# UsnJrnl Parsing for File System History Project Report

Frank Uijtewaal, Jeroen van Prooijen
University of Amsterdam

February 7, 2016

## Abstract

The UsnJrnl is a journal file used in Microsoft's New Technology File System (NTFS). The UsnJrnl has been known for its forensic value for some time now. The UsnJrnl can be combined with other artefacts, which gives a better overview of what happened to a system. At this moment combining these artefacts is laborious work as existing tools don't give a combined overview of the artefacts' data. This research aimed to determine what information the UsnJrnl holds and to find an effective way of linking it with the other files automatically.

The artefacts have been thoroughly examined to pinpoint those fields that can be reliably used as links. These links are combined into a model. This model could be used to create an automated analysis tool, as has become evident from the proof of concept. This proof of concept is open source.

A final conclusion can be made that the UsnJrnl on itself holds concise information which is a good starting point of an investigation.

## Introduction

The UsnJrnl, sometimes also referred to as the "Change Journal", records all changes to the filesystem. It's part of newer versions of NTFS and used by default in Windows Vista and higher.

Besides its use in filesystem management, it has received a growing interest in the IT forensics community as it holds valuable information regarding what happened on a (Windows) system. However, existing forensics tools don't implement means to efficiently use this artefact, making effective use of it tedious work.

## Research question

How can the artefacts found in the UsnJrnl be effectively used in forensic research?

## Related/prior work

Earlier research has pointed out that the UsnJrnl can be validly used for forensic purposes, for example the detection of InPrivate browsing and the use of tools such as CCleaner [1]. In

combination with the NTFS LogFile the UsnJrnl can give a better understanding of which operations have occurred on a file [2]. In addition the UsnJrnl can be seen as part of a trinity. This trinity, which comprises the UsnJrnl, LogFile and Master File Table (MFT) can show in great detail what happened to a system [3] [4] [5].

Knowledge considering the LogFile is more based on experimenting and reverse engineering than it is on academic research or official Microsoft documentation as very few of it is publicly available.

# Background

This section gives a short introduction, or refreshment, into the UsnJrnl and its related files, as some knowledge is required to understand the research. However, this paper is not supposed to describe in detail how to parse the artefacts. Therefore, this section only discusses what is necessary to understand the model. More detailed information on the artefacts' structure can be found in [6] [7] [4].

## MFT

The MFT has been extensively investigated in prior research and is well documented [6] [8] [9]. All files on the file system have their own entry (or entries) in the MFT. These entries contain information on the file's name, its modification date, etc. An MFT entry (sometimes also called "file record") starts with a header which contains basic information on how the entry should be interpreted. What follows is a list of so-called attributes. The most common are the "Standard Information" and "File Name" attributes. A full list can be found at [7].

Every attribute has a set of fields. The fields that play an important role in this research are described briefly below. The structure of an MFT entry starts with a header, see table 1.

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>a</th><th>b</th><th>c</th><th>d</th><th>e</th><th>f</th></tr>
<tr><td>00</td><td colspan="4">signature</td><td colspan="2">offset to fixup array</td><td colspan="2"># entries in offset array</td><td colspan="8">logfile sequence number</td></tr>
<tr><td>10</td><td colspan="2">sequence value</td><td colspan="2">link count</td><td colspan="2">offset to first attribute</td><td colspan="2">flags</td><td colspan="4">used size of mft entry</td><td colspan="4">allocated size of mft entry</td></tr>
<tr><td>20</td><td colspan="8">file refence to base record</td><td colspan="8">next attribute id</td></tr>
<tr><td>30<br>⋮</td><td colspan="16">Attributes</td></tr>
</table>

Table 1: Schematic overview of an MFT entry

The MFT header contains a field named `logfile sequence number`, whose value points to the most recent LogFile entry that contains historic data regarding this MFT entry. Another important field is the `sequence value`, which points out how many times this MFT entry has been used to describe a file. If a file is deleted and later on a new file is created re-using this location in the MFT, the sequence value is incremented by one.

The MFT entry header is followed by two or more attributes. Each attribute has its own data structure and starts again with a header. The "Standard Information" attribute has a field named `usn` (Update Sequence Number) which points to the most recent UsnJrnl record regarding this MFT entry. The "File Name" attribute carries the file name of the MFT entry, but it also has the MFT entry number and sequence value of the parent file. In most cases this is the directory the file is located in. In general every file or folder that that lives on an NFTS

file system has an entry in the MFT. Therefor extracting the MFT from a forensic image already gives the researcher an overview of the files on disk.

