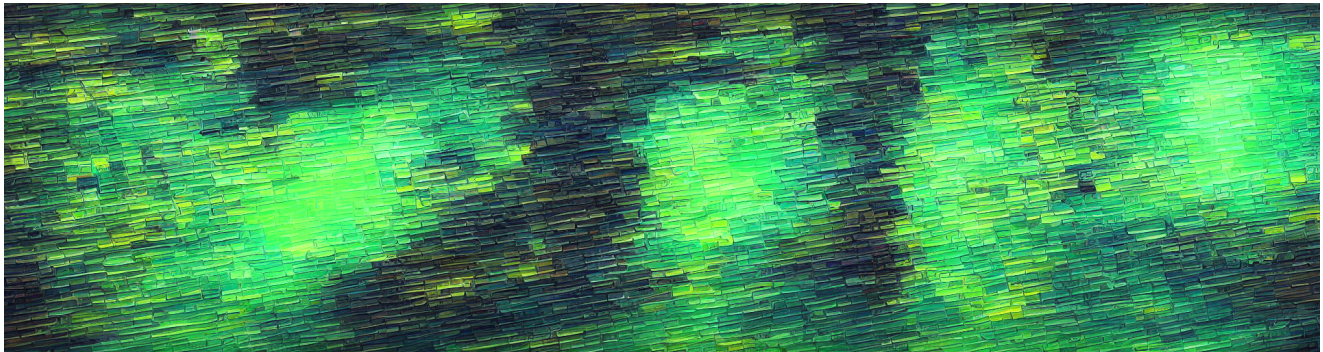


Not always and open and shut case

 gdatasoftware.com/blog/2022/09/37511-detecting-file-manipulation-in-system-files

09/21/2022

Identifying file manipulation in system files



Malware

Sometimes people send files to us that seem to be legitimate Microsoft system files at first glance, yet closer inspection reveals, that they have in fact been modified. Are those manipulations always malicious? And how can file manipulations be identified? Here are seven different ways to do that.

File modifications happen for a number of reasons, the most innocuous one being data corruption or inadvertent partial downloads. Both scenarios often result in non-working files. However, attackers and viruses manipulate original files in a manner that they still work, but additionally execute their own malicious code. In some cases the malicious code is not even there anymore because the files have been cleaned by antivirus software, but the indications of manipulation remain.

Regardless of the reason that these manipulations occur, being able to identify them is important to avoid instability, less secure systems and system infections.

1. Obtain the original and look out for winking bytes

The best way to detect manipulation is a direct comparison with the original file. There are repositories that enable you to obtain original files from Windows systems, e.g., [VanillaWindowsReference](#) by Andrew Rathbun. In case the suspect is not a Microsoft file, the original file can often be downloaded from the developer's website. Just make sure to get the exact same version as noted in the **version information** of the file.

A first comparison step is done via hashes like SHA-256. If these do not match, binary diff tools like VBinDiff will show which specific bytes are different and allow for closer inspection. Colleagues of mine named these "winking" bytes because when switching tabs to-and-fro between the original and suspect file in a hex editor view, the divergent bytes create something resembling a wink.

.\originalMSwinXPFile																			
0000	0160:	00	00	00	00	00	00	00	00	54	A3	00	00	B4	00	00	00		Tú..†...
0000	0170:	00	E0	01	00	50	0D	00	00	00	00	00	00	00	00	00	00	.Ö..P...	
0000	0180:	00	A0	32	00	18	24	00	00	00	00	02	00	B0	06	00	00	.ä2..\$. .	
0000	0190:	E0	21	00	00	1C	00	00	00	00	00	00	00	00	00	00	00	ó!.....	
0000	01A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000	01B0:	68	28	00	00	40	00	00	00	00	00	00	00	00	00	00	00	h(..@...	
0000	01C0:	00	20	00	00	DC	01	00	00	00	00	00	00	00	00	00	00	. . . ■ .	
0000	01D0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000	01E0:	2E	74	65	78	74	00	00	00	08	8E	00	00	00	20	00	00	.text... .	A... .
0000	01F0:	00	90	00	00	00	04	00	00	00	00	00	00	00	00	00	00	.É.....	
0000	0200:	00	00	00	00	20	00	00	00	2E	64	61	74	61	00	00	00		data...
0000	0210:	F0	16	01	00	00	C0	00	00	00	04	00	00	00	94	00	00	-.....L..	ö..
0000	0220:	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	C0	@..	L
0000	0230:	2E	72	73	72	63	00	00	00	50	0D	00	00	00	E0	01	00	.rsrc... .	P....Ö..
0000	0240:	00	0E	00	00	00	98	00	00	00	00	00	00	00	00	00	00	y..	
0000	0250:	00	00	00	00	40	00	00	40	2E	72	65	6C	6F	63	00	00	@..@	reloc..
0000	0260:	BE	0A	00	00	00	02	00	00	00	FA	31	00	00	A6	00	00	¥.....	..1..a..
0000	0270:	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	42		...@..B
0000	0280:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000	0290:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000	02A0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000	02B0:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		

