DWORD PTR DS:[EAX], 0x0
0119267A . JNZ cerber_p.0119270F
01192680 . > XOR EBX, EBX
01192682 . PUSH EBX
01192683 . MOV ESI, cerber_p.011A5C20
01192688 . PUSH ESI
01192689 . CALL DWORD PTR DS:[<&KERNEL32.lstrLenA]
0119268F . PUSH ESI
    ASCII "[{"vendors":["VirusBlokAda","Malwarebytes"]}]"
    String = NULL
    lstrLenA
    
```

Another interesting unencrypted string was a log, showing the statistics on encryption (the feature used if the malware is deployed in the debug mode):

```

01196B94 . PUSH EAX
01196B95 . MOV EAX, ECX
01196B97 . XOR EDX, EDX
01196B99 . DIV EDI
01196B9B . PUSH EAX
01196B9C . PUSH DWORD PTR DS:[0x11F
01196BA2 . LEA EAX, [LOCAL_27]
01196BA5 . PUSH cerber_p.011A6508
01196BA9 . PUSH EAX
01196BAB . CALL DWORD PTR DS:[<&USE
01196BB1 . ADD ESP, 0x18
01196BB4 . PUSH 0x40
01196BB6 . PUSH cerber_p.011A65A0
01196BBB . LEA EAX, [LOCAL_27]
01196BBE . PUSH EAX
01196BBF . PUSH EBX
01196BC0 . CALL DWORD PTR DS:[<&USE
01196BC6 . PUSH 0x1
    <%d> = 75823C33 (1971469363.)
    cerber_p.<ModuleEntryPoint>
    <%d> = 75823C33 (1971469363.)
    <%d> = 0x0
    Format = "Keysize: %d, Encryption time: %d..Total files found: %d, Files crypted: %d"
    s = kernel32.BaseThreadInitThunk
    vsprintfW
    Style = MB_OK|MB_ICONASTERISK|MB_APPLMODAL
    Title = "done!"
    Text = "?????"
    hOwner = 7FFD6000
    MessageBoxW
    
```

### Configuration file

Cerber comes with an encrypted resource, stored as *RC Data*. It is decrypted by a dedicated function:

```

00341080 . PUSH EBX
0034108E . PUSH EDI
0034108F . PUSH [LOCAL_2]
00341092 . MOV ESI, EAX
00341094 . PUSH [LOCAL_3]
00341097 . CALL DWORD PTR DS:[<&KERNEL32.lstrLenA]
0034109D . PUSH EAX
0034109E . MOV EAX, ESI
003410A0 . CALL cerber_p.0034A244
003410A5 . LEA EAX, [LOCAL_5]
    cerber_p.0035D058
    String = "cerber"
    lstrLenA
    decrypt_resource
    
```

Address	Hex dump	ASCII
0035D058	71 7D B1 2B 15 CE CD 30 12 E4 23 E8 87 17 B2 7F	q]~+§ =0+n#Rc#
0035D068	96 03 7F C9 34 AD 5A 8E 95 C6 5F 76 BC 3F 8A 5C	l*0f4sZALR_v'?'
0035D078	04 32 2C 26 3F D2 F0 A3 D6 51 F8 98 62 0A 56 61	d'2,&?0-úi0°sb,Ua
0035D088	0D DE E8 AA 44 F8 68 6B 86 56 CB 06 D0 C8 AA 6E	.0R D°hk0Uf#0° n
0035D098	FD 04 F4 40 07 00 00 04 FF 04 00 0F 00 00 0F 00	°°°°°°°°°°°°°°°°

After decryption, it turns out to be a configuration in JSON format (you can see it full [here](#)):

```

0034108E . PUSH EDI
0034108F . PUSH [LOCAL_2]
00341092 . MOV ESI, EAX
00341094 . PUSH [LOCAL_3]
00341097 . CALL DWORD PTR DS:[<&KERNEL32.lstrLenA]
0034109D . PUSH EAX
0034109E . MOV EAX, ESI
003410A0 . CALL cerber_p.0034A244
003410A5 . LEA EAX, [LOCAL_5]
    String = "nosj"
    lstrLenA
    decrypt_resource
    
