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.

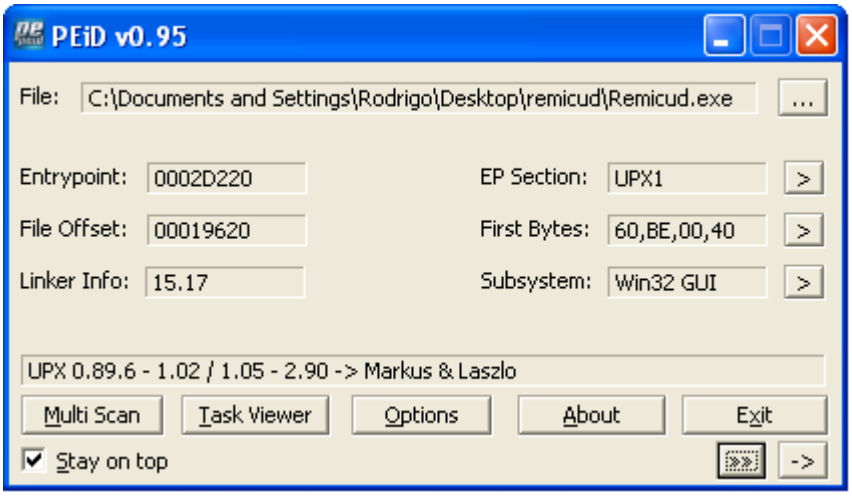

Figure 1: PEid output for malware sample.

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).

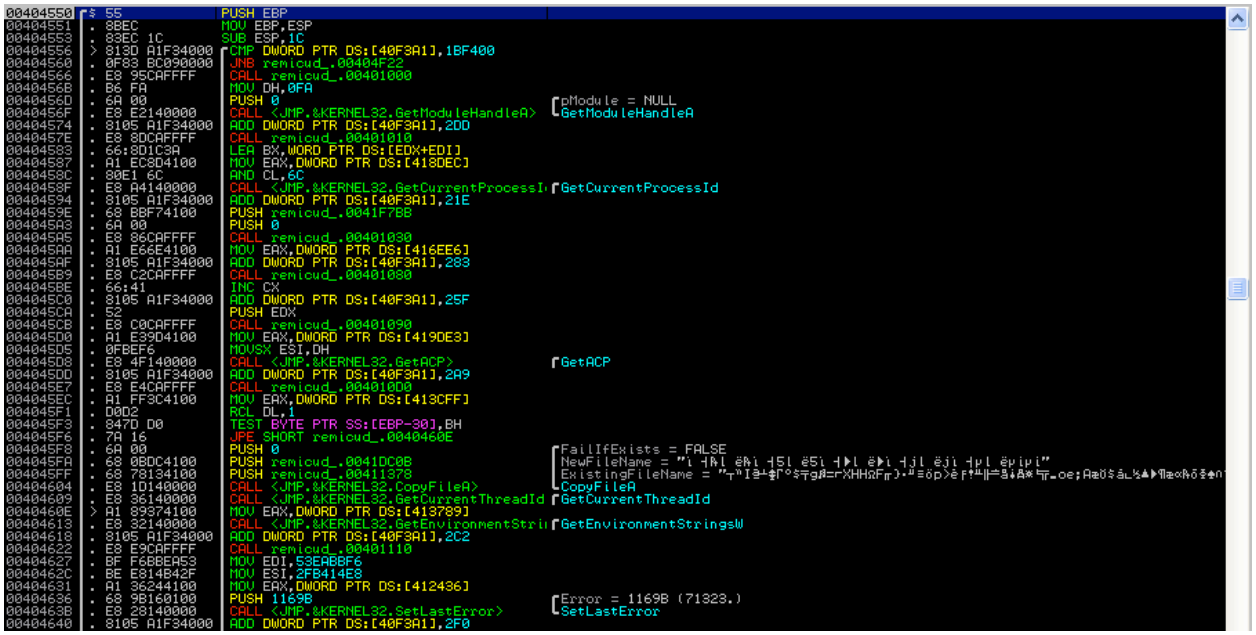

Figure 2: Random Calls to "CopyFileA" API.