"Winking" bytes in the section table of a manipulated file.

2. PE Checksums and reproducible builds

PE files may contain several checksums which are useful to detect manipulation. While some of them are well-known, others might be surprising.

The **Optional Header checksum** is always present, but certainly the least useful. The reason is that Microsoft files built in recent years don't have valid checksums in their Optional Header anymore. However, if you happen to find a file with a valid checksum and one without, the valid one is probably the original file.

[@namazso](#) made me aware that Microsoft files created within the last few years generally use the `/BREPRO` flag for the linker. That flag is used to create so called reproducible builds, which means that different builds based on the same source will have the same hash for the resulting binary. To accomplish this PE-related timestamps and the Optional Header checksum are replaced with a fixed value. This does not only lead to unusable timestamps but also an invalid Optional Header checksum. The `/BREPRO` flag adds an entry to the debug section of the debug type `IMAGE_DEBUG_TYPE_REPRO (0x16)` and this entry in turn contains a SHA-256 hash value. As there is no documentation or name for this hash yet, it will henceforth be referenced as **repro hash**.

	Time	Type	Size	RVA	Pointer
8C928EC1	cv	57	000024D4	18D4	Format: RSDS, {9018817B-8665-DFB5-A777-6960A69F80A7}, 1, breproexample.pdb
8C928EC1	feat	14	0000252C	192C	Counts: Pre-VC++ 11.00=0, C/C++=29, /GS=29, /sd1=1, guardN=28
8C928EC1	coffgrp	284	00002540	1940	4C544347 (LTCG)
8C928EC1	iltcg	0	00000000	0	
8C928EC1	repro	24	000027C4	1BC4	7B 81 18 90 65 86 B5 DF A7 77 69 60 A6 9F 80 A7 98 D0 79 5A 6A 51 AC 6F 7F 54 FA 67 C1 8E 92 8C

Debug section with repro entry and repro file hash. Output via Dumpbin.exe /HEADERS (click to enlarge)

For all files that were created with /BREPRO, manipulations can be detected by checking the validity of the **repro hash** value. I am not aware of any tool that can validate the repro hash at present, but I am going to add this validity check as a feature to [PortexAnalyzer](#) for the upcoming release. The algorithm is [described in more detail by @sixtyvividtails](#).

The checksum of the certificate is only present if the file was signed. This checksum is called **authentihash** and shown on VirusTotal. Invalid checksums lead to an invalid signature. VirusTotal as well as tools like Sysinternals sigcheck.exe show the validity of the certificate. However, the authentihash does not cover the certificate table which has been abused by malware authors. Some files have encoded malware in the certificate section, see also [Code Signing: How Malware Gets a Free Pass](#). VirusTotal currently shows such manipulated files as having invalid certificates, whereas sigcheck.exe does not.

The **Rich Header checksum** is the same as the xor key which is used to encrypt the Rich Header. If this checksum is invalid, it means the Rich Header or DOS stub has been manipulated after linking. Some packers do this to the files that they pack but malware developers may also manipulate this area on purpose.

3. Original files have higher distribution numbers

Manipulated files are generally not as wide-spread as the original version. So you can use this to your advantage and search for similar files with the same version number and compare their distribution numbers. E.g., VirusTotal has a similar samples search and VirusTotal's submitters value is an indicator of said distribution.

