

Cerber ransomware: new, but mature

blog.malwarebytes.com/threat-analysis/2016/03/cerber-ransomware-new-but-mature/

hasherezade

March 11, 2016



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

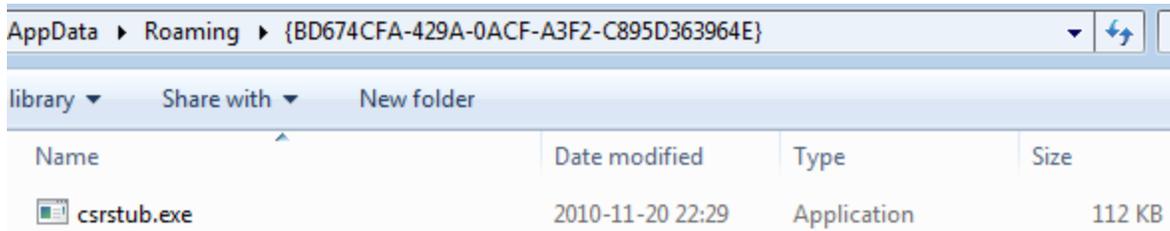
Behavioral analysis

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}.

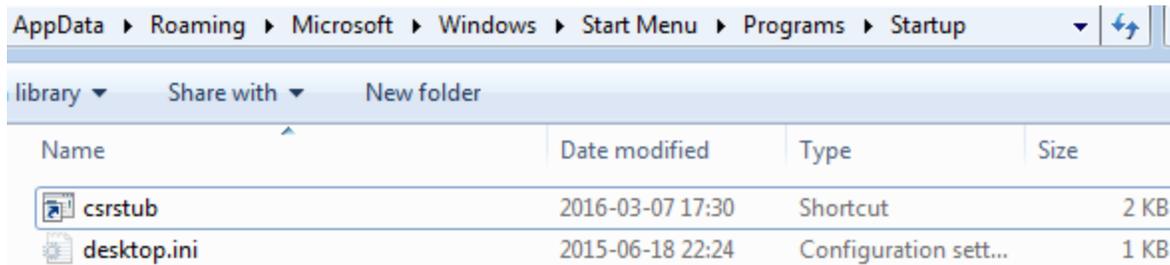
Some observed file names: *csrstub.exe*, *dinotify.exe*, *ndadmin.exe*, *setx.exe*, *rasdial.exe*, *RelPost.exe*, *ntkrnlpa.exe*

The dropped file has an edited creation timestamp.



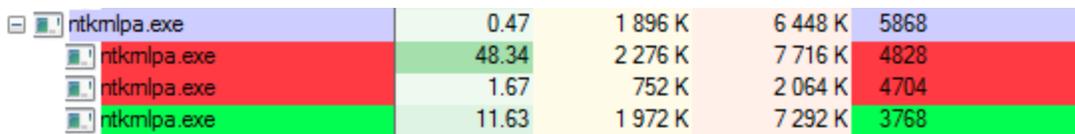
Name	Date modified	Type	Size
csrstub.exe	2010-11-20 22:29	Application	112 KB

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



Name	Date modified	Type	Size
csrstub	2016-03-07 17:30	Shortcut	2 KB
desktop.ini	2015-06-18 22:24	Configuration sett...	1 KB

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).

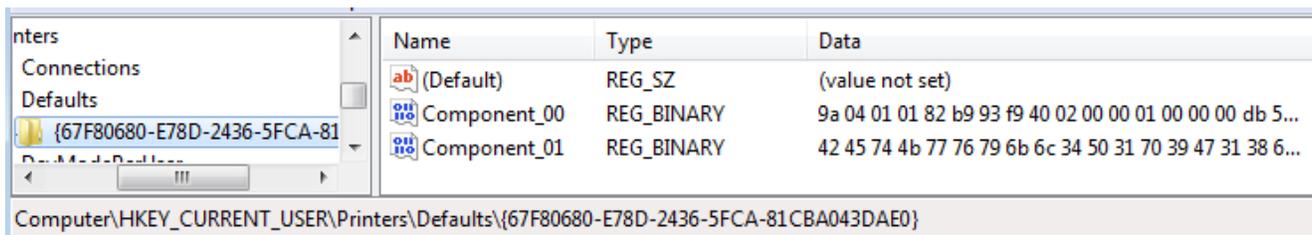


ntkrnlpa.exe	0.47	1 896 K	6 448 K	5868
ntkrnlpa.exe	48.34	2 276 K	7 716 K	4828
ntkrnlpa.exe	1.67	752 K	2 064 K	4704
ntkrnlpa.exe	11.63	1 972 K	7 292 K	3768

Registry keys

The malware makes changes in the Windows registry.