## Logfile

The third entry, with index 2, of the MFT is reserved for the LogFile. The LogFile is a journal file which can be of value in forensic investigations. Most of the community's knowledge about the LogFile and its structure has been gathered by reverse engineering [8] [4] [5] [10] [11]. Even so, most of these references don't have proper documentation, making verification of the assumptions that were made difficult. Despite all efforts, some fields and internals of the Logfile are still unknown [2], making automatic parsing and interpreting of the Logfile harder than is desirable.

The LogFile's normal function is to help in case of a system failure [8]. NTFS can use this file to bring the disk back up roughly to the moment the failure occurred. Also, as it is circular, the oldest entries are constantly being overwritten by new entries. The LogFile's size can be adjusted.

A basic log record page has a size of 4KB. In general each page has multiple entries which are all part of a larger whole - a transaction. These entries go by different names: in order to avoid confusion with `record pages` this paper uses the term `LSN entry` or `LSN record`. A file system operation, like creating a file, results in the creation of multiple LSN entries. While those all belong to the same transaction, LSN records in such a single transaction often reference multiple MFT entries. An overview of a transaction can be seen in figure 1.

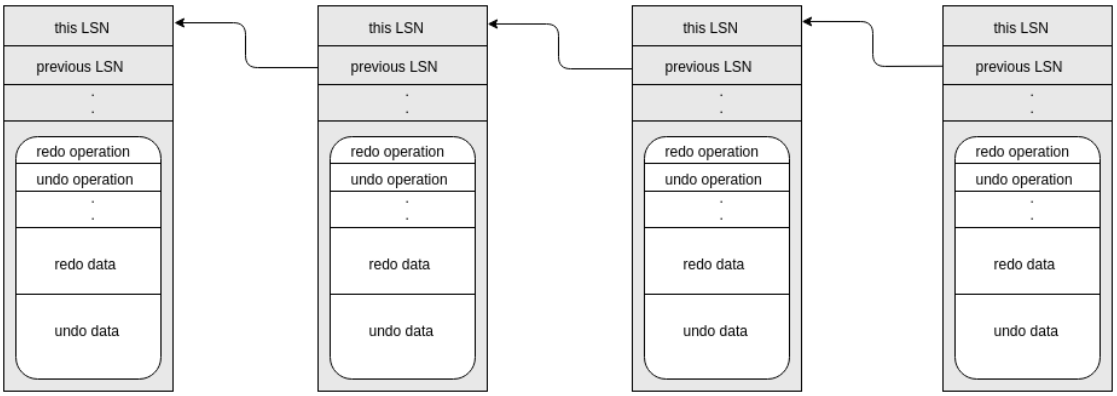


Figure 1: Transaction overview. While the embedded white part may create the impression that an LSN record has a layered structure, it does not. It's white to point out that this part of the LSN record is variable and of most use to forensic applications.

Table 2 describes the structure of a single LSN entry. Important in a single LSN entry are the `this lsn` and `previous lsn` fields.

These fields allow the LSN records to be grouped together into the transaction they belong to and at the same time order them, which is essential in case a failure happens and some filesystem operations need to be repaired or repeated. Together with the actual disk instruction fields, called `redo operation` and `undo operation`, they tell what changes occurred on the disk's files [2].

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | this lsn | | | | | | | | previous lsn | | | | | | | |
| 10 | undo next lsn | | | | | | | | data length | | | | seq number | | client index | |
| 20 | record type | | | | transaction id | | | | flag | | reserved | | | | | |
| 30 | redo operation | | undo operation | | redo offset | | redo length | | undo offset | | undo length | | target attribute | | lcn's to follow | |
| 40 | record offset | | attribute offset | | mft cluster index | | alignment / reserved1 | | target vcn | | | | alignment / reserved2 | | | |
| 50 | target lcn | | | | alignment / reserved3 | | | | redo and/or undo data | | | | | | | |
| 60 ⋮ | (continuation redo and/or undo data) | | | | | | | | | | | | | | | |

Table 2: LSN record

Depending on the redo operation and undo operation some extra data may be embedded in an LSN entry. The length and content of this data varies per variation of the redo/undo pair. The data can hold forensically valuable information, such as USN records or complete MFT entries.