The result below shows one file with 600 submitters and four other files with less than five submitters each. The executable with 600 submitters is in stark contrast to the other files, indicating higher distribution numbers. Additionally this is the only validly signed file, so that we can conclude that the file with 600 submitters is most likely the original one.

FILES 5 / 5

90 days

	Detections	Size	First seen	Last seen	Submitters	
☐ A8824C0607C2E915AF6CF1A30962016D741A60B46292492372E2CCD089A722A6 WindowsXP-KB942288-v3-x86.exe peexe runtime-modules direct-cpu-clock-access checks-disk-space	3 / 69	3.16 MB	2010-05-19 20:38:39	2022-08-31 21:36:42	4	
☐ ACD589B57ED104B5E04A7CE28DC166654C09060C9C31AE8E2C9301FA0988BFA5 c:\users\brkpe\AppData\Local\Temp\...Winrar.TMP\msi\WindowsXP-KB942288-v3-x86.exe peexe checks-disk-space runtime-modules overlay signed direct-cpu-clock-access via-tor known-distributor ...	0 / 70	3.17 MB	2008-09-28 15:01:31	2022-08-28 13:30:03	600	
☐ 3C7B4E3904F7815A0E141B26FBE04C0A846BF10AA67E0ED131E95ABEC4AD1BE5 WindowsXP-KB942288-v3-x86.exe peexe runtime-modules direct-cpu-clock-access	12 / 70	3.17 MB	2018-06-06 17:27:31	2022-08-17 07:48:30	4	
☐ 27AB88BDDA20873F7808C199134CBE234E57593C0F04A4D9466C44891328A71A %TEMP%\..cache_%SAMPLENAME% peexe overlay	1 / 67	4.24 MB	2016-01-29 13:00:14	2022-06-28 20:35:42	2	
☐ 4EDE89D604D103CCBD2F639499BAE14227BA7FC5C78B5157173EEE424C789809 WindowsXP-KB942288-v3-x86.exe peexe runtime-modules checks-disk-space direct-cpu-clock-access invalid-rich-pe-linker-version	15 / 68	3.16 MB	2022-06-16 15:32:53	2022-06-16 15:32:53	1	

VirusTotal results after a similarity search. The original file has 600 submitters and is the only signed file.

4. Patches in the Section Table

The best way to identify PE file manipulation is a static PE parser that shows the header contents. Suitable tools are, e.g., [PortexAnalyzer](#) (CLI) and [PEStudio](#) (GUI).

Many viruses change the section characteristics, so they can place their own code inside a file. Even if such an infected file is cleaned by an antivirus program, the modified headers will still be present because the cleaning tool cannot know what the header looked like before the virus changed it.

A prominent red flag for non-packed files is the presence of **write** and **execute** characteristics in a section, especially if that is the section containing the entry point. Most of the time write and execute characteristics do not appear together in a section in non-packed files, whereas it is rather typical for packed files. The presence of both means the code itself can be changed dynamically. The section characteristics are shown by PE parsers in the section table.

Section Table

	1. .text	2. .data	3. .rsrc	4. .reloc
Entropy	6.51	5.89	3.38	5.16
Chi squared	291250.78	30558.00	319747.86	157236.88
Pointer To Raw Data	0x400	0x9400	0x9800	0xa600
Size Of Raw Data	0x9000	0x400	0xe00	0x31fa00
-> actual read size				0x1000
Physical End	0x9400	0x9800	0xa600	0xb600
Virtual Address	0x2000	0xc000	0x1e000	0x20000
Virtual Size	0x8e08	0x116f0	0xd50	0xabe
-> actual virtual size	0x9000	0x12000	0x1000	0x1000
Pointer To Relocations	0x0	0x0	0x0	0x0
Number Of Relocations	0x0	0x0	0x0	0x0
Pointer To Line Numbers	0x0	0x0	0x0	0x0
Number Of Line Numbers	0x0	0x0	0x0	0x0
Code	x			
Initialized Data		x	x	x
Discardable				x
Execute	x			
Read	x	x	x	x
Write	x	x		

The section table of a manipulated file with execute, read and write characteristics in the first section
(click to enlarge)

5. Fractionated imports

Another virus-typical behaviour is the introduction fractionated imports. Generally all imports are placed in one section, but if they are spread over different sections, they are called fractionated. Some viruses add imports deliberately to make sure they can use certain system APIs. The fractioning is an unintended side-effect from placing the imports at a virus-convenient location that is usually not near the imports of the original file.