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.

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).

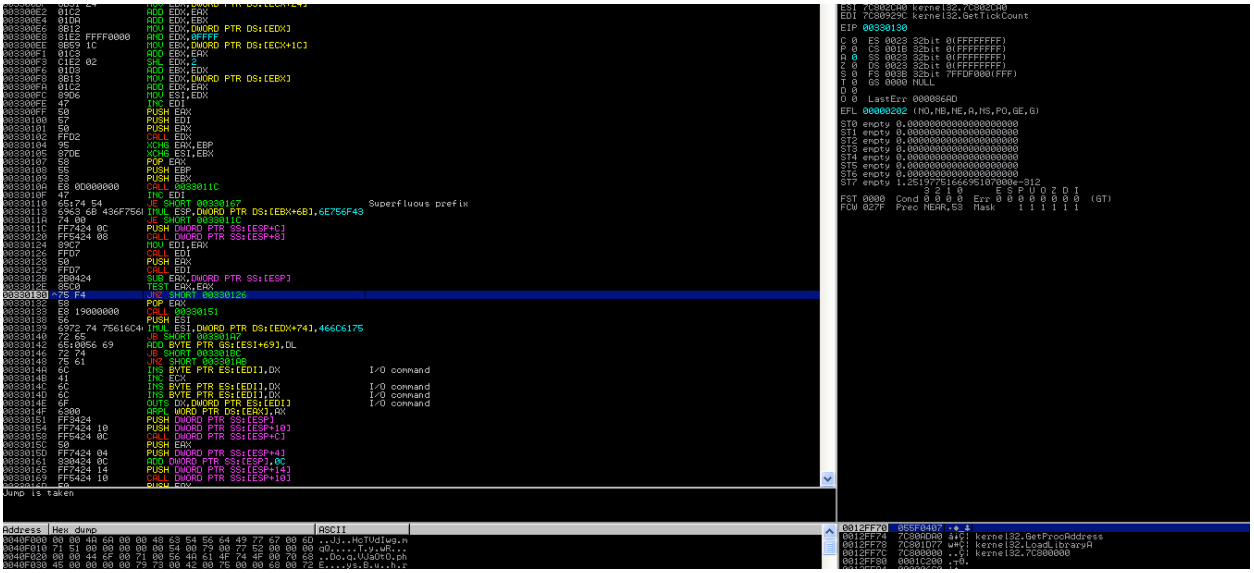

Figure 4: kernel32.GetTickCount Anti-debugging technique.

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 this in loop until it gets the subtraction of these 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 make CALL DWORD PTR SS:[ESP+C], which should ideally be kernel32.GetProcAdddress. 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.

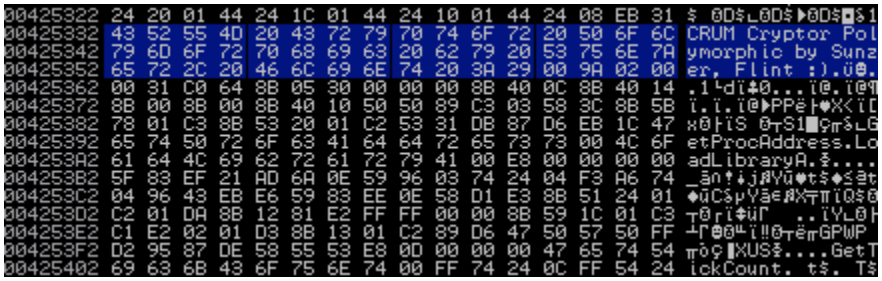

Figure 5: CRUM Cryptor Polymorphic.

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).

