Journal of Computer Virology and Hacking Techniques
https://doi.org/10.1007/s11416-020-00370-y

ORIGINAL PAPER

Superfetch: the famous unknown spy

Mathilde Venault[1] · **Baptiste David**[1]

Received: 11 July 2020 / Accepted: 28 September 2020

Abstract
Since Windows Vista, Microsoft has offered us a new life companion called SysMain or Superfetch from its old name. This is a service which analyzes and records the user daily software use to increase the speed of his or her experience on the operating system. However, this service provides the opportunity to track software used and private files seen such as movies or confidential files, reveal his or her lifetime activities and map directories. More than just a privacy issue, this constitutes a reliable approach in forensic analysis. Furthermore, this service is often misunderstood due to its little documentation and myths surrounding it, which makes things soon complicated to investigate. This paper is an extended version of the talk presented at Black Hat USA 2020: it aims at debunking partial and fake news about SysMain and its files. This paper will examine in detail its architecture, analyze its mechanisms and explain its operating method. It will detail the format of all the prefetch files which has been undocumented or obsolete so far. In addition, this paper will illustrate forensic concrete cases in which SysMain turns out to be useful.

Keywords Superfetch · SysMain · Prefetch

1 Introduction

1.1 Vocabulary and history

The notion of Prefetcher appeared in 2001 within the American brevet 6,633,968 [1], announcing the technique which will be a part of Windows XP. Under Windows Vista, another component called Superfetch is added to the algorithm and the service is renamed within the name of this improvement, until Windows 10. In this version, the service is renamed SysMain but Microsoft did not explain this change [2].

On Windows 10, Superfetch is only a part of SysMain, which is the name of the whole service, containing many parts including the Prefetcher and Superfetch. As the name was only changed with Windows 10, the whole algorithm is commonly, though erroneously called Superfetch.

✉ Mathilde Venault
venault@et.esiea.fr

Baptiste David
bdavid@et.esiea.fr

[1] Laboratoire de Virologie et de Cryptologie Opérationnelles, ESIEA, Laval, France

1.2 Goals and mechanisms

The main goal of the service is to increase the speed of the user experience. To this end, SysMain focuses on two aspects:

- Booting faster;
- Gaining time from the start-up to the closure of any process.

To boot as fast as possible, SysMain will frequently calculate the "optimal layout" which is the order of the file to launch in memory at the boot. This list is established during idle states: whenever CPU, disk and memory utilization are under a certain percentage of use, the service will process to non-urgent operations such as the optimal layout calculation. The result is written on `C:\Windows\Prefetch\Layout.ini` (Fig. 1).

On the other hand, increasing the navigation on applications is based on the mechanism of reducing page faults in memory, which is an optimization in memory paging.

Memory paging A process is a set of pages in memory, which are the same sized block of data containing the instructions of a program. Whenever a process is executed, these pages are mapped in theory into the physical memory (RAM)

Published online: 26 October 2020

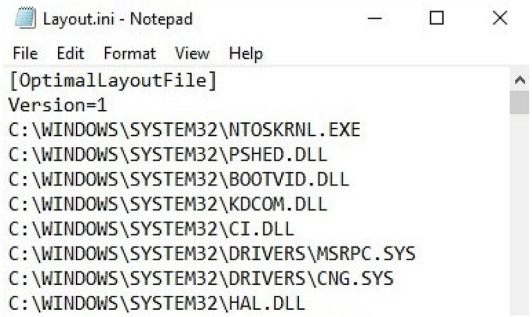


Fig. 1 Extract of Layout.ini

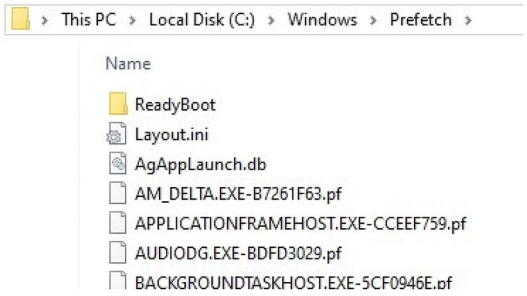


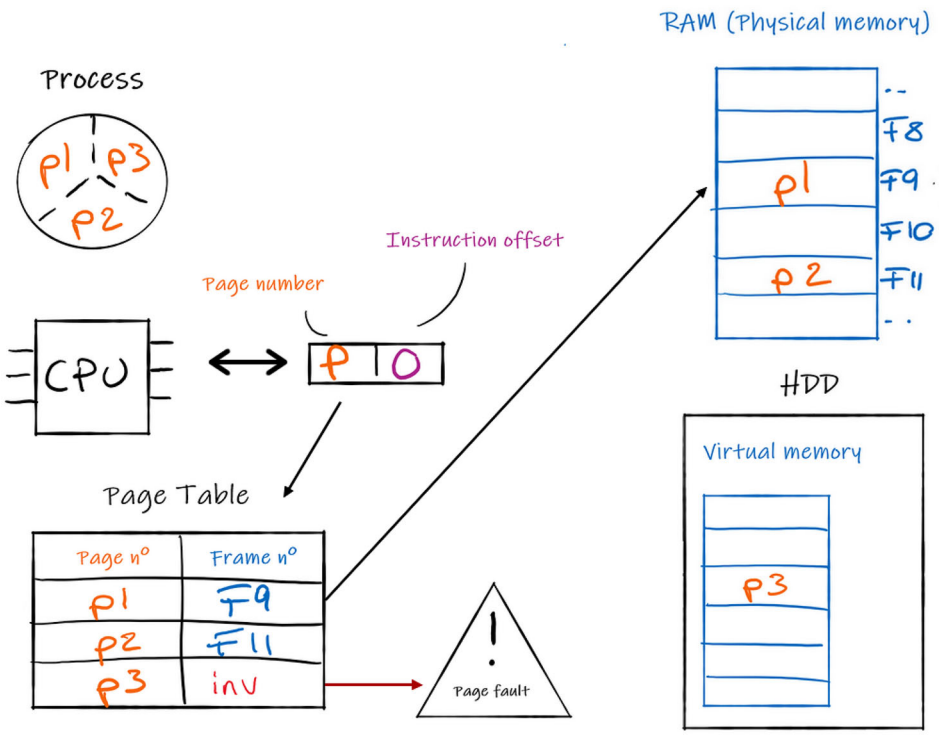
Fig. 3 View of Prefetch directory

within spaces called frames. To execute instructions, the CPU has to figure out where the instructions are with two available pieces of information: the page number and the instruction offset. To facilitate the operation, each process has its own Page table, associating page numbers and corresponding frame numbers. Therefore, when the CPU must find an instruction, it looks into the process' Page Table to get the frame number containing the page of the instruction. Then, the CPU is able to execute the instruction at the offset it had in the first place (Fig. 2).