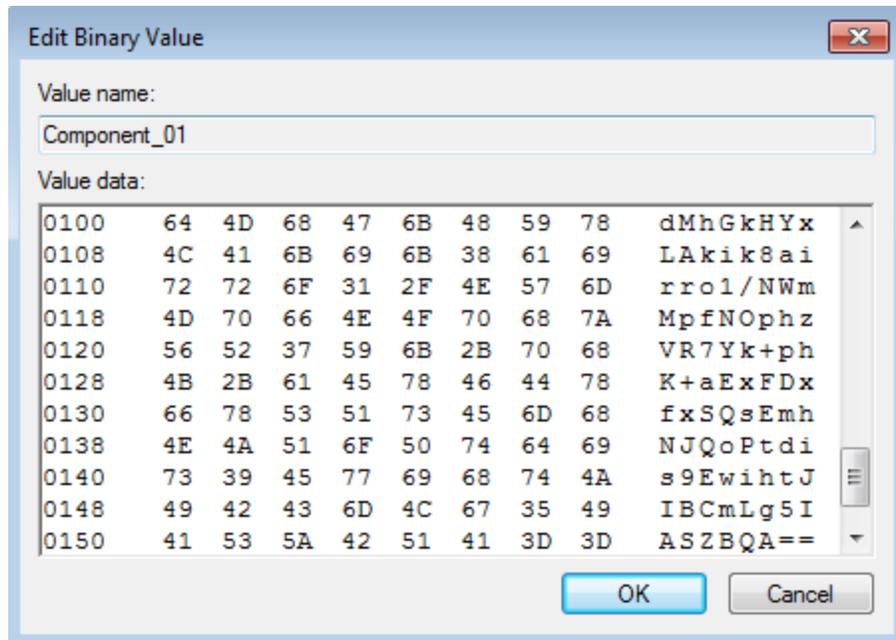
Two entries (Component_00, Component_01) are dropped in Printers\Defaults:



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}

Compont_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"

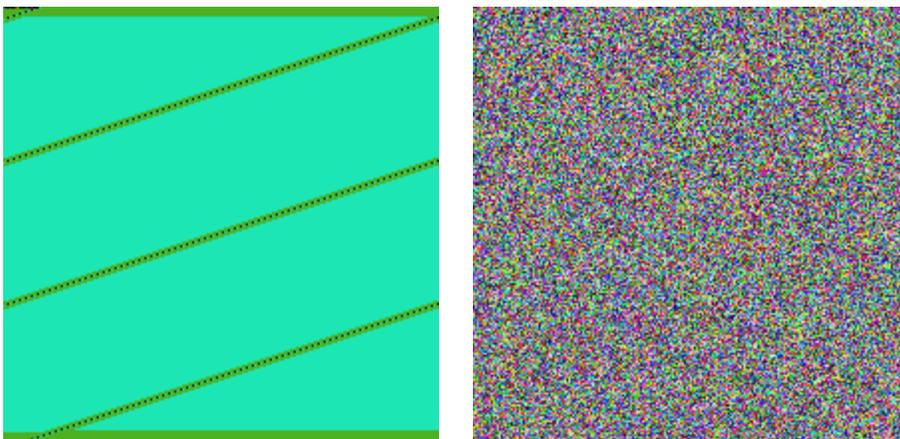
However, when the encryption finishes successfully, the dropped sample is deleted.