```

Address	Hex dump	ASCII
0155B2E8	78 22 61 6E 74 69 61 76 22 3A 31 2C 22 62 6C 61	{"antiav":1,"bla
0155B2F8	63 68 6C 69 73 74 22 3A 78 22 63 6F 75 6E 74 72	cklist":{"count
0155B308	69 65 73 22 3A 5B 22 61 60 22 2C 22 61 7A 22 2C	ies":["am","az"
0155B318	22 62 79 22 2C 22 67 65 22 2C 22 68 67 22 2C 22	"by","ge","kg"
0155B328	68 7A 22 2C 22 6D 64 22 2C 22 72 75 22 2C 22 74	kz","md","ru","t
0155B338	60 22 2C 22 74 6A 22 2C 22 75 61 22 2C 22 75 7A	m","tj","ua","uz
0155B348	22 5D 2C 22 66 69 6C 65 73 22 3A 5B 22 62 6F 6F	"],"files":["boo
0155B358	74 73 65 63 74 2E 62 61 68 22 2C 22 69 63 6F 6E	tsect.bak","icon
0155B368	63 61 63 68 65 2E 64 62 22 2C 22 74 68 75 6D 62	cache.db","thumb
0155B378	73 2E 64 62 22 2C 22 77 61 6C 6C 65 74 2E 64 61	s.db","wallet.da
0155B388	74 22 5D 2C 22 66 6F 6C 64 65 72 73 22 3A 5B 22	t"],"folders":["

Configuration is rich in options. Contains i.e:

- a [blacklist](#) used to exclude some countries, languages, file names and directories from the attack
- a list of [attacked extensions](#)
- [environment checks](#) that are enabled
- whether or not to deploy the sample in a [debug mode](#)
- [encryption settings and output extension](#)
- public [RSA key in base64 \(decoded\)](#).
- [files with ransom note](#) to be dropped
- [list of services](#) used to obtain geolocation
- [range of IPs](#) where to send statistics (*compare with IPs described in the section 'Network communication'*)
- [format of statistics](#) to be sent

**Distributors can customize many things** with the help of the config file. Changing the full look-and-feel of the malware – attacked extensions, ransom note and even extension of encrypted files – can make it appear like a new product. This flexibility made me wonder if the same package is not being distributed in a different campaign – not as a Cerber, but under some other name.

The distributor of the analyzed sample decided to exclude [several countries](#) form the attack (Armenia, Azerbaijan, Belarus, Georgia, Kyrgyzstan, Kazakhstan, Moldova, Russia, Turkmenistan, Tajikistan, Ukraine, Uzbekistan). It will also spare your default Windows directories, [Tor browser](#) and [Bitcoin wallet](#).

## Loading the key

The sample comes with a public RSA key [shipped in the configuration file](#) (described in the previous section).

Below – decrypting public key from Base64:

The screenshot shows a debugger window with assembly code and a hex dump. The assembly code includes instructions like PUSH, LEA, CALL, TEST, and JE. The CALL instruction at address 0080DAEE calls the function crypt32.CryptStringToBinaryA. Below the assembly, a hex dump shows the decrypted public key in ASCII format: "UUUUGQUFP00FR0EFNSULC02dL00FRRUF2a3R5NXFocUU5ZFI5MDc2RmU2cAowdU1Q".

Address	Hex dump	ASCII
016472E0	4C 53 30 74 4C 53 31 43 52 55 64 4A 54 69 42 51	LS0tLS1CRUdJTiB0
016472F0	56 55 4A 4D 53 55 4D 67 53 30 56 5A 4C 53 30 74	UUJMSUMgS0VZLS0t
01647300	4C 53 30 4B 54 55 6C 4A 51 6B 6C 71 51 55 35 43	LS0KTUlJQk lqQU5C

Key is imported using function [CryptImportPublicKeyInfo](#).

```

00404DD3 lea    eax, [ebp+pInfo]
00404DD6 push   eax                ; info
00404DD7 push   esi                ; 0
00404DD8 push   8000h             ; CRYPT_DECODE_ALLOC_FLAG
00404DDD push   [ebp+buf]         ; size = 0x126
00404DE0 mov    [ebp+pInfo], esi
00404DE3 push   edi                ; decoded_key_stage2
00404DE4 push   8                  ; X509_PUBLIC_KEY_INFO
00404DE6 push   10001h           ; X509_ASN_ENCODING | PKCS_7_ASN_ENCODING
00404DEB mov    [ebp+var_10], esi
00404DEE call   ds:CryptDecodeObjectEx
00404DF4 test   eax, eax
00404DF6 jz     short loc_404E16
    