Still, under special circumstances, a page could be mapped into the Virtual Memory, on the disk. When it happens, the page table could not resolve the frame number and indicates an invalid value to the CPU: a page fault occurs. The memory management unit, responsible for declaring the page fault, will then rely on the operating system to find the page in the virtual memory and will bring it back into a free frame of the Physical Memory, so the operation could be redone. Once the page is mapped into the physical memory, the page table is updated and resolves the frame number of the initial page number, just allocated. The CPU can now find the instruction.

This procedure requires time and memory operations thus reducing the program's reactivity. Superfetch aims at curtailing page faults to avoid this loss of time. For each program, it will log hard page faults occurred and it will record general page accesses so it could prelaunch in memory pages the user might need next time. Each process has one or more .pf file on the `C:\Windows\Prefetch` directory (Fig. 3) as a support for the next optimization.

Fig. 2 Memory paging mechanisms



1.3 Involvements

The mechanisms explained above imply that SysMain is watching and keeping traces of each action performed on a computer such as:

- Evidences of software installs;
- Dates and times of application launches;
- Number of executions per program;
- Names and locations of files used by each process;
- Links to cache files that might contain the content of personal text documents.

Therefore, SysMain knows a lot about you, from the time you woke up to your favorite songs. On the one hand, it raises a very serious privacy issue since it tracks your lifetime activities. Even though Superfetch does improve the speed of your experience on your OS, it raises the question of the limit between profiling and spying.

On the other hand, it is a significant forensic opportunity. Within a forensic analysis, it helps to analyze very precisely the activities, especially because a few people are even aware of Superfetch traces, so these traces are usually left on a computer. For instance, in malware analysis, the .pf files could prove the program's execution date and time and shows where were located the malicious files.

So, from a black hat point of view or from a white hat one, it does matter to know how SysMain works and what exactly its files could reveal.

1.4 Previous work, documentation and myths

Few studies have been done so far to document SysMain operation.The first issue is that some of those done focused on Windows versions older than Windows 10 such as the study Digital Forensic Analysis on Prefetch Files [3] and thus, are now obsolete. Regarding the format of the .pf files, documentation has been accessible and known approximately since 2010, and the most up-to-date is Joachim Metz's study [4]. Still, Superfetch has another kind of file ending with a ".db" extension and there are so far only two studies about: Rewolf's Blog [5] and Joachim Metz's study [6]. Despite these, the documentation is incomplete because Rewolf's Blog [5] covers precisely one file but sets aside the others which are very different and Joachim Metz's study [6] focuses on reporting observations more than concluding after a reversing engineering process.

Another widely covered aspect of SysMain has been explained: the hash function, notably [7,8]. Even though lots of these sources disagree on details such as the origin of the string to hash, they all converge on the same algorithm, presented on the section about the hash algorithm, in Sect. 5. Regarding the global operation, the most reliable source is the documentation from Windows Internals [9], covering the basics of the global operation. However, there are inaccuracies in it, reinforcing some myths already widespread. One of them is that SysMain could be disabled setting the registry value *EnablePrefetecher* to 0, within the key `HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\MemoryManagement`. Despite this method having been shared across the Internet and also written about on Windows Internals [9], this is not enough to stop SysMain nowawdays.

Indeed, whatever the value of this key is, SysMain will keep on writing its databases. This is observable checking the "last written time" of files on the`C:\Windows\Prefetch` directory after executing a program, with the value of *EnablePrefetcher* and *EnableSuperfetch* set to 0. While the service was supposed to stop, the prefetch files are still being updated. Regarding the registry value *EnableSuperfetch*, SysMain has a function called `PfSvSuperfetchCheckAndEnable`, which forces this value to 3 no matter the initial value. This proves that these registry values do not have any impact on SysMain's activity. For the record, disabling SysMain manually on the service control manager will solve this issue. To do so, the Service Manager must be opened, SysMain service selected and service properties accessed. The Startup type should be set to *Disabled* and the changes applied. SysMain will not track the user anymore upon the computer restart.

Part I

Global operation

This section will explain the key points of SysMain's operation. The components listed and detailed are not the exhaustive list of SysMain parts but aim at clarifying the key points. At first sight, the service could be seen as many divisions, handled by groups of functions identifiable by their prefix (Table 1).

Further, these groups are connected to each other in order to ensure the different types of tasks (Fig. 4). The major task is the *pf routines*: the non-stop jobs responsible for the essential functions such as processing traces of applications, predicting and pre-launching pages the user might need. They are also the parts communicating with the rest of the kernel: exchanges with the drivers, RPC (Remote Procedure Call) requests, logging and event parts, requests for information WNF (Windows Notification Facility) states, global system information.

Under special circumstances, SysMain can declare an *idle state* state to process actions that need time and memory operations, but do not require to be done every day. This is the reason why SysMain will check for power supply presence

Table 1 Function initials and their meaning

Prefix	Name
PfPr	Prefetch Processor
PfTr	Prefetch Trace
PfSi	Prefetch Section Info
PfHp	Prefetch Heap
PfCl	Prefetch Collector
PfDb	Prefetch Database
PfDi	Prefetch Device Info
Rdb	ReadyBoost
HbDrv	Hybrid Drive
AgAl	Agent Application Launch
AgGl	Agent Global
AgPd	Agent PFN Database
AgRp	Agent Robust Performance
AgTw	Agent Trace Writer

and utilization of CPU, disk and memory to be sure SysMain will not be harmful to the user's activities. In this free time, SysMain will update the optimal layout boot or execute the command `defrag.exe -s -b`: they are the *idle tasks*. For the actions that do not require doing frequently, SysMain has *periodic tasks*, based on action planned to synchronize registry keys values or remnant data. To ensure all these tasks, the work is divided per agent.

2 SysMain's agents

SysMain includes agents: they are components dedicated to a specific task. They are constantly watching for change and could be triggered anytime. They are loaded in order of importance, which is the following:

- Agent PFN (Page Frame Number);
- Agent Global;
- Agent Application Launch;
- Agent Context;
- Agent Robust Performance.

Agent page frame number The page frame number is an array representing each physical page state in memory on the system (Active / Standby / Freed), which will then be aware of page faults. The agent will be the direct interlocutor of the PFN, so it can relay the page faults or page accesses and classify the response. For instance, it classifies the memory page's origin: foreground or background application, or the state: committed page or not. This agent is the one in charge of getting the data from the memory, which will be the basis of future pre-launching.