Earlier research[5] pointed out that in the particular case of a redo/undo operation being "Update Nonresident Value/No operation", an embedded embedded Update Sequence Number (USN) record can be found in the `redo data` field. Such a USN record is a record of the UsnJrnl.

## UsnJrnl

Some say that the UsnJrnl is a user-readable equivalent of the LogFile [8]. The UsnJrnl was introduced in Windows 2000 but it was normally not used until Windows Vista came along, from which point on it has been turned on by default [12] [13]. In contrast to the Logfile, Microsoft has documented the UsnJrnl fairly well [14] [15]. It is one of the files in the $Extend directory and the location on disk can be found by looking at the $J Data stream, which makes the path `\$Extend\$UsnJrnl:$J` [6].

The Usnjrnl consist solely of *USN records*. Each USN record describes, on a higher level than the Logfile does, what happened to a file (or directory, all the same). According to Microsoft's documentation there are three different record versions used in different Windows operating systems, namely USN_RECORD_V2 [14], USN_RECORD_V3 [16] and USN_RECORD_V4 [17]. The difference between versions 2 and 3 are minor; version 4 is not yet documented. Microsoft points out that version three can be found on Windows 8 and Windows Server 2012 and up, but tests on both Windows 8.1 and Windows 10 show that only version 2 is used still.

The UsnJrnl is not circular, which means that as it runs out of space new blocks are allocated. Older blocks are at some point deallocated [18]. If these deallocated blocks have not yet been overwritten, one could try and carve these USN records from disk.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | record length | | | | major version | | minor version | | file reference number | | | | | | | |
| 10 | parent file reference number | | | | | | | | usn | | | | | | | |
| 20 | timestamp | | | | | | | | reason | | | | source info | | | |
| 30 | security info | | | | file attributes | | | | file length | | file name offset | | name | | | |
| 40 ⋮ | (name continuation) | | | | | | | | | | | | | | | |

Table 3: USN record

Table 3 shows the structure of a UsnJrnl record. The reference numbers use the same format as is used in the MFT's "File Name" attribute [6] and show what MFT entry the record is about. The name shows the filename (not the full path) and the USN acts as an unique identifier for this record. Each USN record has a `reason` field, which described what happened to the file to cause this record to be created.

Forensically the UsnJrnl holds valuable information as it can give historic information for NTFS. For example, it can uncover some information on deleted files, or show that files were renamed, opened or moved at a certain point.

# Model

We propose a model that can be used to combine the three separate NTFS artefacts (MFT, Logfile and UsnJrnl) that were discussed earlier. The model is shown in figure 2.

## Forensic value

The observation that the LogFile, MFT and UsnJrnl contain forensically valuable information is not new. Also, both the MFT and the UsnJrnl have been fairly well understood for some time now. What does the model bring that is new and valuable? It's *how* and *when* the artefacts can be systematically and thus automatically linked. Linking the artefacts gives knowledge that would probably not have been gained by examining the artefacts separately. To give some examples, linking the three artefacts:

- Allows precise timelining of LogFile entries (transactions).
- May give a good overview of what files have been deleted, including as much metadata as is available on disk.
- Broadens the field for researching automatic anti-forensics detection, as one can look for inconsistencies transcending the individual artefacts.

