

Analysis: Malware Win32/Rimecud.B

Infections of Win32/Rimecud.B were first spotted in the wild in the second half of 2010, but customers are still calling us due to difficulties in removing it even in the presence of anti-virus software. So we decided to analyze it and on the way also describe some interesting anti-debugging techniques that are used by it. We also analyze the malware's behavior once a system is infected.

Sample

File: ctfmon.exe
MD5: f5f4ec6d780715d713b7e085fd24447c
SHA1: f4507f91806aef7bdbbab1047b5ce4d5d6033e6c
File Type: MS Windows Portable Executable file

Malware Analysis

- 1) Before starting the analysis, open the malware in PEiD to see if the malware was packed using any known available packers. PEid indicates that the malware is packed using UPX packer (fig.1). For further analysis the malware is unpacked using the Ultimate Packer for executable.

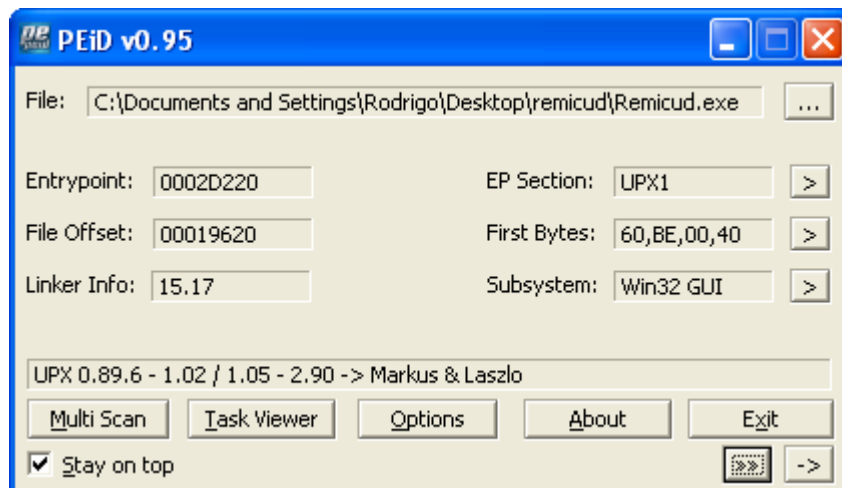


Figure 1: PEid output for malware sample.