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**

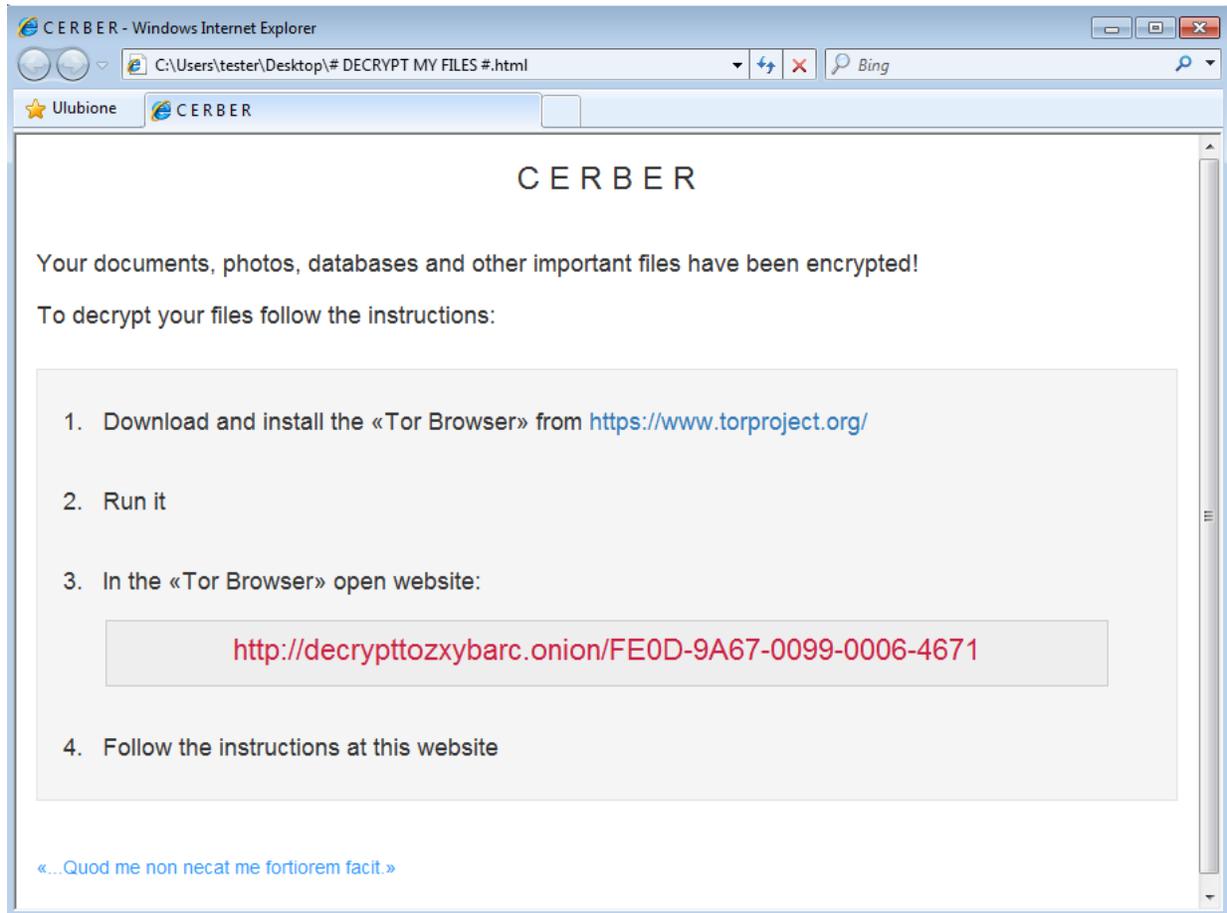
# DECRYPT MY FILES #.html	2016-03-05 17:55	2 KB
# DECRYPT MY FILES #.txt	2016-03-05 17:55	1 KB
# DECRYPT MY FILES #.vbs	2016-03-05 17:55	1 KB
0jkKC-nIfS.cerber		5 KB
73eIbQ21uQ.cerber		1 KB
DiGf3mzEar.cerber		141 KB
gHfAZA4_wA.cerber		49 KB
wsfh6VhkSs.cerber		7 KB
wwcxLuXpE6.cerber		14 KB
YnUo0IHxf8.cerber		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!