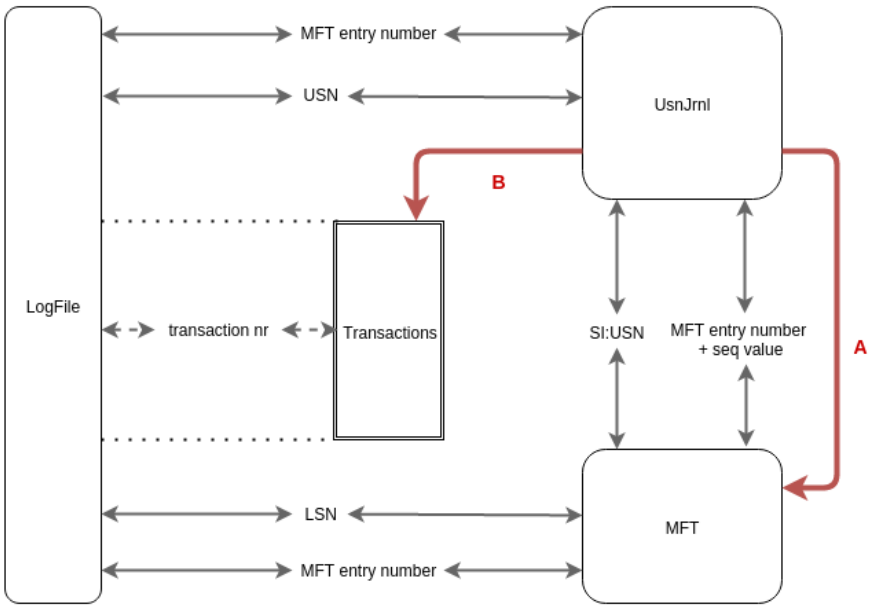


Figure 2: Correlation model. The arrows show what field or value connects two files together. Each of them is discussed briefly below.

## Red arrows

The red arrows show how the model is actually used: the UsnJrnl acts as an entrypoint and the MFT and the transactions are consulted based on information that is already found in the UsnJrnl. *Arrow A* makes use of primarily the `MFT entry number + seq value` link. *Arrow B* provides most of the historic data as the LogFile, and therefor the transactions, is by far the most verbose logging subsystem on NTFS. By having the transactions reconstructed it's easy to immediately find all log data available for just a single change on a file.

The sections below describe the relations drawn in figure 2.

## LogFile ⇆ UsnJrnl

*MFT entry number* - A USN record always contains a MFT entry number (derived from the `file reference` field). This value can be used to connect a USN record to certain LSN records: for these records it is possible to "calculate" the MFT entry to which the LSN record applies.

The calculation uses the `target Virtual Cluster Number (VCN)` and `MFT cluster index` fields in an LSN record. The exact formula is shown in equation 1 at the end of this paper.

*USN* - Each USN record has its own unique identifier called Update Sequence Number (USN). Some LSN records have an embedded USN record in their redo data field. If so then the corresponding USN of that record makes a connection possible between the two journal files.

### LogFile ⇆ Transactions

*Transaction nr* - The LogFile's LSN records can be grouped by the transaction that they belong to, basically chaining them together. Each transaction is given a unique number. This number is the `this lsn` value taken from the second to last LSN record in a transaction. This number seems to be referenced from an MFT entry's header quite often, so this seems a good design choice.

### LogFile ⇆ MFT

*LSN* - Every entry of the LogFile has its own LogFile Sequence Number (LSN). Every MFT entry references the most recent LogFile record that contains log data about it. As the LogFile is circular, however, which causes old records to be overwritten, this link may not always lead to an actual LSN record. The same "problem" applies to the opposite direction.

*MFT entry number* - The LogFile and MFT can be linked by means of the MFT entry number. How this is calculated and in what cases this applies is discussed in "LogFile ⇆ UsnJrnl". Even if finding an MFT entry is successful, this doesn't imply that a valid link can be established. The reason why is that the LogFile may very well contain LSN's that apply to MFT entries that are long gone. One should be cautious when using this link as there's no easy way to differentiate between different MFT entry occupations.

### MFT ⇆ UsnJrnl

*SI USN* - The "Standard Information" attribute, which is usually present in an MFT entry, has a USN field. This USN corresponds to a record in the UsnJrnl. Following the line from the MFT to the UsnJrnl this method will usually be fruitful because the UsnJrnl may hold quite some historic data. Going the other way around, though, relying on this correspondence will not always pay off. This can simply be caused by newer USN records superseding older ones, overwriting the SI:USN in the MFT, but also when a file is deleted and the MFT entry overwritten.

*MFT entry number + seq value* - The UsnJrnl entries give us an MFT entry number and MFT sequence value. The first one points to a specific MFT entry and the second one gives information about the number of times this MFT entry has been occupied. The sequence value in a USN record can be lower than the one contained in the corresponding MFT entry. This happens if the MFT entry is reused. In effect the USN record now holds historic data for this MFT entry, that would not have been possible to retrieve otherwise.

## Experiment

### Tools

The model was verified with the help of a couple of tools. Instead of relying too much on existing tools, which are often either proprietary or hard to extend, some new tools were created. In their current state they allow complete parsing of the UsnJrnl and partial parsing of both the MFT and LogFile. also, they were created with extensibility in mind so that it's possible to combine the parsed data of the various artefacts. The tools are open-source and publicly available at [19].