- 2) Once the unpacked malware executable is opened in a debugger, we will see that the malware does a lot of calls to Windows API "CopyFileA", trying to copy some random files to random location and this is done multiple times in a very big loop. This is junk code used to probably frustrate the reverse engineer (Fig.2).

```

00404550 *$ 55      PUSH EBP
00404551 * 8BEC     MOV EBP,ESP
00404553 * 8BEC     SUB ESP,1C
00404556 > 8130 A1F34000  < CMP DWORD PTR DS:[40F3A11,1BF400
00404560 * 0F83 BC090000  JNB retnicud_.00404F22
00404566 * E8 95CAFFFF  CALL retnicud_.00401000
0040456B * B6 FA      PUSH DH,0FA
0040456F * E8 E2140000  CALL <JMP.&KERNEL32.GetModuleHandleA> [GetModuleHandleA
00404574 * 8105 A1F34000  ADD DWORD PTR DS:[40F3A11,2D0
0040457E * E8 8DCAFFFF  CALL retnicud_.00401010
00404583 * 66 801C30  LEA EBX,MORD PTR DS:[EDX*4ED1]
00404588 * A1 E8D41000  MOV EAX,DWORD PTR DS:[418DEC]
0040458C * 80E1 6C     AND CL,6C
0040458F * E8 A4140000  CALL <JMP.&KERNEL32.GetCurrentProcessId> [GetCurrentProcessId
00404594 * 8105 A1F34000  ADD DWORD PTR DS:[40F3A11,21E
0040459E * 68 B8F74100  PUSH retnicud_.0041F7B8
004045A3 * 6A 00      PUSH 0
004045A5 * E8 86CAFFFF  CALL retnicud_.00401030
004045AA * A1 E66E4100  MOV EAX,DWORD PTR DS:[416EE6]
004045AF * 8105 A1F34000  ADD DWORD PTR DS:[40F3A11,283
004045B3 * E8 2CAFFFF  CALL retnicud_.00401050
004045BE * 66 41     INC CX
004045C0 * 8105 A1F34000  ADD DWORD PTR DS:[40F3A11,25F
004045C4 * 52        PUSH EDX
004045C8 * E8 C8CAFFFF  CALL retnicud_.00401090
004045D0 * A1 E39D4100  MOV EAX,DWORD PTR DS:[419DE3]
004045D5 * 0FB6F6    MOVZX ESI,DH
004045D8 * E8 4F140000  CALL <JMP.&KERNEL32.GetACP> [GetACP
004045DD * 8105 A1F34000  ADD DWORD PTR DS:[40F3A11,2A9
004045E7 * E8 4CAFFFF  CALL retnicud_.004010D0
004045EC * A1 FF3C4100  MOV EAX,DWORD PTR DS:[413CFF]
004045F1 * D0D2     RCL DL,1
004045F3 * 847D D0    TEST BYTE PTR SS:[EBP-30],BH
004045F6 * JRE SHORT retnicud_.0040460E
004045FA * 6A 00      PUSH 0
004045FF * 68 0BDC4100  PUSH retnicud_.0041DC08
004045FA * 68 78134100  PUSH retnicud_.00411378
00404604 * E8 1D140000  CALL <JMP.&KERNEL32.CopyFileA> [CopyFileA
00404609 * E8 36140000  CALL <JMP.&KERNEL32.GetCurrentThreadId> [GetCurrentThreadId
0040460E > A1 89974100  MOV EAX,DWORD PTR DS:[413789]
00404613 * E8 32140000  CALL <JMP.&KERNEL32.GetEnvironmentStringsW> [GetEnvironmentStringsW
00404618 * 8105 A1F34000  ADD DWORD PTR DS:[40F3A11,2C2
00404622 * E8 930AFFFF  CALL retnicud_.00401110
00404627 * BF F6BBE853  MOV EDI,53E8BBF6
0040462C * BE E814B42F  MOV ESI,2FB414E8
00404631 * A1 36244100  MOV EAX,DWORD PTR DS:[412436]
00404636 * PUSH 1169B
0040463B * E8 28140000  CALL <JMP.&KERNEL32.SetLastError> [Error = 1169B (71923.)
00404640 > 8105 A1F34000  ADD DWORD PTR DS:[40F3A11,2F8

```

Figure 2: Random Calls to “CopyFileA” API.

- 3) Inside this junk code, the malware implements a very powerful anti-debugging technique. The malware calls the “kerne32.CloseHandle” API with random values of “hObject” (Fig 3.). If a process being debugged tries to close an invalid handle, it generates a STATUS_INVALID_HANDLE (0xC0000008) exception. The only proper way of bypassing this anti-debugging technique is to modify the syscall data from ring3, before it is called or setup a kernel hook. To bypass this anti-debugging technique we will replace all such random values by NULL and this will allow us to debug our malware smoothly.

```

00404D18 * 30 E0524899  JMP EAX,994852E0
00404D1D * 71 0A      JND SHORT retnicud_.00404D29
00404D1F * 68 A0030000  PUSH 3AB
00404D22 * E8 F70C0000  CALL <JMP.&KERNEL32.CloseHandle> [CloseHandle
00404D29 > 0FA809     BT ECX,EBX

```

Figure 3: CloseHandle Anti-debugging technique.

- 4) However, even after bypassing this anti-debugging technique, if you allow the malware to run, it will get executed and terminate with exit code 0 without doing anything or will stop with “Access Violation” exception, depending upon the time elapsed since the program is executed. This is because of the anti-debugging technique implemented by malware using the ‘kernel32.GetTickCount’ API (Fig.4).