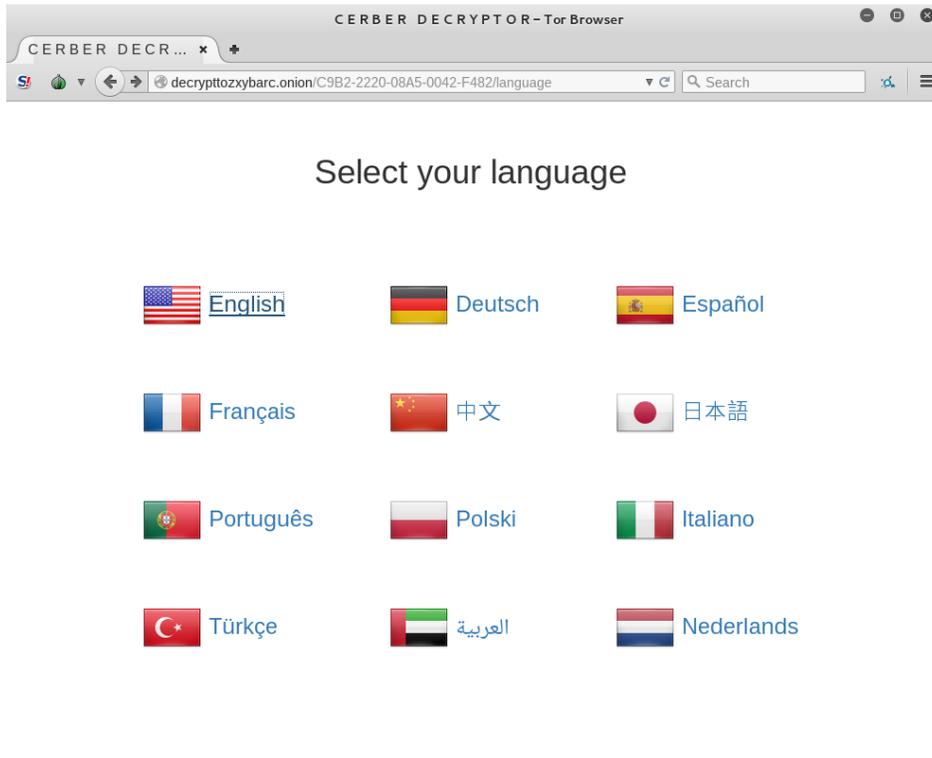
At the bottom of the ransom note attackers added a quote in Latin: «...Quod me non neecat me fortiorem facit.» (“What doesn’t kill me, makes me stronger”). We can only speculate what they wanted to convey – to share their own motto, or to console the victim of the attack?

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:

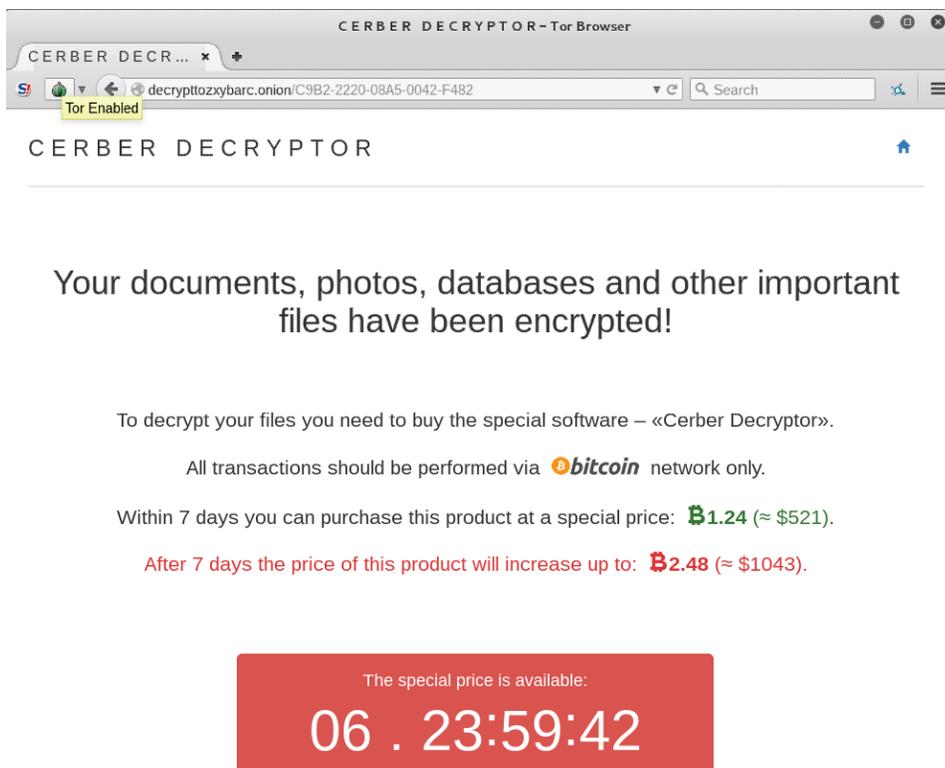
```
Set SAPI = CreateObject("SAPI.SpVoice")
SAPI.Speak "Attention! Attention! Attention!"
For i = 1 to 5
SAPI.Speak "Your documents, photos, databases and other important files have been
encrypted!"
Next
```

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 - <<Cerber Decryptor>>.