Agent global This agent oversees the context for one user. It will define the criteria of Active Days, the limits of daily phases (which hours and days belong to work time/morning schedule), and it might organize *histories* successions of scenarios within a certain phase.

Agent application launch AgApl is involved throughout the prediction chain. First, post-processing the data received from a driver called FileInfo and concerning the files used by a process or by the agent PfnDb. In addition, it creates Markov chains to represent the probabilities of program uses. This constitutes the base for giving predictions. Given the calculated probabilities, the agent will take decisions and ask the memory manager to prelaunch certain pages.

Agent context This agent is responsible for watching the overall context:

- The current state of the computer (standby, hibernation);
- The current session information (SID);
- The current user information (user token).

Whenever a change occurs, it updates the current information so SysMain could be aware of the new situation and reacts if needed. For instance, if there was a user session switching, AgCx would take a snapshot of the current situation to be able to restore it faster if it is required later. There are two modes of disconnection:

- *Classic Disconnect*: properly quitting with the button and logging in on the other session.
- *Lazy Disconnect*: changing within clicking on "disconnecting". In this case, the agent will also go through Classic Disconnection mode.

Agent robust performance Basically, this agent oversees SysMain performance. According to Windows Internals [9], it *watches for specific file I/O access that might harm system by populating the standby lists with unneeded data*. It also checks the frequency of accesses to the files referenced by SysMain to avoid pre-launching irrelevant data such as the whole content of a file opened just once. Thanks to an internal threshold, AgRp prioritizes the files referenced to make sure the performance is at its best and processes regular checks to avoid keeping irrelevant data.

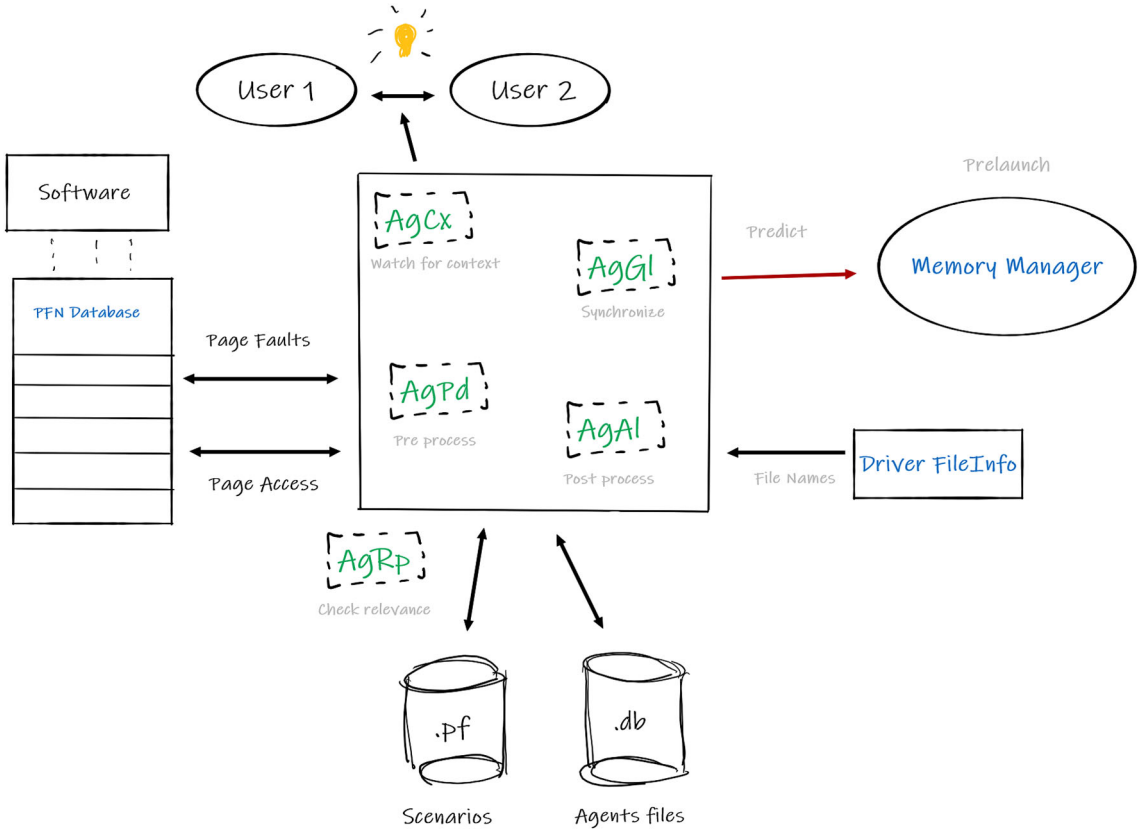


Fig. 4 SysMain global operation

3 SysMain's pillar: PfSvcGlobals

To facilitate communication from one function to another, SysMain uses global variables. *PfSvcGlobals* is the major global variable since the creation of the service. Unlike the other global variables, this one is initialized before the main thread worker and is by far the largest variable with 4 456 bytes.*PfSvcGlobals* contains values necessary to the proper functioning of the whole service including:

- Handles to its own heap;
- Handles to registry keys and registry values;
- Handles to process logging, process events or other external kernel communications;
- Time references for synchronization, task scheduling or time measurement;
- Current session information (SID, user information, state of computer's components);
- References to other global variables;
- References to important structures;
- Countless flags used everywhere.

It is important to understand that *PfSvcGlobals* is not "a big array": it is the pillar of all the components of the algorithm. *PfSvcGlobals* contains references to many other important structures such as the ones related to the agents or parts of the prefetcher (PfXp, PfSi, PfTr, PfCl, PfIu) which are needed for the global operation. In addition, many of its flags have an impact on the type of actions which have to be done in the pf routines.

4 Drivers connected to SysMain's activity

4.1 RdyBoost driver

There are two terms close to each other that might be confusing: *ReadyBoot* and *ReadyBoost*. ReadyBoost is the name of a driver, located at `C:\Windows\System32\Drivers\RdyBoost.sys` and which has a set of information on registry within the key `HKLM\SYSTEM\CurrentControlSet\Services\rdyboost`. ReadyBoot refers to one of the functionalities of SysMain to increase boot speed and has its proper directory within the `C:\Windows\Prefetch\ReadyBoot`. According to Windows Internals [9], ReadyBoost is "responsible for writing the cached data to the NVRAM device. When you insert a USB flash disk into a system, ReadyBoost looks at the device to determine its per-

formance characteristics and stores the results of its tests in `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVe`. Depending on these characteristics, ReadyBoost might dedicate a certain space of the disk for caching data and create a file `ReadyBoost.sfcache` in the root of the device. Data cached is compressed and encrypted per block using AES (Advanced Encryption Standard) encryption. It allows faster access to the data from the USB disk, contributing to SysMain's goals.