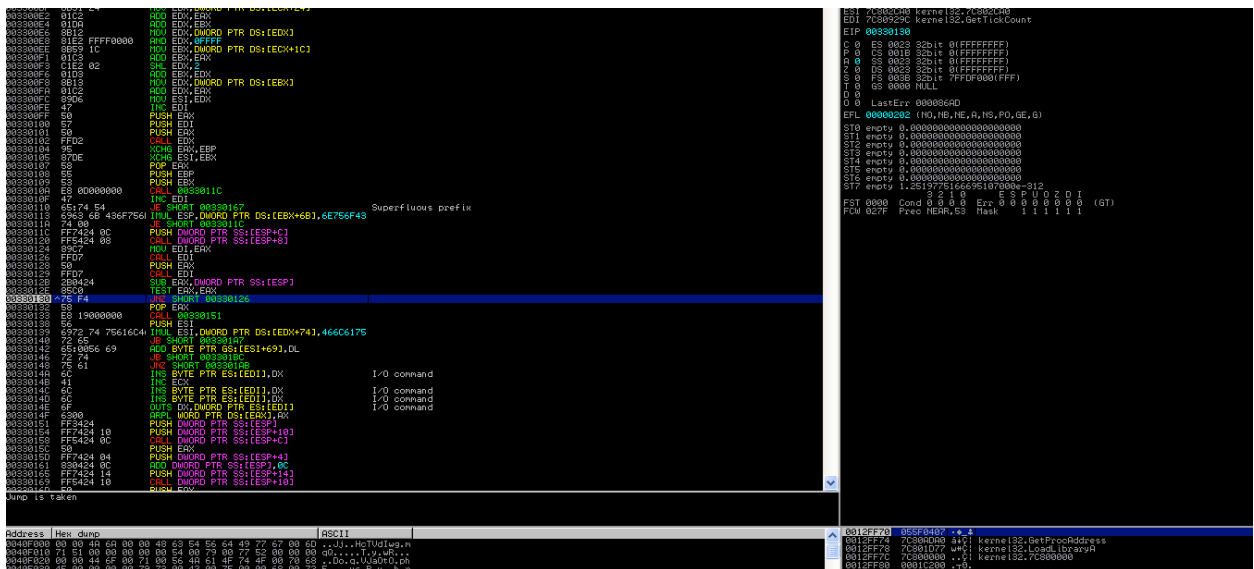


Figure 4: kernel32.GetTickCount Anti-debugging technique.

The instruction at 0x00330126 will call kernel32.GetTickCount and PUSH that value on stack. It again makes the same call, subtracts that value from the one obtained previously and tests if it is zero. It continues in this loop until it gets the subtraction of two values as zero. On every time this loop is executed, the value of kernel32.GetTickCount is pushed on the stack. After coming out of this loop, CALL 00330151 is made. This function makes CALL DWORD PTR SS:[ESP+C], which should ideally be kernel32.GetProcAddress. However if you are debugging the malware, the stack might have values that were pushed on stack because of the previous 'GetTickCount' loop and hence trigger an Access Violation. To bypass this debugging technique you need to adjust the ESP value so that [ESP+C] points to kernel32.GetProcAddress.

- 5) The malware under analysis is created using a CrimeWare Kit that is available in the underground market called CRUM Cryptor Polymorphic by Sunzer Flint (Fig 5). This is a program that is used by malware authors to encrypt malware through a random key of 256 bytes and also subject it to polymorphism.

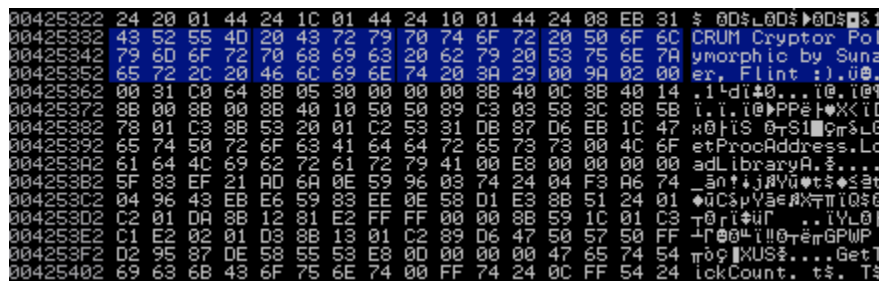


Figure 5: CRUM Cryptor Polymorphic.