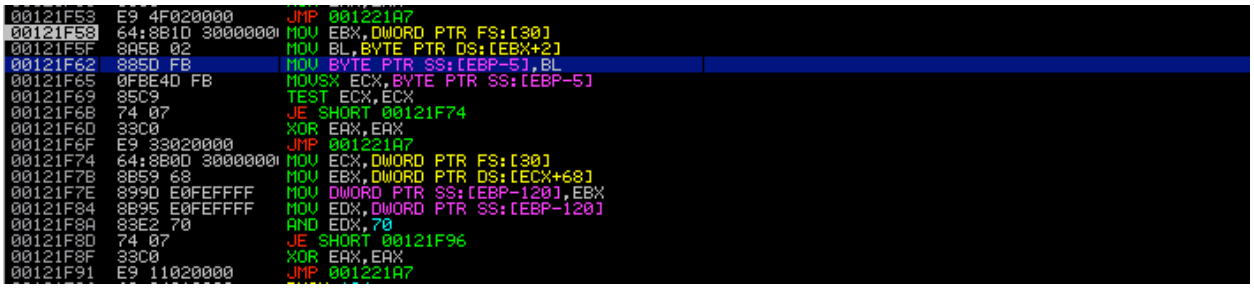

Figure 6: PEB Anti-debugging Technique.

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).

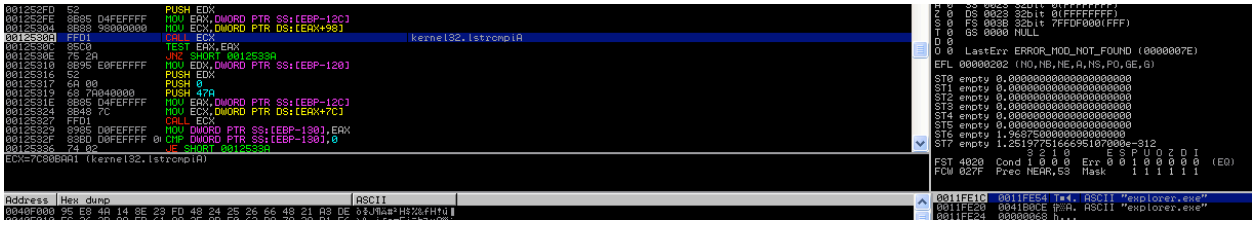

Figure 7: Malware trying to find the “explorer.exe” process.

9) The malware then reserves a region of memory within the virtual address space of the “explorer.exe” process using kernel32.VirutalAllocEx 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.

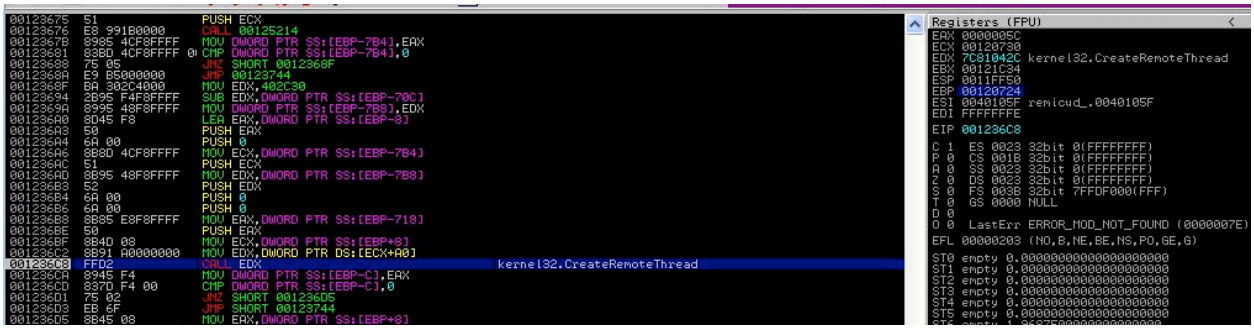

Figure 8: Malware Creates a Remote Thread in explorer.exe.

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.