4.2 FileInfo driver

FileInfo is a mini-filter driver which gives to SysMain information about the files used by a process, located on `C:\Windows\System32\Drivers\FileInfo.sys`. FileInfo allows SysMain to get the names of the current memory pages processed and information about their origin. Since FileInfo plays an important role in file construction, at every launch SysMain starts the service and might even change the driver's start configuration, set to automatic start depending on its needs.

According to Windows Internals [10] *the prefetcher was supposed to execute transparently to other activities on a system but its file references can lead to sharing violations. FileInfo was used to watch for potential sharing violation collisions and prevent them by stalling a second operation on a file being accessed*. This explains the need for creating an independent driver and not include the component within the service.

In addition, since SysMain does not have the rights to get to information from ring 0, using a driver to do so was the easiest solution. Windows Internals [9] explains FileInfo driver *associates streams, identified by a unique key, currently implemented as the FsContext field of the respective file object, with file names so that the user-mode Superfetch service can identify the specific file stream and offset with which a page in the standby list belonging to a memory-mapped section is associated*.

In concrete terms, this driver tracks names and paths of the files used by a given process to build a buffer required to create the .pf files. To communicate, SysMain sends an IO Control code through the function `NtDeviceIoControlFile()` and gets back the buffer built by FileInfo Fig. 5 through the function parameters. The output buffer is written in the NL format, which includes path environment instead of directories full path. The buffer will be then translated under the watch of SysMain and post-processed to contribute to the formation of .pf files.

Once SysMain has the NL formatted buffer, the environment variable will be translated thanks to an internal "translation table" and the data regarding the page numbers and offsets will be stored into hash tables: the basis of the .pf file.

```
2d 00 0d 00 41 53 43 43-10 1c 13 00 01 00 00 00  -...ASCC........
03 00 00 00 00 00 00 00-00 00 00 00 18 00 00 00  ................
1c 15 00 00 00 00 00 00-42 03 00 00 97 10 17 23  ........B......#
01 00 00 00 00 00 00 00-3e 8b 1a 1c 90 65 d5 01  ........>....e..
30 e0 15 72 86 dc ff ff-3e 81 1c 2c 00 00 00 00  0..r....>..,....
2e 00 2e 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
5c 00 44 00 45 00 56 00-49 00 43 00 45 00 5c 00  \.D.E.V.I.C.E.\.
48 00 41 00 52 00 44 00-44 00 49 00 53 00 4b 00  H.A.R.D.D.I.S.K.
56 00 4f 00 4c 00 55 00-4d 00 45 00 32 00 00 00  V.O.L.U.M.E.2...
40 03 00 00 97 10 17 23-02 00 00 00 00 00 00 00  @......#........
c0 95 de 7e 08 c6 ff ff-30 e0 15 72 86 dc ff ff  ...~....0..r....
01 00 00 00 00 00 00 00-01 00 1a 00 5c 00 53 00  ............\.S.
59 00 53 00 54 00 45 00-4d 00 20 00 56 00 4f 00  Y.S.T.E.M. .V.O.
4c 00 55 00 4d 00 45 00-20 00 49 00 4e 00 46 00  L.U.M.E. .I.N.F.
4f 00 52 00 4d 00 41 00-54 00 49 00 4f 00 4e 00  O.R.M.A.T.I.O.N.
00 00 00 00 00 00 00 00-40 03 00 00 97 10 17 23  ........@......#
03 00 00 00 00 00 00 00-00 47 10 80 08 c6 ff ff  .........G......
30 e0 15 72 86 dc ff ff-01 00 00 00 00 00 00 00  0..r............
01 00 1c 00 5c 00 24 00-45 00 58 00 54 00 45 00  ....\.$.E.X.T.E.
4e 00 44 00 5c 00 24 00-52 00 4d 00 4d 00 45 00  N.D.\.$.R.M.M.E.
54 00 41 00 44 00 41 00-54 00 41 00 5c 00 24 00  T.A.D.A.T.A.\.$.
54 00 58 00 46 00 4c 00-4f 00 47 00 00 00 00 00  T.X.F.L.O.G.....
c0 01 00 00 97 10 17 23-04 00 00 00 00 00 00 00  .......#........
10 40 16 72 86 dc ff ff-30 e0 15 72 86 dc ff ff  .@.r....0..r....
01 00 00 00 00 00 00 00-01 00 05 00 5c 00 24 00  ............\.$.
4d 00 46 00 54 00 00 00-00 02 00 00 97 10 17 23  M.F.T..........#
05 00 00 00 00 00 00 00-20 6b 16 72 86 dc ff ff  ........ k.r....
30 e0 15 72 86 dc ff ff-01 00 00 00 00 00 00 00  0..r............
01 00 09 00 5c 00 24 00-4c 00 4f 00 47 00 46 00  ....\.$.L.O.G.F.
49 00 4c 00 45 00 00 00-80 03 00 00 97 10 17 23  I.L.E..........#
06 00 00 00 00 00 00 00-b0 4c 06 80 08 c6 ff ff  .........L......
30 e0 15 72 86 dc ff ff-01 00 00 00 00 00 00 00  0..r............
01 00 1f 00 5c 00 24 00-53 00 45 00 43 00 55 00  ....\.$.S.E.C.U.
52 00 45 00 3a 00 24 00-53 00 49 00 49 00 3a 00  R.E.:.$.S.I.I.:.
24 00 49 00 4e 00 44 00-45 00 58 00 5f 00 41 00  $.I.N.D.E.X._.A.
4c 00 4c 00 4f 00 43 00-41 00 54 00 49 00 4f 00  L.L.O.C.A.T.I.O.
4e 00 00 00 00 00 00 00-40 02 00 00 97 10 17 23  N.......@......#
```

Fig. 5 Buffer given by FileInfo to SysMain

5 SysMain hash algorithm

Whenever it comes to reverse SysMain, the hash process soon becomes familiar. SysMain has its own hash algorithm to serve two purposes:

- Building a part of the .pf files name;
- Making references on internal hash tables.