- 6) The last two anti-debugging techniques that are implemented by malware before it decrypts itself, is done by accessing the Process Environment Block (PEB) of the current process. The first technique is checking if the byte at offset 0x02(IsDebugged) in the PEB is set or not. If a program is being

debugged, this byte is set to 1 else it is 0. The other anti-debugging technique is to check for the NtGlobalFlags at offset 0x68 in the PEB. If the process is debugged, some flags controlling the heap manipulation routines in ntdll will be set. This anti-debug can be bypassed by resetting the NtGlobalFlags field (Fig. 6).

```

00121F53 E9 4F020000 JMP 001221A7
00121F58 64:8B1D 30000001 MOV EBX,DWORD PTR FS:[30]
00121F5F 8A5B 02 MOV BL,BYTE PTR DS:[EBX+2]
00121F62 8B5D FB MOV BYTE PTR SS:[EBP-5],BL
00121F65 0FBE4D FB MOVSX ECX,BYTE PTR SS:[EBP-5]
00121F69 85C9 TEST ECX,ECX
00121F6B 74 07 JE SHORT 00121F74
00121F6D 33C0 XOR EAX,EAX
00121F6F E9 33020000 JMP 001221A7
00121F74 64:8B0D 30000001 MOV ECX,DWORD PTR FS:[30]
00121F7B 8B59 68 MOV EBX,DWORD PTR DS:[ECX+68]
00121F7E 899D E0FEFFFF MOV DWORD PTR SS:[EBP-120],EBX
00121F84 8B95 E0FEFFFF MOV EDX,DWORD PTR SS:[EBP-120]
00121F8A 83E2 70 AND EDX,70
00121F8D 74 07 JE SHORT 00121F96
00121F8F 33C0 XOR EAX,EAX
00121F91 E9 11020000 JMP 001221A7

```

Figure 6: PEB Anti-debugging Technique.

- 7) Once we have bypassed all these anti-debugging technique, the malware will start importing the different library it requires using the kernel32.LoadLibraryA API.
- 8) The malware then tries to find if the process “explorer.exe” is running on the system and gets handle to this process via the kernel32.OpenProcess API(Fig. 7).

```

001252FD 82          PUSH EDI
001252FE 8B85 D4FEFFFF MOV EAX,DWORD PTR SS:[EBP-12C]
00125304 8B83 30000000 MOV ECX,DWORD PTR DS:[EAX+98]
00125307 FFD1       CALL ECX,EXX          kernel32.1stronp1A
00125309 8299       TEST ECX,ECX
0012530E 75 2A     JNC SHORT 00125330
00125310 8B95 E0FEFFFF MOV EDI,DWORD PTR SS:[EBP-120]
00125316 82       PUSH EDI
00125317 6A 00     PUSH 0
00125318 68 70940000 MOV EAX,478
0012531E 8B85 D4FEFFFF MOV EAX,DWORD PTR SS:[EBP-12C]
00125324 8B48 7C MOV ECX,DWORD PTR DS:[EAX+74]
00125327 FFD1       CALL ECX,EXX
00125329 8B85 D4FEFFFF MOV EDI,DWORD PTR SS:[EBP-120],EDI
0012532F 838D D4FEFFFF CMP DWORD PTR SS:[EBP-130],0
00125332 74 02     JLE SHORT 0012533A
ECX=00000001 (kernel32.1stronp1A)

```

Figure 7: Malware trying to find the “explorer.exe” process.