The tool [PortexAnalyzer](#) automatically detects such imports and marks them as an anomaly.

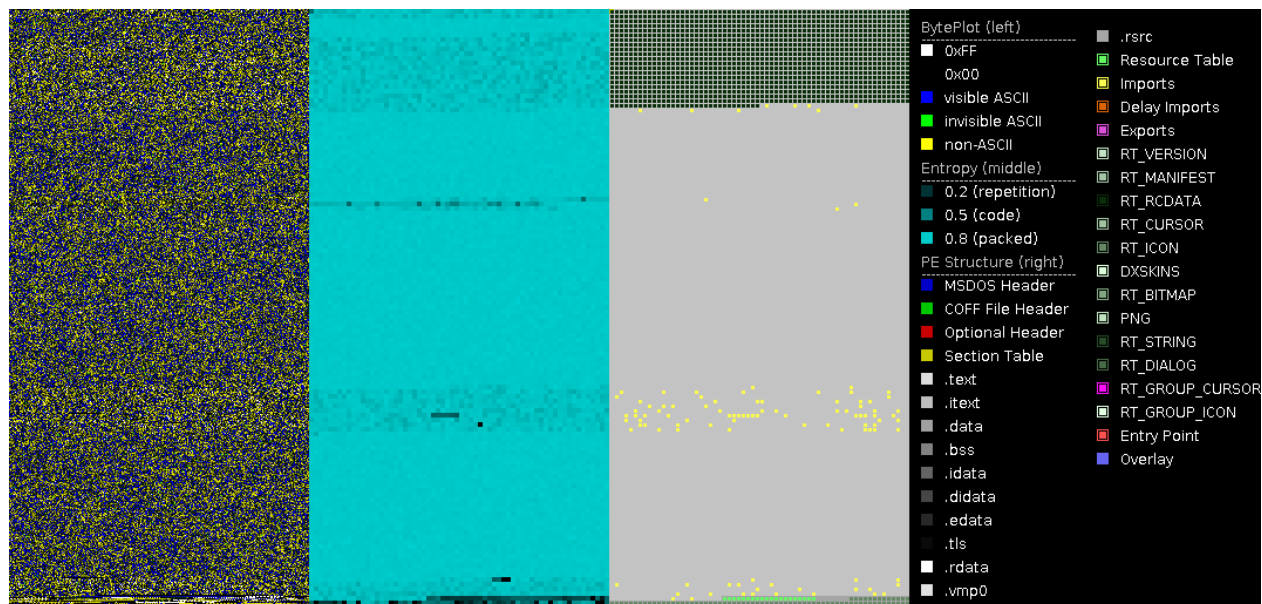
Fractionated imports are also common among certain packed files, but in such cases the imports are often scattered throughout the whole file as in the image below. These are not indicative of virus modifications.

```

136 Imports
137 *****
138
139 KERNEL32.DLL
140 -----
141 [Dynamic-Link Library]
142 rva: 0x36000, va: 0x436000, offset: 0x2400, hint: 0, name: LoadLibraryA -> Maps the specified
143 executable module into the address space of the calling process.
144 rva: 0x36004, va: 0x436000, offset: 0x2404, hint: 0, name: GetProcAddress -> Retrieves the
145 address of an exported function or variable from the specified DLL.
146
147 advapi32.dll
148 -----
149 [Registry]
150 rva: 0x34078, va: 0x434078, offset: 0x478, hint: 0, name: RegCloseKey -> Closes a handle to the
151 specified registry key.
152
153 kernel32.dll
154 -----
155 [Process and Thread] <Thread>
156 rva: 0x34080, va: 0x434080, offset: 0x480, hint: 0, name: Sleep -> Suspends the execution of the
157 current thread for a specified interval.
158
159 user32.dll
160 -----
161 [Timer]
162 rva: 0x34088, va: 0x434088, offset: 0x488, hint: 0, name: SetTimer -> Creates a timer with the
163 specified time-out value.
164
165 wsock32.dll
166 -----
167 [Winsock]
168 rva: 0x34090, va: 0x434090, offset: 0x490, hint: 0, name: recv -> Receives data from a connected
169 or bound socket.