```

```

00404DF8 lea    eax, [ebp+phKey]
00404DFB push   eax                ; phKey
00404DFC push   [ebp+pInfo]       ; pInfo
00404DFF push   1                  ; dwCertEncodingType
00404E01 push   hCryptProv        ; hCryptProv
00404E07 call   ds:CryptImportPublicKeyInfo
    
```

Configuration mentioned: "rsa\_key\_size": 576 – but it turns out to be a 2048 bit key (BLOB size – 276 bytes)

BLOBHEADER {PUBLICKEYBLOB, CUR\_BLOB\_VERSION, 0, CALG\_RSA\_KEYX}  
 RSAPUBKEY {"RSA1", len=2048, public\_exponent=65537}

Address	Hex	Dump	ASCII
003B0CB0	06 02 00 00 00 A4 00 00	52 53 41 31 00 08 00 00	+0...A..RSA1..
003B0CC0	01 00 01 00 9D 07 3D A8	59 DB 68 53 48 3A 39 02	0.0.L.=EYkSH:90
003B0CD0	3D 06 C7 C0 F6 14 2F 2D	AC 12 C0 16 61 45 87 91	=i3^+q/-C+^_aEcL
003B0CE0	1B 28 FF A1 9D B6 35 70	29 D8 38 04 D6 90 23 DA	+ (LAsp)e8+iE#r
003B0CF0	2C 33 A2 BC AA A7 B4 58	36 39 B3 1F CD 99 84 38	.3d^ zHX69!^o38
003B0D00	C6 6E 59 B9 E6 4F 7A A3	53 42 2C CD 69 E1 5A 40	Any\!S0zuSB,=igZ@
003B0D10	AC A0 E7 30 D2 6E 28 A4	48 8E 93 9E A9 93 12 4D	C3300n(AHA6xe6#M
003B0D20	F3 C1 55 43 E0 E9 9D 23	04 2A F9 C7 B4 10 B7 43	^+UC00k#**^@!EC
003B0D30	DE 3C 73 AC 44 F8 BE 6E	12 65 20 94 C3 FC D4 6F	0<sCD^zn#e o!Ad'o
003B0D40	11 29 59 FB 94 A4 CE 43	EE A0 35 3D 8B 6F FC FE	4)Y00#fCt35=0oR#
003B0D50	42 CF 92 09 25 3C B0 F0	C8 A7 0D B3 35 5C 6A 9E	B0(.%<@=-!2.15\jx
003B0D60	2D 38 8A FC 04 33 FD 50	B1 39 F6 EE F5 1C 10 19	-;0R3#P#9+tsL+>
003B0D70	DC 68 0C 9A 87 72 B6 19	64 32 DE C1 9E C6 E7 4D	k.ücrA+d20+XASM
003B0D80	3F 87 01 98 38 7F 0A 70	E0 B5 4E 07 81 D3 5A DB	?c0#80.p0AN.üEZ
003B0D90	D6 4A BF BF A0 43 AB C1	8A C1 E7 4F 4F 9C 12 84	iJhãC2+0+S00v#ã
003B0DA0	08 46 88 D8 66 31 E1 44	41 8F B1 03 40 CD 9A 4D	■Fëf16DAC#0=UM
003B0DB0	86 EC 0F E3 D2 E9 EB 15	FA 4E F7 51 27 13 6A A8	cY**AD00\$'N,Q'!!JE
003B0DC0	E6 72 4B BE		3rKz

### Installation

A file name of the dropped sample is created in a pretty interesting way. It is not fully random, but based on name of some file existing in the system, that is searched in the system using a random filter (format: "[random char]\*[random char]. exe", i.e "p\*h.exe"):

```

010F6189 .: | LEA EAX, [LOCAL.411]
010F618F .: | PUSH EAX
010F6190 .: | LEA EAX, [LOCAL.263]
010F6196 .: | PUSH EAX
010F6197 .: | CALL DWORD PTR DS:[K&KERNEL32.FindFirstFile]
010F619D .: | CMP EAX, -0x1
010F61A0 .: | JE SHORT cerber_p.010F6144
    
```

```

pFindFileData = 0038E718
FileName = "C:\Windows\system32\p*h.exe"
FindFirstFileW
    
```

The found file is compared with some built-in blacklist. When it pass the check, it is chosen as the new name of the dropped copy of the malware.

In order to prevent user from finding the malicious file by its creation timestamp it is changed to the timestamp of kernel32.dll existing on the local system.

After the successful installation, the initial malware sample terminates and deploys the dropped copy instead.