Under Windows 10 the hash algorithm is:

$$
\begin{aligned}
&Result = 314159;\\
&for(i = 0; i < len(StringToHash); i++)\\
&\qquad char = StringToHash[i];\\
&\qquad charmaj = RtlUpcaseUnicodeChar(char);\\
&\qquad Result = (Result \times 37 + charmaj) \times 37;
\end{aligned}
\tag{1}
$$

The initialization value used as a seed is the beginning of the pi decimals (3,14159). Superfetch has known several hash algorithms since its creation, but they all share the same basis. According to some studies [7,8], the following elements might have been changed on the previous versions:

- initialization value of the hash;
- multiplier coefficient;

- add of modulo operation.

This hash is far from being cryptographic: its operations are basic and easy to reverse. Since uppercase and lowercase characters will be processed the same way, two different input strings might have the same hashed output. Therefore, the algorithm is not second pre-image resistant nor collision resistant. Still, this hash algorithm does not need to be cryptographic because the strings hashed are used as references. There is no actual need to protect the input, which is not sensitive information.

6 Registry keys

More than 22 registry keys are frequently consulted, created, or deleted within routine protocols.The most important among them are:

HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\Memory Management\PrefetchParameters

- BaseTime
- BootId
- EnablePrefetcher *Despite countless sources claiming that setting this value to 0 disable SysMain, this is not the case.*
- EnableSuperfetch *This value is forced by SysMain to 3 on the function PfSvSuperfetchCheckAndEnable().*

HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Prefetcher

- BootFilesOptimized *Changed on PfXpUpdateOptimalLayout() just before the update of layout.ini.*
- LastDiskLayoutTime
- MinRelayoutHours
- LastDiskLayoutTimeString *Date and time of the last optimal layout update.*
- MaxPrefetchFiles *By default, 256.*

HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Superfetch

- PfPddata *This value is database buffer, read within a function called PfFgContextLoad(). Eventhough this is a registry, there is no doubt it is a database such as the one in the .db files, since it goes through the same checks as the buffers from .db files.*
- PfIuBatteryPaths *Path of the battery.*
- PfIuHistory *Compressed buffer of 5604 bytes containing file paths. The value is updated periodically within the function PfIuHistorySave().*
- LastResPriGenTime *Corresponds to a SystemFileTime of the last synchronization, value x transformed such as x shleft 64 shleft 23.*
- ResPriOptions
- StartedComponents
- Rebalancer Flags
- PfRbMinPagesToPrefetch;
- PfRbPrefetchStopTreshold
- PfRbPostBootDurationInMs
- PfRbPrioInversionThreshold
- PfRbImageScoreBoost
- PfRbLargeFileSizeInPages

\PfAp

- ApFetch_%SIDHashed *Compressed buffer.*
- ApLaunch_%SIDHashed *Compressed buffer with a fixed size which is actually a data containing full name of applications. This buffer is updated periodically.*
- UserTime_%ID *Compressed buffer related to the context for a given user. This buffer is updated periodically.*

\DiskAssessment

- DiskNumber
- LongSeekMicrosecondsBase
- LongSeekMicrosecondsPerSqrtGB
- PeakTransferMBsPerSecond
- RPM
- SeekBreakPages
- SizeInGb
- VolumeCreateTime
- VolumeSerialNumber
- PfRbSmallFileScoreBoost

\StaticConfig

- ProtectedProcesses
- ResPriHMImageListFilePath
- ResPriImageListFilePath
- Sku
- ServiceKeyPath
- MigratedServiceKey

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\OptimalLayout

- Enable auto layout *Whenever the operating detects an idle state, SysMain updates the optimal layout if the value is set to 1. Not always available.*
- LayoutFilePath *Full path of the optimal layout, by default:* `C:\Windows\Prefetch\Layout.ini`.

HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\EMDMgmt *This key is linked to the ReadyBoost job, EMD referring to External Memory Device, the working name of ReadyBoost during its development according to Windows Internals* [9].

- GroupPolicyDisallowCaches
- Attributes
- CacheSizeInMB
- DeviceStatus
- LastTestedTime
- ReadyBootTrainingCountSinceLastServicing *This value refers to the activation of ReadyBoost.*

Part II

Prefetch Files

SysMain has two major types of support files:

- The scenario files (.pf) relative to programs;
- The database files (.db or .7db) relative to the agents.

They both can be found on the `C:\Windows\Prefetch` directory, but the databases files are not always present in the Prefetch directory. Another common characteristic is the compression within the `XPRESS_HUFFMAN` algorithm and with the `RtlCompressBuffer()` function from `Ntroskrnl.lib`. The Table 2 details the databases files and their brief description.

7 Databases: the agent's support

The database files are the files ending with .db or .7db files. Since they are not always present on the directory, they are often forgotten and only a few documentations have so far been made. Their main goal is to keep traces of the Agent's work so the data collected could be remnant through OS reboots or context changes such as user switching or hibernation. Each agent of SysMain (AgCx, AgGl, AgAl, AgRp, AgPd) has one or more .db associated, with different names, since one agent might need more than one internal database for its operation. Their names and purposes have not been documented at all and unfortunately the few studies made about the format only partially documents it.

7.1 Databases construction

Databases files are traces of internal databases left during Superfetch activities. Each Agent has its type of database, but it is important to understand all the databases are connected to each other. Despite their different purposes, they are built on the same basis. Sysmain initialization functions build internal databases "from scratch", from different functions proper to the agent. Each initialization function defines either default parameters or specific parameters, then calls an underlying function and so on. This means they are all built on the same basis. Since they have the same basis, database files are read within the same process and have the same way of finding back the information. Here is the lecture process that aims at extracting useful information from the database file:

1. Get the view of the file when it is required, decompress the file if required and read the buffer;
2. Initialize a corresponding internal database with default parameters and defaults sizes;
3. Check file format conditions;
4. Fill in the information extracted from the file and adapts the internal database characteristics with the information.

Table 2 Database names detailed

Database name	Details
AgAppLaunch.db	Related to the applications; contains names of .exe.
AgRobust.db	Related to the performance of SysMain.
AgCx_%SID.snp.db	Related to user sessions; once per user. If there was a disconnection from one session to another, SysMain would take a snapshot of the previous session so it could be loaded faster if required later.
AgGlFaultHistory.db AgGlFgAppHistory.db AgGlGlobalHistory.db	Related to the AgPd Agent, covering data from PFN, respecively referencing to Page Faults, Foreground application and global accesses.
AGlUAD_P_%SID.db AgGlUAD_%SID.db	UAD might refer to User Active Days. Since SysMain includes the context within its prediction, could be related to the context.
dynrespri.7db cadrespri.7db	Referencing to Dynamic Reserved priority. They seem to be used as a basis to sort data and synchronize. Whenever the file is created, the registry value *LastResPriGenTime* is updated.