Network communication

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, [ARG.2]
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 from 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:[0x11A6508]
01196BA2 . . LEA EAX, [LOCAL.27]
01196BA5 . . PUSH cerber_p.011A6508
01196BAA . . PUSH EAX
01196BAB . . CALL DWORD PTR DS:[<&USER32.Format]
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:[<&USER32.MessageBoxW]
01196BC6 . . PUSH 0x1

```

<%d> = 75823C38 (1971469363.)
cerber_p.<ModuleEntryPoint>
<%d> = 75823C38 (1971469363.)
<%d> = 0x0
Format = "Keysize: %d, Encryption time: %d..Total files found: %d, Files crypted: %d"
s = kernel32.BaseThreadInitThunk
wsprintfW
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:

```

0034108D . . 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

0034A244=cerber_p.0034A244

Address	Hex dump	ASCII
0035D058	71 7D B1 28 15 CE CD 30 12 E4 23 E8 87 17 B2 7F	q)~+3i=0#n#Rc#0
0035D068	96 03 7F C9 34 AD 5A 8E 95 C6 5F 76 BC 3F 8A 5C	l*0r4sZALu'?'0\
0035D078	D4 32 2C 26 3F D2 F0 A3 D6 51 F8 98 62 0A 56 61	d'2,&'0-4i0°sb.Ua
0035D088	0D DE E8 AA 44 F8 68 68 86 56 CB 06 D0 C8 AA 6E	.0R D°hk6U#d° n
0035D098	FD 04 F4 40 07 D0 00 04 FF 04 00 0F 00 00 0F 00	4#010C000400 100

005FF700 003410A5
005FF704 00000006
005FF708 0035D058
005FF7DC 0000109D
005FF7E0 0155B2E8
005FF7E4 00000006

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] [String = "nosj"
0034109D | . | PUSH EAX | lstrlenA]
0034109E | . | MOV EAX,ESI
003410A0 | . | CALL cerber_p.0034A244 | decrypt_resource
003410A5 | . | LEA EAX,[LOCAL.5]
003410A8 | . | PUSH EAX
003410AA | . | CUIE EBX

```

Address	Hex dump	ASCII
0155B2E8	7B 22 61 6E 74 69 61 76 22 3A 31 2C 22 62 6C 61	{ "antiau":1,"bla
0155B2F8	63 6B 6C 69 73 74 22 3A 7B 22 63 6F 75 6E 74 72	cklist":{"count
0155B308	69 65 73 22 3A 5B 22 61 6D 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	6B 7A 22 2C 22 6D 64 22 2C 22 72 75 22 2C 22 74	kz","md","ru","t
0155B338	6D 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 6B 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 from 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:

```

008DDAE7 . . . PUSH 0x1
008DDAE9 . . . PUSH ESI
008DDAEA . . . LEA EAX, DWORD PTR SS:[EBP-0x78]
008DDAED . . . PUSH EAX
008DDAEE . . . CALL DWORD PTR DS:[&CRYPT32.CryptStringToBinaryA crypt32.CryptStringToBinaryA
008DDAF4 . . . TEST EAX, EAX
008DDAF6 . . . JZ SHORT cerber_p.008DDB16
008DDAF8 . . . MOV EAX, DWORD PTR SS:[EBP-0x2C]
008DDAFB . . . MOV EAX, DWORD PTR SS:[EBP-0x2C] EAX

```

EAX=0030FBEB, (ASCII "UUUGUUFFP00FROEFNSUICQ2dL00FRRUF2a3R5NXFocUU5ZFI5MDc2RmV2cAowdU10")

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	UUJMSUMgS0UZLS0t
01647300	4C 53 30 4B 54 55 6C 4A 51 6B 6C 71 51 55 35 43	LS0KTU1JQk lq0U5C

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
00404DD9 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
00404DFE 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 90 07 30 A8 59 0B 68 53 48 3A 39 02	0.0.jL.=EYkSH:90
003B0CD0	3D 06 C7 C0 F6 14 2F 2D AC 12 C0 16 61 45 87 91	=ia+q/-C+L.aEcL
003B0CE0	1B 28 FF A1 9D B6 35 70 29 08 38 04 D6 90 23 DA	+([kASp)é8+iE#r
003B0CF0	2C 33 A2 BC AA A7 B4 58 36 39 B3 1F CD 99 84 38	,30" z1X691V=0a8
003B0D00	C6 6E 59 B9 E6 4F 7A A3 53 42 2C CD 69 E1 5A 40	AnVl]50zuSB,=ipZ@
003B0D10	AC A0 E7 30 D2 6E 28 A4 48 8E 93 9E A9 93 12 40	Ca\$00n(QHA0xe0#M
003B0D20	F3 C1 55 43 E0 E9 9D 23 04 2A F9 C7 B4 10 B7 43	"+UC0Uk#**"sh EC
003B0D30	DE 3C 73 AC 44 F8 BE 6E 12 65 20 94 C3 FC D4 6F	0<sCD0zn#e 0Ad'o
003B0D40	11 29 59 FB 94 A4 CE 43 EE A0 35 3D 8B 6F FC FE	4)Y00AftCtã5=0oA#
003B0D50	42 CF 92 09 25 3C B0 F0 C8 A7 8D B3 35 5C 6A 9E	B0f.%@-Ez.15\jx
003B0D60	2D 3B 8A FC 04 33 FD 50 B1 39 F6 EE F5 1C 10 19	-;0R+3tF#9+t&L!+
003B0D70	DC 6B 0C 9A 87 72 B6 19 64 32 DE C1 9E C6 E7 4D	#k.UcrA+d20+XR\$M
003B0D80	3F 87 01 98 38 7F 0A 70 E0 B5 4E 07 81 D3 5A D8	?c0\$00.p0AN-uE2█
003B0D90	D6 4A BF BF A0 43 AB C1 8A C1 E7 4F 4F 9C 12 84	iJh7ãCz+0+00v#ã
003B0DA0	08 46 88 D8 66 31 E1 44 41 8F B1 03 40 CD 9A 4D	█Fêf1B0AC#0=UM
003B0DB0	8E EC 0F E3 D2 E9 EB 15 FA 4E F7 51 27 13 6A A8	0y#AD00'S'N,0'!!JE
003B0DC0	E6 72 4B BE	SrKz

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_4111]
010F618F | . | PUSH EAX
010F6190 | . | LEA EAX, [LOCAL_2631]
010F6196 | . | PUSH EAX
010F6197 | . | CALL DWORD PTR DS:[<&KERNEL32.FindFirstFileW]
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
00976EB0 | . | ModuleFileName = "C:\Users\tester\AppData\Roaming\{CBD674CFA-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:\Windows\system32, 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:\Windows\system32 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:

```
<autoElevate>true</autoElevate>
<requestedExecutionLevel level="requireAdministrator"/>
```

00DE5880	MOV BYTE PTR SS:[EBP-0x19],BL	
00DE5883	PUSH 0x2	
00DE5885	PUSH EBX	
00DE5886	PUSH DWORD PTR SS:[EBP+0x8]	
00DE5889	CALL DWORD PTR DS:[&KERNEL32.LoadLibraryExW]	Flags = LOAD_LIBRARY_AS_DATAFILE hFile = NULL FileName = "C:\\Windows\\system32\\AdapterTroubleshooter.exe"
00DE588F	MOV EDI, EAX	LoadLibraryExW
00DE58C1	MOV DWORD PTR SS:[EBP-0x2C],EDI	
00DE58C4	CMP EDI, EBX	
00DE58C6	JE cerber_p.00DE5A1F	
00DE58CC	PUSH 0x18	
00DE58CE	PUSH 0x1	
00DE58D0	PUSH EDI	
00DE58D1	CALL DWORD PTR DS:[&KERNEL32.FindResourceW]	ResourceType = 0x18 ResourceName = 0x1 hModule = 00390001 FindResourceW
00DE58D7	MOV ESI, EAX	
00DE58D9	CMP ESI, EBX	
00DE58DB	JE cerber_p.00DE5A16	
00DE58E1	PUSH ESI	
00DE58E2	PUSH EDI	
00DE58E3	CALL DWORD PTR DS:[&KERNEL32.SizeofResource]	hResource = 003926E8 hModule = 00390001 SizeofResource
00DE58E9	CMP EAX, EBX	
00DE58EB	JE cerber_p.00DE5A16	
00DE58F1	PUSH ESI	
00DE58F2	PUSH EDI	
00DE58F3	CALL DWORD PTR DS:[&KERNEL32.LoadResource]	hResource = 003926E8 hModule = 00390001 LoadResource
00DE58F9	CMP EAX, EBX	
00DE58FB	JE cerber_p.00DE5A16	
00DE5901	PUSH EAX	
00DE5902	CALL DWORD PTR DS:[&KERNEL32.LockResource]	hResource = 00817FE8 LockResource
00DE5908	MOV DWORD PTR SS:[EBP-0x28],EAX	
00DE590B	CMP EAX, EBX	
00DE590D	JE cerber_p.00DE5A16	
00DE5913	MOV DWORD PTR SS:[EBP-0x20],EBX	
00DE5916	MOV DWORD PTR SS:[EBP-0x4],EBX	
00DE5919	PUSH 0x76A2BBBE	
00DE591E	PUSH 0xB	
00DE5920	PUSH cerber_p.00DF6260	
00DE5925	CALL cerber_p.00DEA216	
00DE592A	ADD ESP, 0xC	
00DE592D	PUSH EAX	
00DE592E	PUSH DWORD PTR SS:[EBP-0x28]	
00DE5931	CALL DWORD PTR DS:[&SHLWAPI.StrStrIA]	Pattern = "autoElevate" String = "<?xml version="1.0" encoding="UTF-8" standalone="1"?"><autoElevate>true</autoElevate><requestedExecutionLevel level="requireAdministrator"/></manifest>" StrStrIA
00DE5937	MOV DWORD PTR SS:[EBP-0x20],EAX	
00DE593A	OR DWORD PTR SS:[EBP-0x4],0xFFFFFFFF	
00DE593E	JMP SHORT cerber_p.00DE594D	

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]	NewFileName = "C:\\Users\\tester\\AppData\\Local\\Temp\\1443.tmp" ExistingFileName = "C:\\Windows\\system32\\d3d9.dll" 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 coping 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.