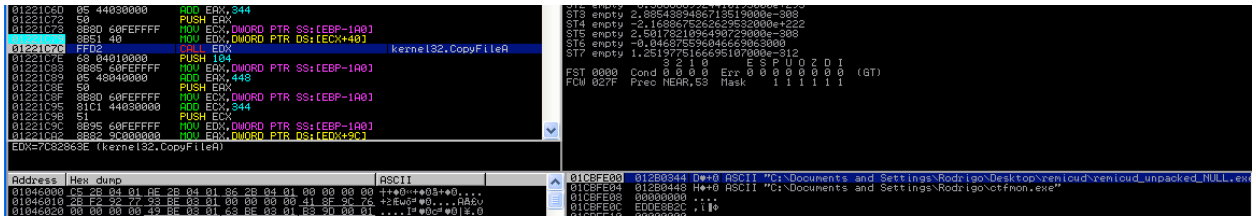

Figure 9: Explorer Thread making a copy of itself as "ctfmon.exe".

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.

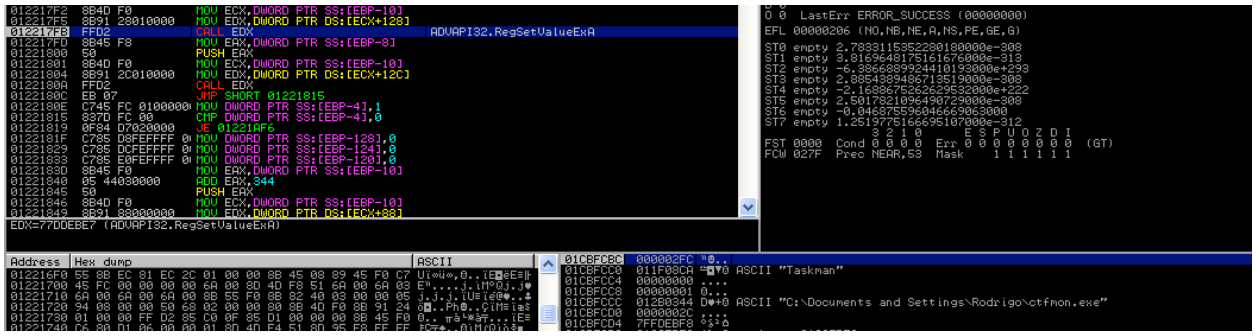

Figure 10: Explorer Thread creating the "TaskMan" registry.

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

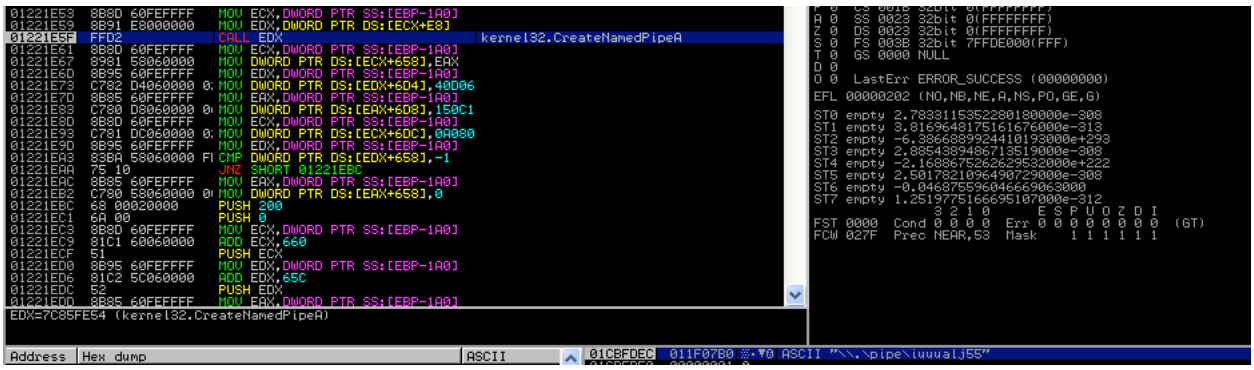

Figure 11: Explorer Thread creating a NamedPipe.