Once the step of linking the database file and the internal database is completed, the database is ready to be used elsewhere on the service.

7.2 Database format

Under Windows 10, the format of the compressed databases files is Table 3.

Whenever the file is decompressed, the common header format for the .db files is Table 4. Please note after this header the generalities are not possible to be made because of each database specificities.

Magic number The magic number have been seen for groups of databases but any connection could have been established. Table 5 shows the magic number seen and their associated database on one specific machine.

Sizes The minimum total size is 72 bytes, verified in common compliance function called `PfDbFileVerify Common()`. The header size follows the same rule, it must be higher than 72 bytes. Still, each database file might have its own criteria thus the required sizes could be higher in complementary compliance functions.

Table 3 Compressed .db file format

Offset	Content
0x00	0x4d4d41 MAM
0x04	Total size of the decompressed data
0x08	Checksum
0x10	Data compressed

Table 4 Uncompressed .db file format

Offset	Content
0x00	Magic Number
0x04	Total Size
0x08	Header Size
0x0C	FileType Param
0x10	Param 1
0x14	Param 2
0x18	Param 3
0x1C	Param 4
0x20	Param 5
0x24	Param 6
0x28	Param 7
0x34	Count of volumes
0x38	Count of paths registered
0x3C	Check condition verified after the lecture process to ensure integrity of the data
0x40	Condition to do specific lecture operations

Table 5 Magic number and associated .db

Magic number	.db associated seen
03	AgCs_%s.db AgGlFaultHistory.db AgGlFaultAppHistory.db AgGIUAD_P%s.db dynrespri.7db cadrespri.7db
05	PfPre_%sidhash.mkd AgAppLaunch.db
0F	AgRobust.db

Table 6 FileType parameters

FileType	Parameters
5	40h; 58h; 10h; 10h; 10h ; 10h; 0h ;0h
6	48h; 58h; 60h; 18h; 20h ; 10h; 10h ;0h
7	48h; 48h ;60h ;18h ;10h ;10h ;10h ; 0h
8	60h; 38h; 50h; 8; 8; 14h; 8; 0h
9	0h
A	60h; 38h; 50h; 8; 8; C; 8h; 0h
B	60h; 38h; 50h; 10h; 10h; 10h; 10h
C	60h; 38h; 50h; C; 08h; 08h; 08h
D	0h
E	48h; 70h; 90h; 10h; 10h; 10h; 10h
F	68h; 40h; 50h; 8h; 8h; 14h; 8h
10	60h; 40h; 88h; 10h; 18h; 8h; 8h
11	0h
12	50h; 50h; 58h; 18h; 10h; 10h; 10h
13	60h; 38h; 50h; 8h; 8h: 8h; 8h
14	60h; 40h; 58h; 10h; 8h; 8h; 8h
15	60h; 50h; 58h; 10h; 18h; 8h; 8h
16	60h; 40h; 50h; 8h; 8h; 8h; 8h

Filetype parameters The FileType number indicates the index in an internal array relative to database sizes and offset calculation on the file (Table 6). The array is a 9 DWORD table long, declared within the name `PfDbDatabaseParamsForFileType`. If there are reading problems, SysMain, thanks to the index indicated by the filetype number, knows where to find the parameters.

The spread of the different FileType could be explained by retro compatibility. Under Windows 10, the values tend to the last values, then the previous versions might have previous numbers and some of them remained and some others evaluated.

Explanation of the parameters

- Parameter 1: it is useful to calculate the offset of the volume path, with the following sum: Offset of volume path = End of header + Param 1.
- Parameter 2: it is useful to calculate the offset of the second string, with the following sum: Offset of string 2 = End of volume path + Param 2.
- Parameter 3: it is useful for offset calculation as recurrent patterns. Unlike the Parameter 1 and 2, it will be used in an offset calculation loop.
- Parameter 4, 5, 6 and 7: Size parameters for internal database.

8 Scenarios: the traces of the user's activities

Scenarios are the supports for Superfetch to log what happened during a program's execution and improve future predictions. An application has one or more scenario files attributed depending on the way it has been executed. The name is always composed of "NameoftheApp - HASH.pf", the hash referring to the command line that allowed the execution hashed. The highest number of scenarios files within the Prefetch directory and their size are fixed by the value of the registry key: `SOFTWARE\Microsoft\Windows NT\CurrentVersion\Prefetcher`. By default, the scenario maximum number is 256 and the maximum size is 10 485 760 bytes. Whenever a process starts up, the scenario is immediately created or updated, referencing the page accesses and the page faults that occurred to avoid them at the next launch. Given the references of these pages, the next time, the process will be started, SysMain determines whether or not it is a prelaunchable application or not, calculates the entry threshold, compares it to an internal threshold so it could be sure it is worth to prelaunch the page and does so.

8.1 Scenario construction

Whenever a program is launched, SysMain always follows the same logic to build the corresponding scenario:

1. Get what the scenario name would be;
2. Retrieve the file used by the process thanks to the minifilter driver FileInfo;
3. Initialize a "scenario info" buffer, containing the basic header such as process information, the current date and time execution;
4. Only if the scenario file already exists:
   a Open the existing file;
   b Decompress and gets the older buffer;
   c Verify the buffer format and basic conditions. If there is a single problem, the file is deleted.
   d Copy the obsolete content on the new buffer, select and update information.
5. Compress the buffer, add a header and write it into the file.

8.2 Scenario format

Under Windows 10, the format of the compressed scenario is Table 7.

Whenever the scenario is decompressed, the header format is Table 8.

8.3 Scenario content

The scenarios are the tip of the SysMain iceberg: they contain the result of a long process of the data generated by the user experience within a certain context, and the condition to achieve what SysMain has been created for: prelaunching what is required by a user when the time comes. Once the buffer is decompressed, the last part of the file contains full paths of files used by the applications. The majority are internal files required for the global operation of the application: DLLs (Dynamic Link Libraries), dependencies, or any files with an extension specific to the application. Still, there are

Table 7 Compressed .pf file format

Offset	Content
0x00	0x4d4d41 MAM
0x04	Total size of the decompressed data
0x08	Data compressed

Table 8 Uncompressed .pf file format

Offset	Content
0x00	Operating system identifier
0x04	SCCA: prefetch signature
0x08	11 41 43 43: format condition
0x0C	Total size of the file
0x10	Program name
0x4C	Hash Value
0x50	RESERVED
0x58	Count of paths registered
0x64	Offset of paths block
0x6C	Offset of volume block
0x64	Size of volume block
0x80	8 last dates and times of execution
0xC8	Count of executions

