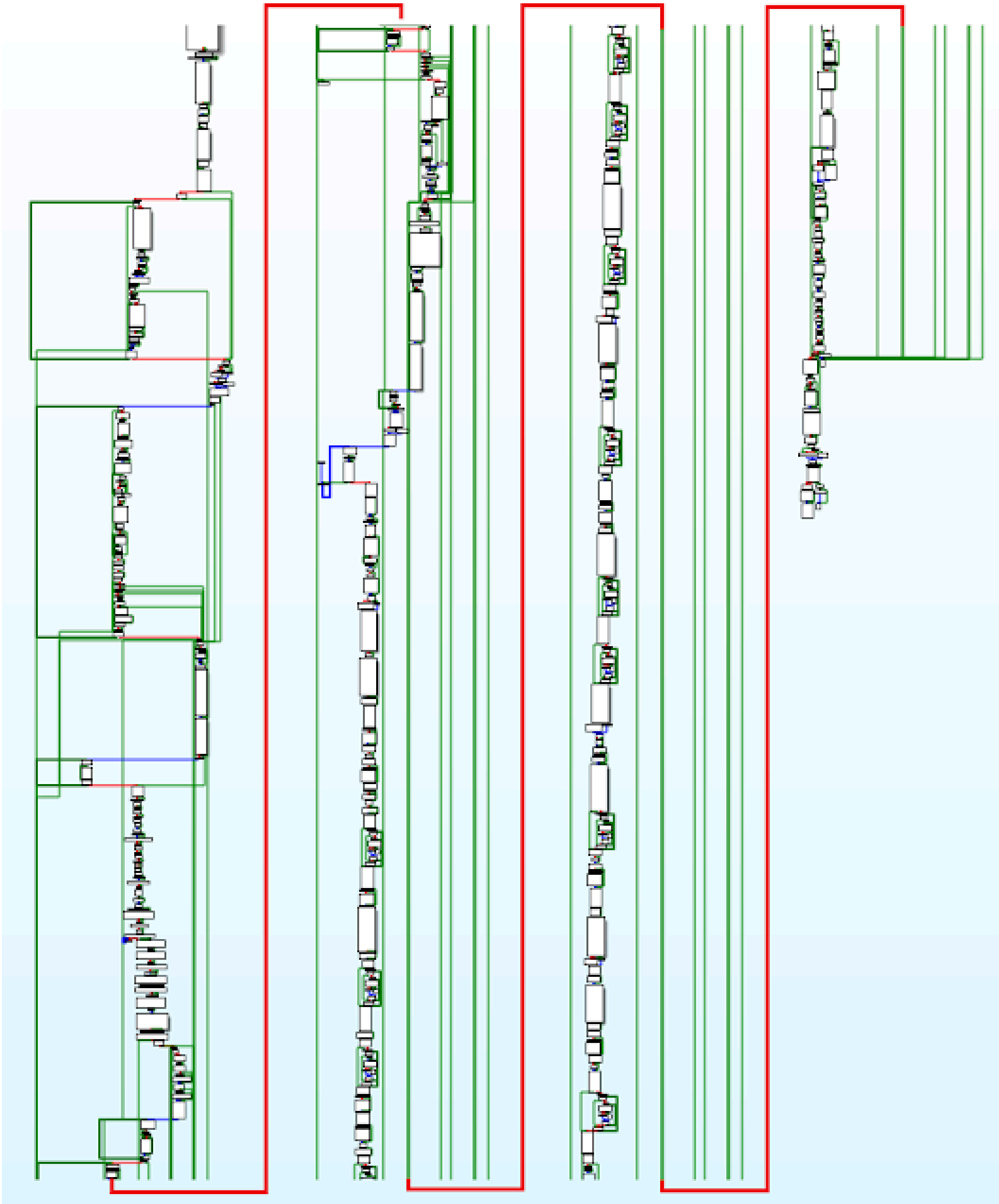
Deep Malware and Phishing Analysis - Nymaim

By Joe Security LLC

Archived: 2026-04-05 18:18:08 UTC

Recently we were investigating interesting piece of malware that was generating quite huge workload in the sandboxed environment. To introduce proper countermeasures we had to fully reverse it. It turned out that the file belongs to the **Nymaim** family, which is active at least since 2013 [1]. This particular file consists of few layers, first one is meant to slowdown / timeout various sandboxes / replicators / emulators, the last layer hinders static analysis and debugging, layers in the middle are just responsible for decompression and decryption.