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

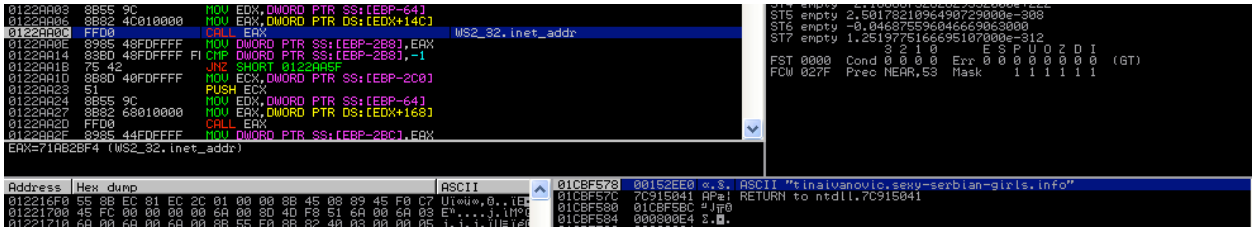

Figure 12: Malware trying to communicate on Internet.

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).

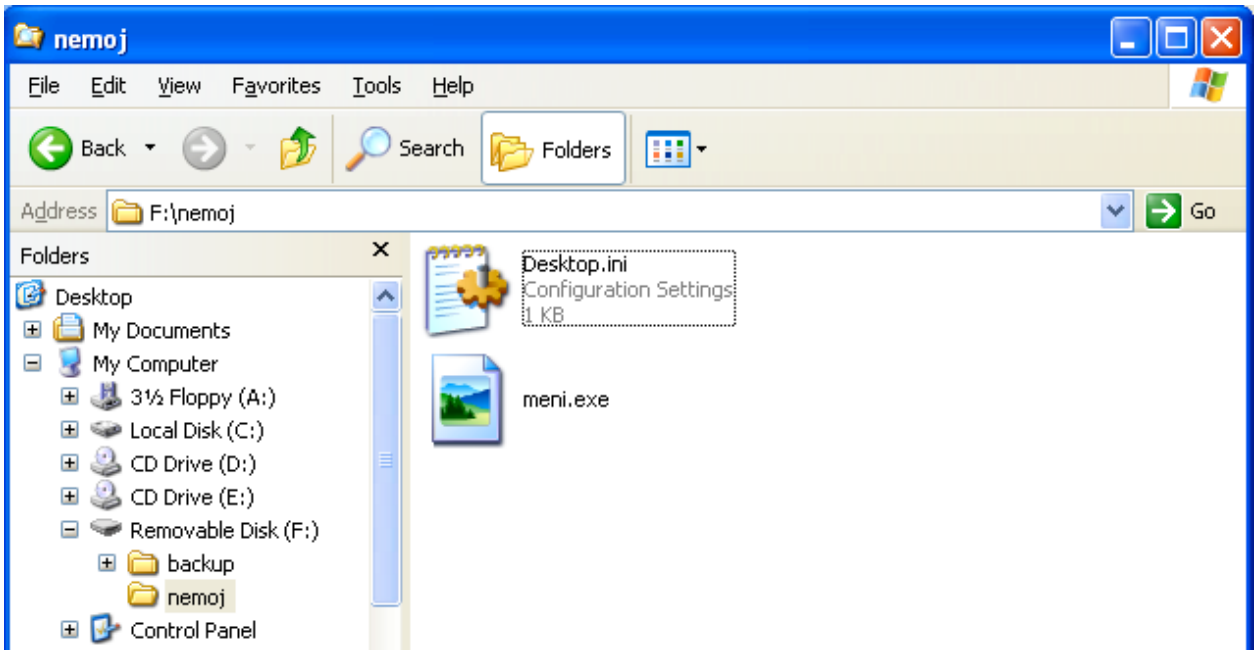

Figure 13: Malware trying to spread via removable drive

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`