```

010F661B | CALL to CreateProcessW from cerber_p.010F6615
00376EB0 | ModuleFileName = "C:\\Users\\tester\\AppData\\Roaming\\{BD674CFA-429A-0ACF-A3F2-C895D363964E}\\csrstub.exe"
00000000 | CommandLine = NULL
00000000 | pProcessSecurity = NULL
00000000 | pThreadSecurity = NULL
00000000 | InheritHandles = FALSE
01000000 | CreationFlags = 1000000
00000000 | pEnvironment = NULL
00000000 | CurrentDir = NULL
0038EB68 | pStartupInfo = 0038EB68
0038EB58 | pProcessInfo = 0038EB58
    
```

## UAC Bypass

Cerber uses tricks to bypass Windows [User Account Control \(UAC\)](#) and deploy itself with elevated privileges. It is achieved by the following steps:

1. Search an executable in C:Windowssystem32, that can auto elevate it's privileges.
2. Search in it's import table a DLL that can be hijacked
3. Copy the DLL into %TEMP% folder and patch it – add a code in a new section and patch entry point in order to redirect execution there. It will be used in order to run the cerber sample with elevated privileges. It uses: `WinExec("[cerber_path] -eval 2524", SW_SHOWNORMAL)`
4. Inject the code into explorer.exe – it is responsible for executing the UAC bypass. Creates a new folder in C:Windowssystem32 and copy there both files – an EXE and the patched DLL – under original names, then it deploys the EXE causing DLL to load and execute the malicious code.
5. When the UAC bypass is executed successfully, it is signaled to the original cerber sample by setting a property `cerber_uac_status` – added to a Shell\_TrayWnd. Then, the original sample deletes dropped files and exits. Otherwise, it tries the same trick with different pair of EXE + DLL.

### See below how it looks in action:

First, it searches an application that can be used to elevate privileges. The check is based on the fields in application manifest:

```

true
    
```

Among it's imported DLLs it searches a candidate suitable to be hijacked. This DLL is copied into %TEMP% folder

```

00DE57C2 | . | LEA EAX, [LOCAL.131]
00DE57C8 | . | PUSH EAX
00DE57C9 | . | PUSH [ARG_2]
00DE57CC | . | CALL DWORD PTR DS:[<&KERNEL32.CopyFileW] CopyFileW
00DE57D2 | . | LEA EAX, [LOCAL.131]
00DE57D8 | . | PUSH EAX
00DE57D9 | . | CALL cerber_p.00DE53FC
    
```

Then, it creates a suspended process of *explorer.exe*, allocates memory in it's context and injects there own code. Details given below.

Injection into explorer is performed in several steps. First – malware is copying memory from the context of current process into the context of *explorer.exe*. Current image of Cerber sample is replicated into a memory allocated in explorer at 0x70000. Similarly, the page containing filled data is copied at offset 0x91000 in explorer.

```

000E4FD9 | . | LEA EAX, [LOCAL.3]
000E4FDC | . | PUSH EAX
000E4FDD | . | PUSH ESI
000E4FDE | . | PUSH [ARG.2]
000E4FE1 | . | MOV ESI, DWORD PTR DS:[&&KERNEL32.Wri
000E4FE7 | . | PUSH EBX
000E4FE8 | . | PUSH [ARG.1]
000E4FEB | . | CALL ESI
000E4FED | . | LEA EAX, [LOCAL.3]
000E4FF0 | . | PUSH EAX
000E4FF1 | . | PUSH [ARG.5]
000E4FF4 | . | PUSH [LOCAL.4]
000E4FF7 | . | PUSH [LOCAL.4]
000E4FFA | . | PUSH [ARG.1]
000E4FFD | . | CALL ESI
    
```

pBytesWritten = 0026DF38  
 BytesToWrite = 7580C1DE (1971372510.)  
 Buffer = cerber\_p.00DE0000  
 kernel32.WriteProcessMemory  
 Address = 0x70000  
 hProcess = 00000148  
 WriteProcessMemory

pBytesWritten = 0026DF38  
 BytesToWrite = 10BC (4284.)  
 Buffer = cerber\_p.00DFA170  
 Address = 0x91000  
 hProcess = 00000148  
 WriteProcessMemory

In order to run the injected code when the explorer.exe is resumed, malware performs patching of the carrier's Entry Point:

```

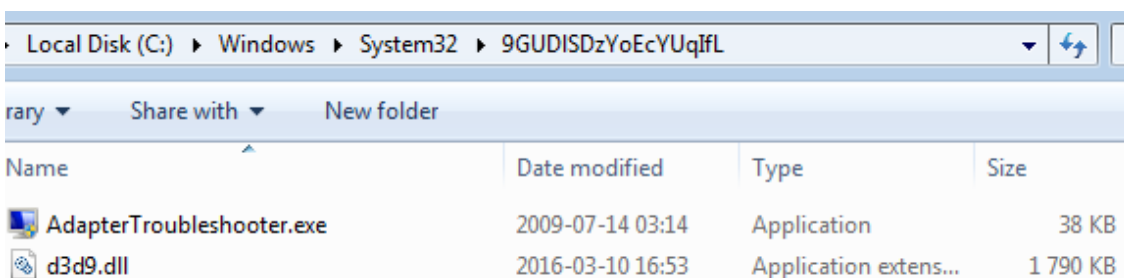
000E500B | . | PUSH EAX
000E500C | . | PUSH 0xE
000E500E | . | LEA EAX, [LOCAL.9]
000E5011 | . | PUSH EAX
000E5012 | . | PUSH EDI
000E5013 | . | PUSH [ARG.1]
000E5016 | . | MOV BYTE PTR SS:[EBP-0x24], 0xB8
000E501A | . | MOV WORD PTR SS:[EBP-0x1F], 0xB850
000E5020 | . | MOV DWORD PTR SS:[EBP-0x1D], EBX
000E5023 | . | MOV WORD PTR SS:[EBP-0x19], 0xD0FF
000E5029 | . | MOV BYTE PTR SS:[EBP-0x17], 0xC3
000E502D | . | CALL ESI
000E502F | . | PUSH 0x0
000E5031 | . | PUSH 0x0
000E5033 | . | PUSH [ARG.1]
000E5036 | . | CALL DWORD PTR DS:[&&KERNEL32.FlushInstruction
    