There are many different methods of introducing slowdowns in sandboxed environments, some of them are more effective some of them are not effective at all. **Nymaim** uses **Win32 API hammering**, which means that it constantly calls benign Win32 API functions in the loop. It's clearly visible on the WinMain function graph (glued side by side, since the function is way too big):



Readers familiar with IDA function graphs should notice unusual length of the code.There is a lot of loops and a lot of various API calls:

API Name	Number of calls
KERNEL32.dll.GetLastError	49739
USER32.dll.GetDlgItem	34446
KERNEL32.dll.TlsGetValue	34434
KERNEL32.dll.SetLastError	34434
dbghelp.dll.SymCleanup	30608

API Name	Number of calls
USER32.dll.ShowWindow	30608
KERNEL32.dll.GetCurrentProcess	30608
KERNEL32.dll.LeaveCriticalSection	15306
KERNEL32.dll.EnterCriticalSection	15306
KERNEL32.dll.CloseHandle	15305
USER32.dll.FindWindowExA	15304
GDI32.dll.MoveToEx	15304
USER32.dll.GetClassNameA	15304
PSAPI.DLL.GetPerformanceInfo	15304
USER32.dll.SetWindowPlacement	15304
KERNEL32.dll.GlobalMemoryStatusEx	15304
USER32.dll.PostMessageA	15304
PSAPI.DLL.EnumProcesses	15304
KERNEL32.dll.GetVersionExA	15304
dbghelp.dll.SymInitialize	15304
ACTIVEDS.dll.ord_9	15304
dbghelp.dll.SymEnumSymbols	15304
GDI32.dll.Rectangle	15304
USER32.dll.GetWindowPlacement	15304
dbghelp.dll.SymLoadModuleEx	15304
ADVAPI32.dll.OpenProcessToken	15304
USER32.dll.EnumDisplaySettingsA	**7652**
USER32.dll.SendDlgItemMessageA	7652
OLEAUT32.dll.ord_4	7652
USER32.dll.GetDC	3839
KERNEL32.dll.GetProcAddress	3828

API Name	Number of calls
KERNEL32.dll.lstrlenA	3828
KERNEL32.dll.GetModuleHandleA	3827
KERNEL32.dll.WideCharToMultiByte	3826
GDI32.dll.ChoosePixelFormat	3826
OLEAUT32.dll.ord_8	3826
KERNEL32.dll.MultiByteToWideChar	3826
GDI32.dll.SetPixelFormat	3826
OPENGL32.dll.wglCreateContext	3826

Without any monitoring tools, execution of WinMain takes around 46 seconds. Now if any of the above APIs triggers some event that is (or should be) monitored in the sandboxed environment you can imagine what would happen to those 46 seconds. So far we have not seen any sandbox able to analyze the malware successfully.
During the analysis we were able to identify one unwelcome side effect of the **USER32.dll.EnumDisplaySettingsA** function call, namely it loads and unloads the **vga.dll** kernel library during the call:

ChildEBP RetAddr
890136f8 828563ef nt!DbgLoadImageSymbols+0x47
89013714 82a05b21 nt!DbgLoadImageSymbolsUnicode+0x23
89013750 82a02531 nt!MiDriverLoadSucceeded+0x183
890137d0 82a8ccf8 nt!MmLoadSystemImage+0x720
8901391c 8287a8c6 nt!NtSetSystemInformation+0x967
8901391c 82879969 nt!KiSystemServicePostCall
890139a0 907a895a nt!ZwSetSystemInformation+0x11
89013b1c 907a858b win32k!ldevLoadImage+0x215
89013b54 907a2b9a win32k!ldevLoadDriver+0x78
89013b70 907aaec4 win32k!ldevGetDriverModes+0x1c
89013b9c 90806eb6 win32k!DrvBuildDevmodeList+0x134
89013bfc 90806aea win32k!DrvEnumDisplaySettings+0x3b9
89013c1c 8287a8c6 win32k!NtUserEnumDisplaySettings+0x27
89013c1c 772270f4 nt!KiSystemServicePostCall
001241d0 762f13c4 ntdll!KiFastSystemCallRet
001241d4 763065c1 USER32!NtUserEnumDisplaySettings+0xc
00124214 76306502 USER32!EnumDisplaySettingsExA+0xbc
0012422c 004023d4 USER32!EnumDisplaySettingsA+0x23
0012fef8 0040698f 885+0x23d4
0012ff88 760eee1c 885+0x698f

0012ff94 772437eb kernel32!BaseThreadInitThunk+0xe

0012ffd4 772437be ntdll!__RtlUserThreadStart+0x70

0012ffec 00000000 ntdll!_RtlUserThreadStart+0x1b

This of course triggers driver analysis (in case the sandbox offers it)... 7652 times and the only thing that separates good analysis and total failure is a proper filtering of collected data.