- 9) The malware then reserves a region of memory within the virtual address space of the “explorer.exe” process using kernel32.VirtualAllocEx API and creates a thread in the explorer.exe process via the kernel32.CreateRemoteThread API (Fig. 8). Once the remote thread is created in the “explorer.exe” process, the malware terminates itself with exit code 0.

```

00123675 51          PUSH ECX
00123676 E8 991B0000 CALL 00125214
0012367B 8B85 4CF8FFFF MOV EDI,DWORD PTR SS:[EBP-7B4],EAX
00123681 8B8D 4CF8FFFF CMP DWORD PTR SS:[EBP-7B4],0
00123688 75 05     JNC SHORT 0012368F
0012368A E9 B5000000 JMP 00123744
0012368F 8A 302C4000 MOV EDI,482C90
00123694 2B95 F4F8FFFF SUB EDI,DWORD PTR SS:[EBP-70C]
0012369A 8B95 48F8FFFF MOV EDI,DWORD PTR SS:[EBP-7B9],EDI
001236A0 8D45 F8 LEA EAX,DWORD PTR SS:[EBP-8]
001236A2 80       PUSH EAX
001236A4 6A 00     PUSH 0
001236A6 8B8D 4CF8FFFF MOV ECX,DWORD PTR SS:[EBP-7B4]
001236AC 51       PUSH ECX
001236AD 8B95 48F8FFFF MOV EDI,DWORD PTR SS:[EBP-7B9]
001236B3 6C       PUSH EDI
001236B4 6A 00     PUSH 0
001236B6 6A 00     PUSH 0
001236B8 8B85 E8F8FFFF MOV EDI,DWORD PTR SS:[EBP-718]
001236BE 51       PUSH EDI
001236BF 8B4D 08 MOV ECX,DWORD PTR SS:[EBP+8]
001236C1 8B91 A0000000 MOV EDI,DWORD PTR DS:[ECX+A0]
001236C3 FFD1       CALL EDI,EXX          kernel32.CreateRemoteThread
001236C4 8915 F4 MOV EDI,DWORD PTR SS:[EBP-C],EAX
001236CD 837D F4 00 CMP DWORD PTR SS:[EBP-C],0
001236D1 75 02     JNC SHORT 001236D5
001236D3 EB 4F     JMF SHORT 00123744
001236D5 8B45 08 MOV EDI,DWORD PTR SS:[EBP+8]

```

Figure 8: Malware Creates a Remote Thread in explorer.exe.

10) Once this new thread is created in the explorer process, the original malware file is copied to “%USERPROFILE%\ctfmon.exe” location (Fig. 9) and sets file attributes to system, read-only and hidden.

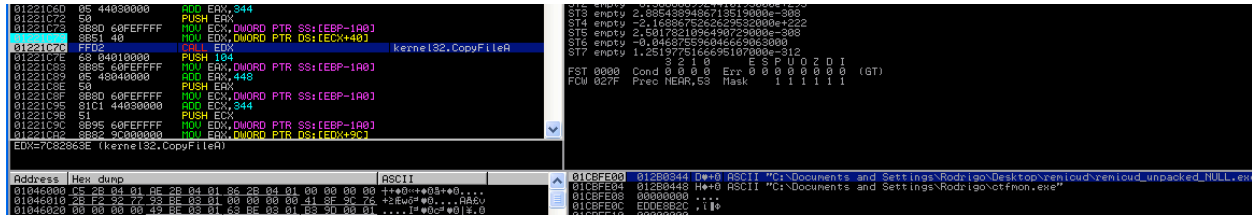


Figure 9: Explorer Thread making a copy of itself as “ctfmon.exe”.

11) After creating the executable, the malware creates the key “HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Taskman”: “%USERPROFILE%\ctfmon.exe” (Fig. 10). This key ensures that every time explorer.exe process is created, the malware gets executed.