```

00DE4FD9 | . LEA EAX, [LOCAL.3]
00DE4FDC | . PUSH EAX
00DE4FDD | . PUSH ESI
00DE4FDE | . PUSH [ARG.2]
00DE4FE1 | . MOV ESI, DWORD PTR DS:[<&KERNEL32.Wr
00DE4FE7 | . PUSH EBX
00DE4FE8 | . PUSH [ARG.1]
00DE4FEB | . CALL ESI
00DE4FED | . LEA EAX, [LOCAL.3]
00DE4FF0 | . PUSH EAX
00DE4FF1 | . PUSH [ARG.5]
00DE4FF4 | . PUSH [ARG.4]
00DE4FF7 | . PUSH [LOCAL.4]
00DE4FFA | . PUSH [ARG.1]
00DE4FFD | . 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:

```

00DE500B | . PUSH EAX
00DE500C | . PUSH 0xE
00DE500E | . LEA EAX, [LOCAL.9]
00DE5011 | . PUSH EAX
00DE5012 | . PUSH EDI
00DE5013 | . PUSH [ARG.1]
00DE5016 | . MOV BYTE PTR SS:[EBP-0x24], 0xB8
00DE501A | . MOV WORD PTR SS:[EBP-0x1F], 0xB850
00DE5020 | . MOV DWORD PTR SS:[EBP-0x10], EBX
00DE5023 | . MOV WORD PTR SS:[EBP-0x19], 0xD0FF
00DE5029 | . MOV BYTE PTR SS:[EBP-0x17], 0xC3
00DE502D | . CALL ESI
00DE502F | . PUSH 0x0
00DE5031 | . PUSH 0x0
00DE5033 | . PUSH [ARG.1]
00DE5036 | . CALL DWORD PTR DS:[<&KERNEL32.FlushInstruction
00DE503C | . MOV BYTE PTR SS:[EBP-0x24], 0xB8

```

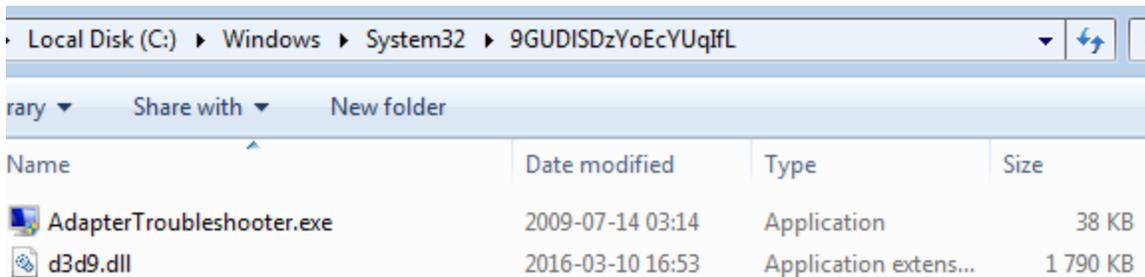
pBytesWritten = 0026DF20
BytesToWrite = E (14.)
Buffer = 0026DF20
Address = 0x630EFA
hProcess = 00000148
WriteProcessMemory
RegionSize = 0x0
RegionBase = NULL
hProcess = 00000148
FlushInstructionCache