Second stage of the malware is executed through the callback from **EnumResourceTypesA**, it decompress and decrypts the final stage, which is heavily obfuscated. Entry point of the final stage suggests that it can be run from within both x86 and x64 processes. It uses simple trick to detect bitness of the process [2] and it contains both x86 and x64 payloads.

x64 decoding

```
                is32Bits      proc near
31 C0                         xor     eax, eax
40 90                         xchg    eax, eax
C3                            retn
                is32Bits      endp
```

x86 decoding

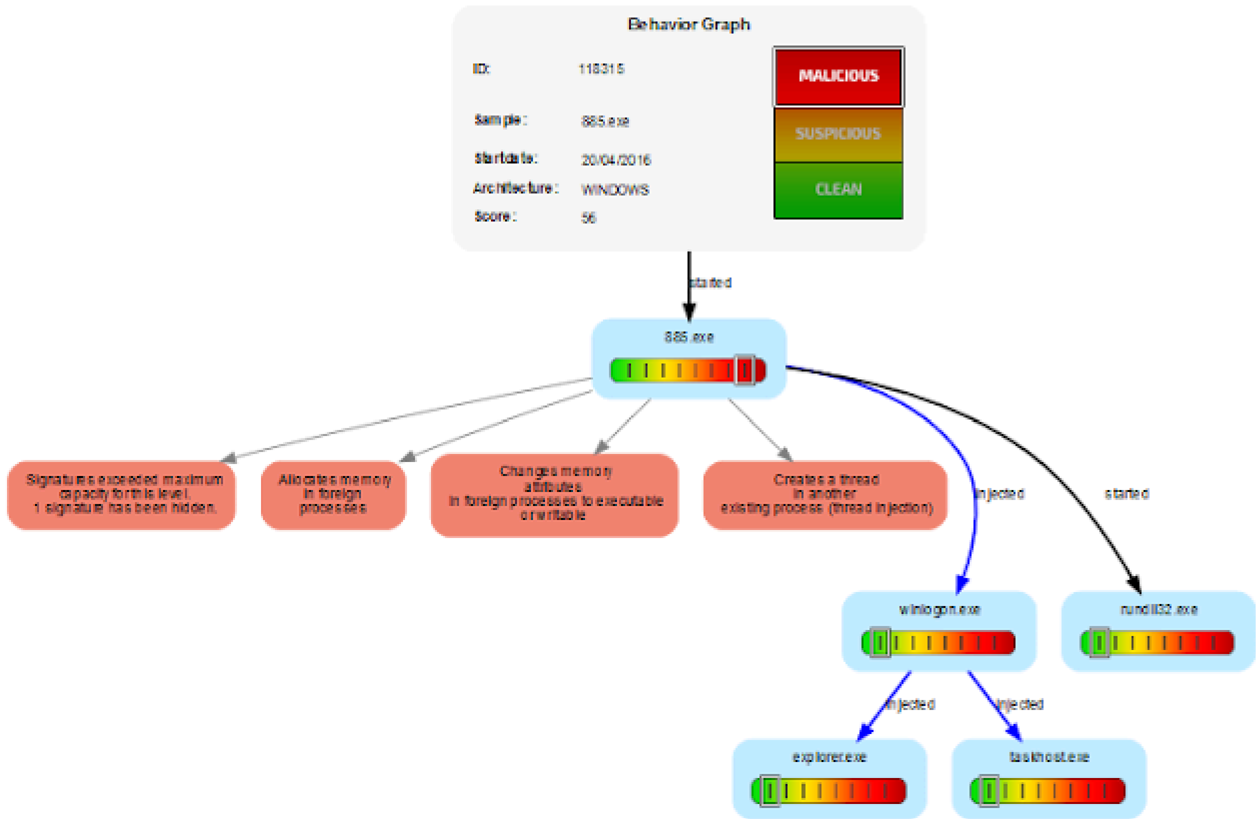
```
                is32Bits      proc near

31 C0                         xor     eax, eax
40                            inc     eax
90                            nop
C3                            retn
                is32Bits      endp
```

After deobfuscation we were able to identify another simple check to evade analysis, mentioned sample uses **GetSystemTime** API to verify expiration date, and does not execute after **8th of April 2016**. We can easily handle such cases in Joe Sandbox through our Cookbooks system [3], whole operation boils down to adding below line to the Cookbook:

_SetDate(06, 04, 2016)

It's always good to re-run the analysis with the different dates to verify if the sample doesn't expire or if it isn't activating in the near future. Usually it's safe to assume the day when the sample was received as the initial date, timestamp from the PE header should work as well (in this case it is GMT Tue Apr 05 22:31:47 2016).

Last but not least, link to the full Joe Sandbox report (click the picture to open):



Nymaim proves that it is very important to have a flexible malware analysis system which enables analysts to easily change settings on the analysis machine. Joe Sandbox features an extensive technology called cookbooks. Cookbooks enable to completely define the analysis and allow to change OS settings, simulate user behavior and more. Further Joe Sandbox analyzes malware on physical machines (bare metal) defeating any VM evasions.

References:

[1] http://www.welivesecurity.com/2013/08/26/nymaim-obfuscation-chronicles/

[2] http://www.ragestorm.net/blogs/?p=376

[3] https://www.joesecurity.org/joe-sandbox-technology#cookbooks

Source: https://www.joesecurity.org/blog/3660886847485093803