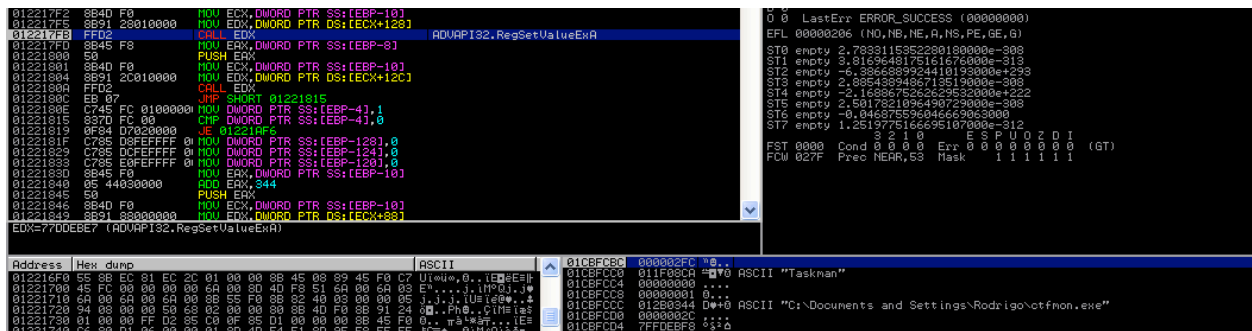


Figure 10: Explorer Thread creating the “TaskMan” registry.

12) The malware creates a NamedPipe which can be later used for inter-process communication (Fig. 11).

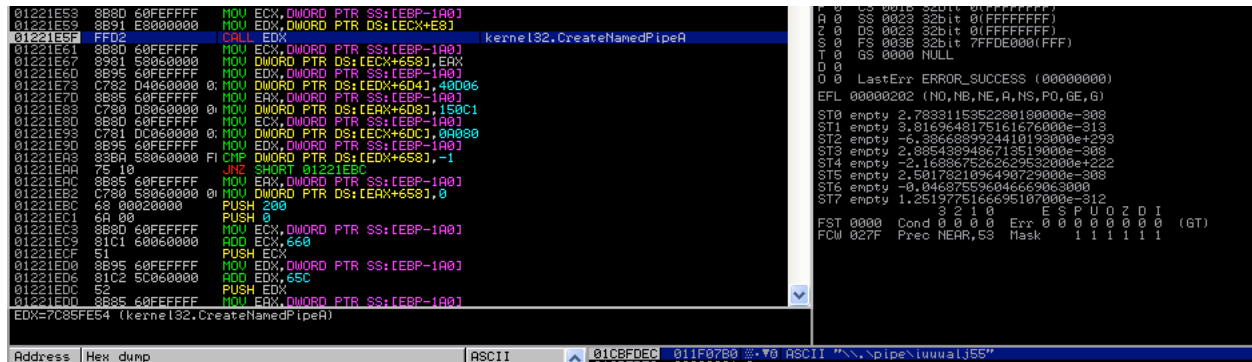


Figure 11: Explorer Thread creating a NamedPipe.

13) The malware then tries to communicate to its masters at “tinaivanovic.sexy-serbain-girls.info” (Fig. 12).

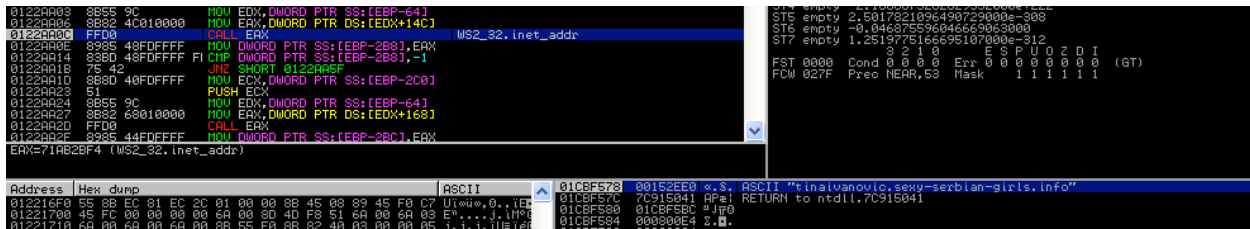


Figure 12: Malware trying to communicate on Internet.