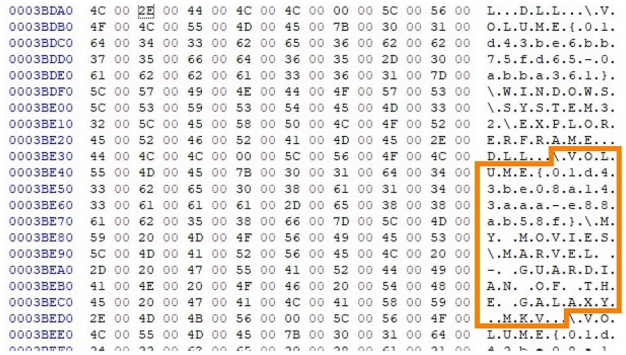


Fig. 6 Extract of VLC.EXE-5A3EF7FA.pf (decompressed)

also recent files or often used files (Fig. 6). Thus, the scenario of photo editors contains the name of the last photos opened, the scenario of media players contains the names of songs listened to or the title of the last movie seen but also dates and hours of these actions. Combining the list of those, SysMains allows to determine habits and hobbies of the user.

In addition, SysMain goes further, storing references to the cache files. The cache files are the result of the Cache Manager performance, which stores temporarily data to reduce the access time next time the data is required. Within the `UserDirectory\AppData\Local\Microsoft\Windows\Cache`, there are parts of files accessed remaining from previous use. The Cache is designed to do stream caching, which implies the data stored could be also parts of files, in clear. Therefore, SysMain referencing the cache files gives the possibility to view clear text data from the user applications, including documents editor containing private data.

9 Forensic uses and opportunities

Superfetch resources turn out to be useful for many forensic situations. Finding traces of executed programs provides the opportunity to determine which activities have been done on a computer. In addition, the details given by the scenarios allow to understand the context of the program use, thanks to the hours of the activity or the amount of time it has been used. Thus, it is possible to find some patterns in the user activities or establish connections between actions. This part aims at showing some of the potentials clues that may be found in the scenarios. All of the operations can be achieved with the tool called SysMain View, available on Github at *MathildeVenault/SysMainView*. This tool allows to compress, decompress, edit, and view information on databases and scenarios files. This section details some circumstances where SysMain traces might contribute to forensic analysis.

9.1 Malware analysis

As part of malware analysis, SysMain could be useful to trace the history of events on a computer and determine the circumstances of the attack. This may help to figure the exact time of the attack and to identify the malicious payload.

Methods It would be possible to to visit the prefetch files, examine the program traces at the estimated period of the attack and look for something suspicious. If the malware is an executable, its scenario would have been created: look for prefetch files with unfamiliar names and recent activity. If the malware is in the form of a script, it would have been executed through a shell: search into the command-line interpreter scenarios the names of the scripts they have executed, name and the location of the payload might be found there. If the script has not been removed, it might remain on the system and it might allow a reverse engineer process of the malware.

The scenarios could also help to understand the vector of the infection. If it is a phishing attack, traces might be left in Word office programs scenarios or pdf files readers: look for anything suspicious in the recent files opened. In this way, this might complete a forensic investigation in web browsers.

Example On an incident response mission, a computer from an employee has been infected by a malware which deletes files on the system and when the victim found out, he immediately turned off his computer. The goal: find out what happened. The first look at the recent activities reveals that PowerShell has been active at 4:23 pm, whereas the owner is not supposed to use this kind of application. Another scenario demonstrates PowerShell was executed less than 5 times but unfortunately does not list any suspicious file.

Investigating further on the recent activities shows that the victim has opened Word for the last at 4:21 pm, and among the files related to Word's activity there is one from a USB device which is: `D:\MYUSB\NovemberSchedule.docx`. Afterwards, the victime precise he was reading this document from a USB key while all of his professional documents were disappearing from his desktop. The device is retrieved and the document opened in a sandbox: there is VBA macro executing directly on PowerShell the command deleting all the content of the user documents.

9.2 Suspicion of illegitimate activity

The prefetch files play an important role in the context of criminal investigations since they are a means to track any suspect as long as there is an access to his or her computer. Indeed, they provide lots of information on the user personal

files and give concrete evidence of certain activities, which are essential for forensic purposes.

Methods The first step in this kind of forensic investigation involves profiling the owner of the machine. It includes determining what the common use of the machine is: what the context of use is (professional or private), which programs are used the most, when the usual activities are realized, and so on. To this end, it is interesting to observe for each program how many times they have been executed and the last dates and times of execution. The hours of program execution could give the exact schedule of the user. For instance, if Microsoft Teams and any mailbox application are opened every morning between 8 am and 10 am except the Saturday and Sunday it is possible to assume the computer is used for professional purposes. Similarly, if a movie media player or video games applications are opened each evening, the machine is likely a personal computer.

Profiling could be expended also thanks to the names of the files used by each process, recorded in the scenarios. This tells a lot about user activities, especially text editors or media players. This kind of program reveals the type of documents the user has and exposes his or her hobbies and preferences. Given the movies and songs recorded on media players, the type of music and movies he or she likes could be guessed. This applies to lots of other things such as text documents, which might reveal favorite readings or even wider information, such as works the user is working on lately. Sometimes, the files recorded lead to other important information such as events or intents. If the user has lately written a CV, it is possible to speculate he or she might think about getting a new job. If the user has seen his or her last holiday pictures, then the name of the folder would be recorded, and it might contain details about the date or the destination.

In addition to profile someone, the scenarios constitute concrete evidence of the user's activity on the machine. Indeed, whether the file is still stored on the computer memory or not, the record on the scenarios remains. Looking into the prefetch files allows listing the files that have been opened on the computer, even if they are now removed or if they have been consulted from an external device. Irrespective of whether the file is still on the disk, the record remains since SysMain references the original path of the file at the moment the file was consulted. Grouping the different directories and files registered can thus lead to a precise mapping of the computer, in the past.

The same goes for the applications: SysMain keeps the scenario of a program uninstalled for a certain time afterwards. The maximum duration depends on the global use of the computer; SysMain will remove the corresponding scenario once it figures out the file is not useful anymore, thanks to its Agent Robust Performance. In the case of criminal acts, the traces are often destroyed or at least hidden as much as possible and SysMain is one of the rare means to get precise and reliable information.

Example A company just fired an employee for professional misconduct. The company suspects illegal activities on the professional machine. How could it be possible to find evidence of such acts?