```

The KERNEL32.DLL imports of GetProcAddress and LoadLibraryA have been added by Salty and placed in a different section than the other imports. (Click to enlarge)



Scattered imports (yellow rectangles, right part of the image) typically seen in packed files. These are not indicative of virus manipulation. (Click to enlarge)

6. Linker mismatch

If the Rich Header is present, a linker mismatch between the Rich Header linker and Optional Header linker version indicates manipulation of the headers. Since the Rich Header is a means to attribute samples to threat actors, malware authors might get the idea to swap the DOS Stub and Rich Header with those of other threat actor's samples. In those cases the Rich Header checksum will still be valid but the linker versions might not match anymore.

VirusTotal highlights linker mismatches using a warning, but said warning is not always accurate. There seem to be issues with newer or unknown Visual Studio versions, so I suggest to rather compare the linker versions yourselves. PE parsers like PortexAnalyzer and PESTudio are able to interpret the linker version in the Rich Header. Compare this value with the Optional Header **major linker version** and **minor linker version**, e.g., by using PortexAnalyzer to get a textual representation of the version numbers.

History ⓘ	
Creation Time	2008-05-20 11:58:08 UTC
First Submission	2022-06-16 15:32:53 UTC
Last Submission	2022-09-01 12:14:29 UTC
Last Analysis	2022-09-01 12:14:39 UTC

Capabilities And Indicators ⓘ	
ⓘ	The PE linker versions conflict, meaning that that the PE Rich header or PE header has been modified.

VirusTotal displays a message in the Details tab if the linker versions in the Optional Header and the Rich Header are not the same. (Click to enlarge)

7. Truncated files

Files may be truncated, e.g., from partial downloads or because threat actors cut the invalid certificate table from the end of the file. Depending on how much has been cut, this does not always result in file corruption. But it might show up in the section table or data directory.

For truncated files, the section table may define sections whose raw pointer + size values point to somewhere beyond the end of the file. The data directory may point to structures outside of the file as well. Both are treated as anomalies in PortexAnalyzer reports.

data directory	rva	-> offset	size	in section
import table	0xa354	0x8754	0xb4	1 .text
resource table	0x1e000	0x9800	0xd50	3 .rsrc
certificate table	0x32a000	0x32a000	0x2418	-
base relocation table	0x20000	0xa600	0x6b0	4 .reloc
debug	0x21e0	0x5e0	0x1c	1 .text
load config table	0x2868	0xc68	0x40	1 .text
IAT	0x2000	0x400	0x1dc	1 .text

The filesize of this file is 0x32a000 which means the certificate table has been cut off (click to enlarge)

Anomalies

Total anomalies: 9

- * Checksum is invalid. Actual checksum is 0x1658f
- * Optional Header: size of initialized data is too small (0x1e00), it should be 0x2200
- * Import function typical for code injection: LoadLibraryA maps module into the address space of the calling process
- * Import function typical for code injection: CreateProcessA creates a process (check if SUSPENDED flag is used)
- * Import function typical for code injection: CreateThread is used to open and execute a thread in the victim process
- * Optional Header: The default image base is 0x00400000, but actual value is 0x1000000
- * Section Header 1 with name .text has unusual characteristics, that shouldn't be there: Write
- * Entry point is in writeable section 1 with name .text
- * Optional Header: invalid data directory entry for certificate table, its entry points outside of the file
- * Section 1 with name .text has write and execute characteristics.

The anomaly report of PortexAnalyzer for a truncated file where the certificate table was cut off (click to enlarge)

The repro hash needs more attention

Of all these ways to identify file manipulation the most surprising for me was the repro hash. Apart from the tweet by @sixtyvividtails the hashing algorithm has not been documented yet. But once it is built into tools like PortexAnalyzer and services like VirusTotal, it will be a great asset for malware analysts.

Thank you for your contributions regarding the repro hash [@namazso](#) and [@sixtyvividtails](#)



Karsten Hahn

Principal Malware Researcher

Share Article

© 2025 G DATA CyberDefense AG. All rights reserved.

© 2025 G DATA CyberDefense AG. All rights reserved.