```

pBytesWritten = 0026DF20  
 BytesToWrite = E (14.)  
 Buffer = 0026DF20  
 Address = 0x630EFA  
 hProcess = 00000148  
 WriteProcessMemory  
 RegionSize = 0x0  
 RegionBase = NULL  
 hProcess = 00000148  
 FlushInstructionCache

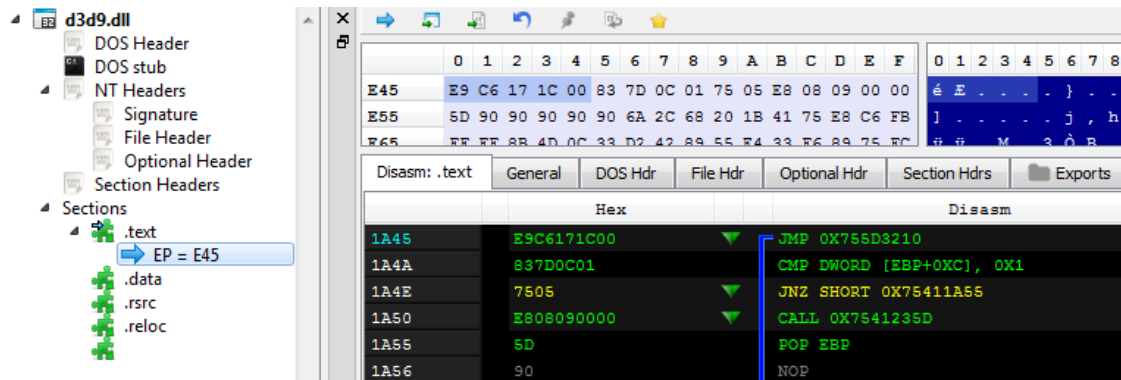
Stack SS:[0026DF50]=00DE0000 (cerber\_p.00DE0000)

Address	Hex dump	Disassembly	Comment
0026DF20	B8 00100900	MOV EAX, 0x91000	
0026DF25	50	PUSH EAX	
0026DF26	B8 E1550700	MOV EAX, 0x755E1	
0026DF2B	FFD0	CALL EAX	
0026DF2D	C3	RETN	

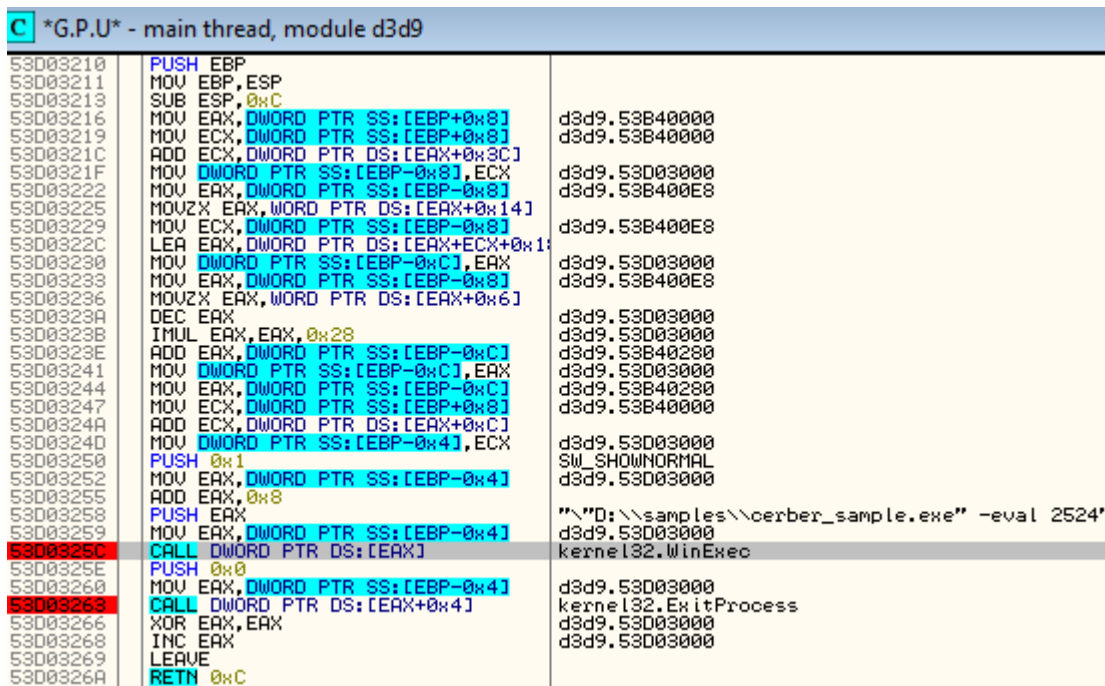
Now, Explorer's execution starts from the call to injected code. It is a function of Cerber sample – at RVA 0x55E1, called with a parameter 0x91000 – pointer to the memory page containing various dynamically loaded data, like function's handlers, paths of the files to be used, etc. From inside this code injected to explorer, the DLL patched for UAC bypass is copied under the original name – along with the appropriate EXE. The executable is deployed (using [ShellExecuteExW](#)) and along with it, the patched DLL also runs.



The *d3d9.dll* is used in order to run the Cerber sample with elevated privileges. Entry Point of the DLL is patched with a jump to the new section.



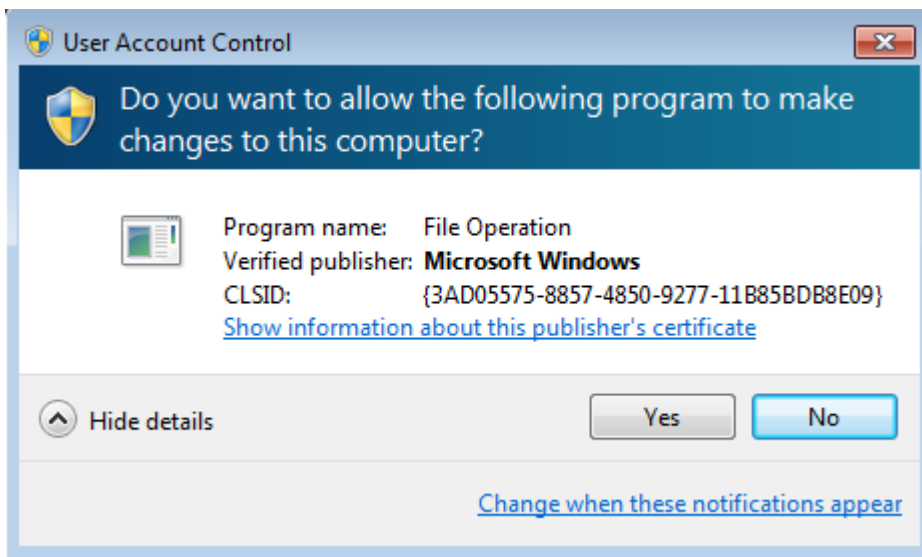
The new section contains the code that is supposed to execute the Cerber sample:



Successful UAC bypass is signaled by setting a property named “**cerber\_uac\_status**” in a found window of the class “*Shell\_TrayWnd*“. The initial Cerber sample waits for this status to change. If the timeout passed and it didn’t changed it makes a new attempt of UAC bypass – using a different pair (EXE+DLL). Otherwise it cleans up the environment and terminates. Infection proceeds from inside of the elevated sample.

00DE570F	PUSH EBX	Title = NULL
00DE5710	PUSH EAX	Class = 00021000 ???
00DE5711	CALL DWORD PTR DS:[<&USER32.FindWindow]	FindWindow
00DE5717	PUSH EBX	hData = NULL
00DE5718	MOV EBP, cerber_p.00DFB024	UNICODE "cerber_uac_status"
00DE571D	PUSH EBP	Property = "cerber_uac_status"
00DE571E	PUSH EAX	hWnd = 00021000
00DE571F	MOV DWORD PTR DS:[0xDFA170],EAX	
00DE5724	CALL DWORD PTR DS:[<&USER32.SetPropW]	SetPropW
00DE572A	MOV EDI, cerber_p.00DE55E1	