Stack SS:[0026DF50]=00DE0000 (cerber_p.00DE0000)

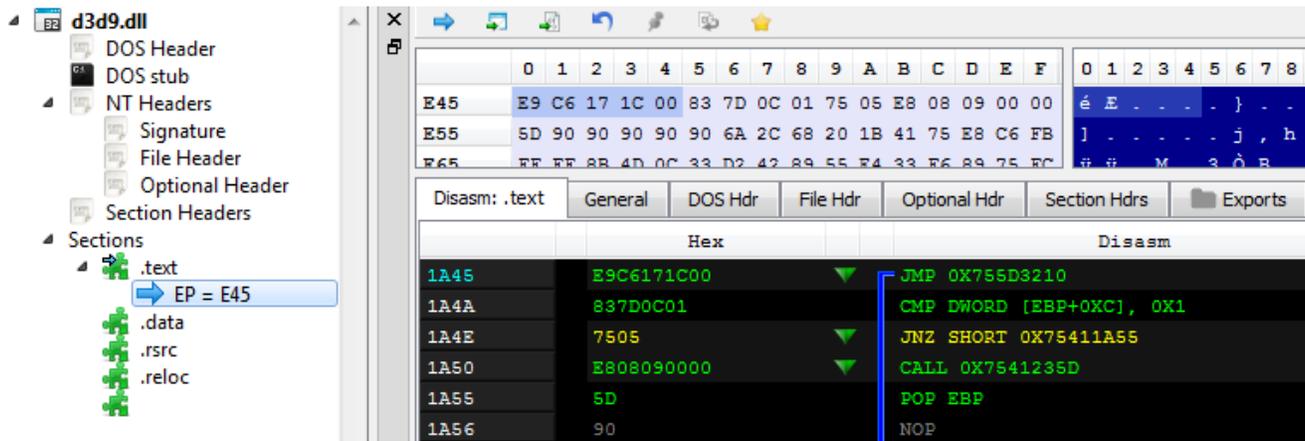
Address	Hex dump	Disassembly	Comment
0026DF20	B8 00100900	MOV EAX, 0x91000	
0026DF25	50	PUSH EAX	
0026DF26	B8 E1550700	MOV EAX, 0x755E1	
0026DF2B	FFD0	CALL EAX	
0026DF2D	C3	RETN	

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.



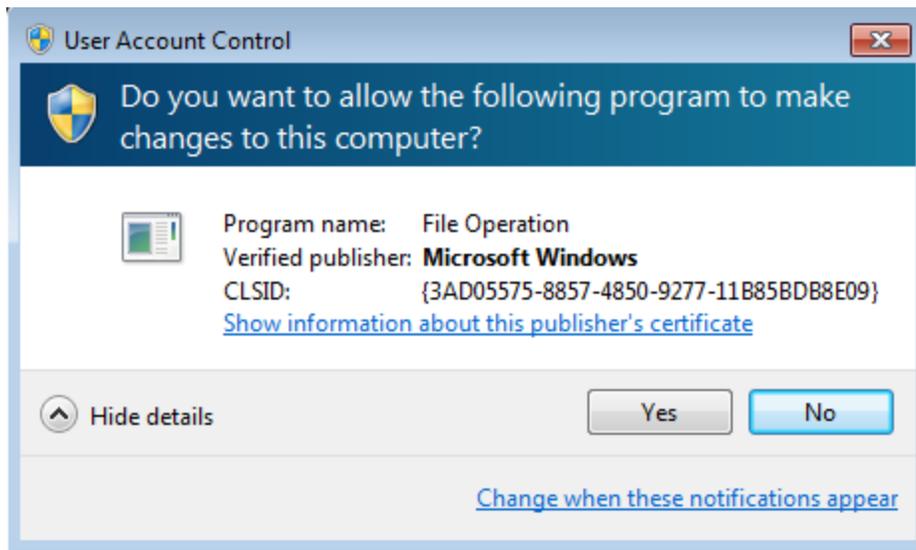
The new section contains the code that is supposed to execute the Cerber sample:



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.SetProp@]	SetProp@
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	JE 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.GetProp@]	GetProp@
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	JB 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