The first approach is to look for suspicious programs, that could have been used for illegal activities, and among the prefetch files, one catches attention: a scenario proving the use of tor browser, one of the means to access to the dark web. This shows that tor has been executed more than fifty times and among the last execution dates and times, the majority indicate the execution at night. Looking into other scenarios reveals a list of more than 30 files named "Passport" with identification numbers and last names in the scenario of a photo editor.

9.3 Warning: scenario falsification

It is important to keep in mind that Superfetch traces might be falsified. Indeed, any hacker could remove prefetch files or edit the information that could betray what has been done on the computer. For the record, it is possible to save the scenario corresponding to the program that is about to been used and once the action that has to be hidden is done, replace the legitimate scenario with the scenario backed-up before. Similarly, it is possible to edit sensitive information such as the dates and the count of executions, directly on the legitimate scenario. Windows will not notice the change and process the scenario, without the traces of what has been done. In a forensic investigation whilst falsification is unlikely to have been done, it is still a possibility. SysMain traces are useful but not infallible.

Part III

Remarks and Conclusions

9.4 Weaknesses of SysMain

The biggest weaknesses of SysMain are due to the need for retro compatibility. Among its 2500 functions, some of them show deficiencies when functions need to adapt to old OS characteristics. For instance, the process of loading agents. Whenever it comes to load its agents, SysMain has two ways of processing:

- `PfSvLoadDefaultAgents()` which is an association of basis load;

- `PfPrAgentsLoadFromRegistry()` which is based on references on the registry.

The function `PfPrAgentsLoadFromRegistry` plays an important role. First, it opens the following key: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Superfetch`. It gets all the values on the `Agents` hive, separating multiple names within the same value name if they are many of them with the `RegEnumValueA()` (Fig. 7).

SysMain has now strings association, which are actually library names (argument LibraryName) and function names of the specified library (argument ProcName). The function `PfPrAgentLoad()` will be called for each association.

Therefore, as soon as the registry value is edited, it is possible to indicate to Sysmain a specific dynamic library and any function of this dll, which will be loaded and executed without any further checks. In addition, it is also possible to execute multiple functions this way since the `PfPrAgentLoad()` function is called once per value on the `\Agents` hive. The SysMain process of agents loading is not explicit enough about characteristics of the DLL to be loaded and it is thus possible to load a malicious DLL: this is *DLL side-loading*. Even though this is not a DLL Hijacking such as explained on Blog [11], it was important enough to report the weakness to Microsoft. This weakness is likely due to retro compatibility because this function does not seem to be used anymore under Windows 10.

9.5 Conclusion

SysMain has been misunderstood for a while, whereas it plays an important role within Windows' daily performance. Reversing was challenging: SysMain has lots of functions belonging to lots of different notions and the backward compatibility needs make its architecture, which is already complex by design, even wider. Another major point was evaluating the consequences of SysMain's job. The privacy issue is undeniable. As long as SysMain is enabled on a computer, it is possible to track the user and exploit the information. This paper aimed at clarifying SysMain's operation to give the means to anyone to make this decision in full knowledge of the facts. It also aimed at illustrating forensic concrete cases in which SysMain turns out to be useful and at showing how the tool presented can help in these kind of forensic analysis.

```
ErrCode = RegEnumValueA(
            RegistryAgents_,
            i,
            (LPSTR)&ProcName,
            &cchValueName,
            0i64,
            &Type,
            (LPBYTE)&LibraryName,
            &cbData);
ErrCodecp_ = ErrCode;
```

Fig. 7 Part 1 of `PfPrAgentsLoadFromRegistry()` function

9.6 Limitations

This study represents a solid base to understand SysMain on Windows 10. Nevertheless, SysMain is intended to evolve, and it might change dramatically with the next Windows version or at any upgrade. Therefore, what is documented now could be anytime obsolete or upgraded. Also, access to Prefetch directory requires an administrator's privilege. From a black hat point of view, if the hacker does not have administrator access, the process becomes complicated and the knowledge about SysMain might be useless.

9.7 Future work

Even though the study is a solid base covering all of the essential aspects of the global operation, it would be interesting to explore further on the external components of SysMain's performance such as the drivers. Indeed, the drivers related to SysMain's activity have lots of functions unused by SysMain. What are their uses? Do they have other functionalities? In addition, SysMain's performance is closely tied to the memory manager, including the cache. As the cache management is not widely documented, it would be rewarding to understand its mechanisms and its exact links with SysMain.

References

1. Zwiegincew, A., Walsh, J.E.: Prefetching of pages prior to a hard page fault sequence. U.S. Patent (2001)
2. suat.cini: Superfetch service has been promoted to sysmain. congratulations! [Online]. Available: https://answers.microsoft.com/en-us/insider/forum/all/superfetch-service-has-been-promoted-to-sysmain/395cd8b7-7a02-44fa-af91-dd6b358b7276. Accessed 07 2018
3. Shashidhar, N.K., Novak, D.: Digital forensic analysis on prefetch files. Int. J. Inf. Secur. Sci. **4**(2), 39–49 (2015). https://pdfs.semanticscholar.org/2e5e/bffd41661a4ca85420be881f70b2162a4638.pdf
4. Metz, J.: Superfetch databases [Online]. Available: https://github.com/libyal/libscca/blob/master/documentation/Windows%20Prefetch%20File%20%28PF%29%20format.asciidocc. Accessed 02 2020
5. Blog, R.: Windows superfetch file format-partial specification [Online]. Available: http://blog.rewolf.pl/blog/?p=214. Accessed 10 2011
6. Metz, J.: Superfetch databases [Online]. Available: https://github.com/libyal/libagdb/blob/master/documentation/Windows%20SuperFetch%20(DB)%20format.asciidoc. Accessed 04 2014
7. Blog, H.: Prefetch hash calculator [Online]. Available: http://www.hexacorn.com/blog/2012/06/13/prefetch-hash-calculator-a-hash-lookup-table-xpvistaw7w2k3w2k8. Accessed 06 2012

8. Hiddenillusion Blog: Go prefetch yourself [Online]. Available: https://hiddenillusion.github.io/2016/05/10/go-prefetch-yourself/. Accessed 05 2016
9. Yosifovich, S., Russinovich, M.E., Ionescu, A.: Windows Internals, Part 2 (6th edn.). Microsoft Press, Redmond, Washington (2017)
10. Margosis, A., Rusisnovich, M.: Windows Sysinternal Administrator's Reference. Microsoft Press, Redmond, Washington (2011)
11. Blog, I.: Windows dll hijacking (hopefully) clarified [Online]. Available: https://itm4n.github.io/windows-dll-hijacking-clarified. Accessed 04 2020