14) The malware is known to spread via USB drives. On connecting a USB stick to an infected host, the malware drops a copy of itself in the “[RemovableDrive]\\\nemoj\\meni.exe” and creates an autorun.inf file (Fig. 13).

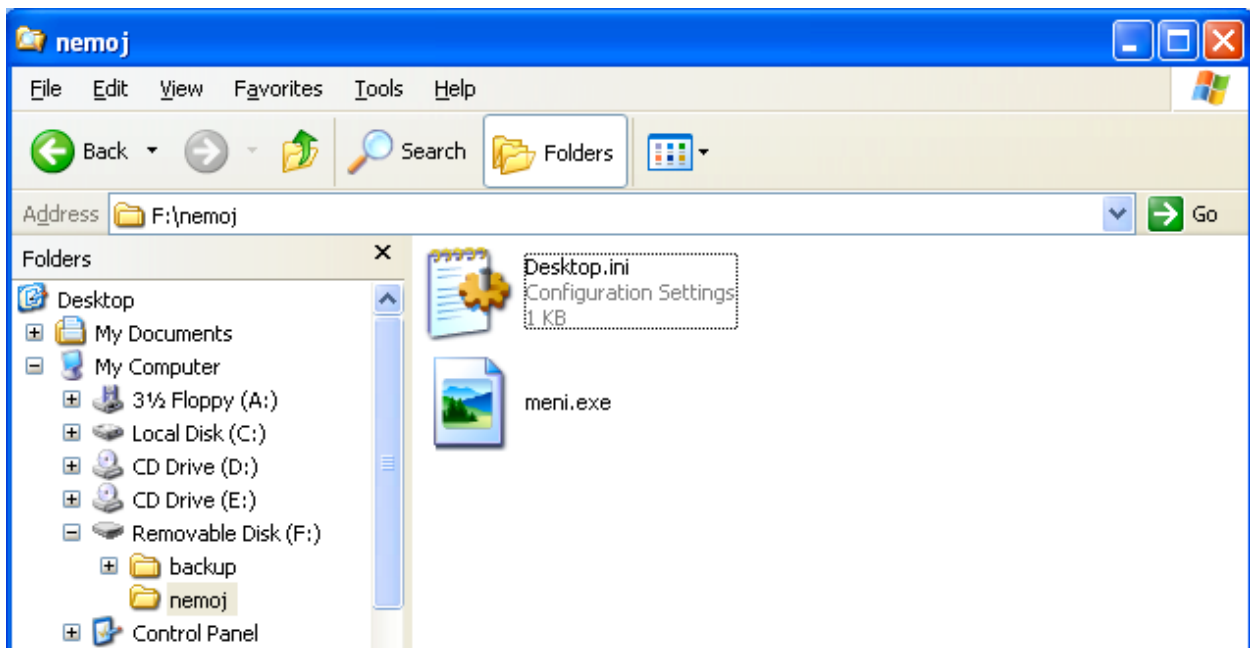


Figure 13: Malware trying to spread via removable drive

Removal Instructions

- 1) Open “Regedit” and locate the above mentioned registry key. Delete this registry key.
- 2) Open “Task Manager” and find explorer.exe in the “Processes” tab. Right click on explorer.exe and select “Kill Process”. If you are comfortable using command line, use the following steps to kill explorer.exe:
 - `tasklist | find /i "explorer"`
This command will give you the process Id of explorer.exe process.
 - `taskkill /PID 12345 /f`
(12345 to be substituted with the process id of explorer.exe obtained from the above step)
- 3) Upon doing this you will notice that another process named “ctfmon.exe” appears in the process list. Kill “ctfmon.exe” as well, same way as we killed explorer.exe.

4) Browse to the %UserProfile% directory using a command line. Use “dir /ah” command to list all the files in that directory. You should be able to see “ctfmon.exe” file in that directory. This file has “SHR” attribute. Remove these attributes of the file so that you can delete this file. Use the following commands to do this:

- `attrib -S -H -R ctfmon.exe`
- `del ctfmon.exe`