00DE572F	PUSH EDI	
00DE5730	CALL cerber_p.00DEDC6A	
00DE5735	MOV ESI, EAX	image_base
00DE5737	CALL cerber_p.00DEDCD4	
00DE573C	MOV DWORD PTR SS:[ESP],0x10BC	
00DE5743	PUSH cerber_p.00DFA170	Arg5 = 00DFA170
00DE5748	SUB EDI, ESI	cerber_p.00DE0000
00DE574A	PUSH EDI	Arg4 = 000055E1
00DE574B	PUSH EAX	Arg3 = 00021000
00DE574C	PUSH ESI	Arg2 = 00DE0000
00DE574D	PUSH cerber_p.00DFB230	"C:\\Windows\\explorer.exe"
00DE5752	CALL cerber_p.00DE5052	inject_into_explorer
00DE5757	ADD ESP,0x18	
00DE575A	TEST EAX, EAX	
00DE575C	JBE SHORT cerber_p.00DE5786	
00DE575E	XOR ESI, ESI	cerber_p.00DE0000
00DE5760	PUSH EBP	Property = "cerber_uac_status"
00DE5761	PUSH DWORD PTR DS:[0xDFA170]	hWnd = 0001004E (class='Shell_TrayWnd')
00DE5767	CALL DWORD PTR DS:[<&USER32.GetPropW]	GetPropW
00DE576D	TEST EAX, EAX	
00DE576F	JNZ SHORT cerber_p.00DE5784	is_uac_bypassed?
00DE5771	PUSH 0x3E8	Timeout = 1000. ms
00DE5776	CALL DWORD PTR DS:[<&KERNEL32.Sleep]	Sleep
00DE577C	INC ESI	cerber_p.00DE0000
00DE577D	CMP ESI,0x3C	
00DE5780	JBE SHORT cerber_p.00DE5760	
00DE5782	JMP SHORT cerber_p.00DE5786	
00DE5784	MOV BL,0x1	success
00DE5786	POP EDI	cerber_p.00DFB230

In case if UAC level is set to default (or lower), Cerber can bypass it silently. However, in case if it is set to the highest, the following alert pops up:



It keeps reappearing till the user click “Yes”:

## Conclusion

Cerber is a pretty powerful ransomware written with attention to details. This analysis highlighted only some of the elements. It has rich customization options and various tricks to make analysis harder. Although this product appeared recently, for sure its authors are not new in the field of malware development. We can expect it will be gaining popularity and may carry some new tricks in the future.

## Appendix

- <http://www.bleepingcomputer.com/forums/t/606583/cerber-ransomware-decrypt-my-files-vbs-txt-html/> – thread about Cerber on forum Bleeping Computer
- <http://www.bleepingcomputer.com/news/security/the-cerber-ransomware-not-only-encrypts-your-data-but-also-speaks-to-you/>

## **About the author**

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

---

Source: <https://blog.malwarebytes.com/threat-analysis/2016/03/cerber-ransomware-new-but-mature/>