## Image preparation

Multiple Windows 8.1 and 10 virtual machines where created to aid testing. VirtualBox was used as virtualization software. `VBoxManage clonemedium` was used to convert the virtual machine's disks to raw disk images. All testing was done on these converted images.

Two types of images were used:

- Acquired from Windows 8.1 and Windows 10 systems
- Acquired from external disk (no OS)

All these images were created for and operated by a virtual machine running in VirtualBox. As Windows produces a lot of disk activity on its own, the "system" images contain a lot of clutter that makes isolating the test cases more difficult. The "external disk" image, however, allowed for a very clean testing environment. The external disk is an extra virtual drive that was created from the graphical VirtualBox interface (by opening the settings menu for the virtual machine, opening the `Storage` submenu, choosing `Adds new storage attachment..`

The external disk was formatted to NTFS from within a Windows 10 virtual machine. Although Windows is supposed to use the UsnJrnl by default since Windows Vista, the (virtual) external disk didn't have a UsnJrnl file. USN journalling had to be manually activated. The command that can be used for this is[1]:

```
fsutil usn createjournal m=1048576 a=1024 <volume path>
```

In order to work with the images, they were converted from their virtual container into a raw image using the following command [2]:

```
VBoxManage clonemedium --format raw <input> <output>
```

## Test case

A small test case was created to validate the model. The output illustrates a possible outcome of combining the UsnJrnl with the LogFile and MFT.

An "external disk" image with a size of 10 MB was used as test case. Some files were added, deleted and modified, which triggers Windows to create some data in the journal files. Table 4 gives an overview of the different files and their state on the disk.

| file name | MFT inum | sequence value | state |
|---|---|---|---|
| cat.jpg | 36 | 1 | in use |
| target.jpg | 40 | 1 | deleted |
| secret.txt | 41 | 1 | deleted & overwritten |
| password.txt | 41 | 2 | deleted |

Table 4: Files created by an fictional user, used to put the model in practice

When the "external disk" is mounted in Windows it only shows the `cat.jpg` picture as the other files are deleted. However, most of the deleted data can still be recovered. By combining the data from the different artefacts it is possible to retrieve the history of the already overwritten

[1] https://technet.microsoft.com/nl-nl/library/cc788042%28v=ws.10%29.aspx
[2] https://www.virtualbox.org/manual/ch08.html#idp46730494487616

file and represent it in a logical way. Part of the output can be seen in listing 1. The current sequence value is 3 and the `flags` field indicates that the MFT entry is not in use yet. We see that this MFT entry has already been used before with a sequence value of 1 and 2.

```
###############################################################################
# Current MFT information                                      ################
###############################################################################
MFT entry number: 41
Sequence value  : 3
Currently in use: False -> Historic data in MFT entry, easy to extract
File name       : password.txt

SUMMARY:
seq  USN record list
  1  [3064, 3168, 3272, 3376, 3456, 3536, 3616, 3696, 3776, 3856]
  2  [3936, 4096, 4200, 4304, 4392, 4480, 4568, 4656, 4744, 4832]
```

Listing 1: Start of the output for a single MFT entry. This MFT entry has sequence value 3 and its `flags` field indicates that it's not in use. Observations show that this means that the file with, in this case, sequence value 2, was the last to occupy this MFT entry. The data belonging to sequence value 2 can still be found in the MFT since this entry is not yet reused and therefor not overwritten yet.

By printing the combined history of the UsnJrnl and LogFile an overview can be created showing what has happened with this MFT entry. listing 2 shows the last USN record (4832) for this MFT entry with sequence value 2. This data shows the most illustrative information from the USN record such as file name, timestamp and reason. The output continues with more detailed information on the LogFile transaction such as its transaction number and the LSN entries with their corresponding operations.

```
USN      : 4832
File name: password.txt
Timestamp: 2016-01-29 21:29:12.795932
Reason   : FILE_DELETE|CLOSE

$LogFile transaction number: 38

LSN     Redo operation                    Undo operation
1085650 Delete Index Entry Allocation     Add Index Entry Allocation
1085675 Delete Index Entry Root           Add Index Entry Root
1085697 Deallocate File Record Segment    Initialize File Record Segment
1085711 Clear Bits in Nonresident Bitmap  Set Bits in Nonresident Bitmap
1085723 Set New Attribute Sizes           Set New Attribute Sizes
1085742 Update Nonresident Value          No-Operation
1085764 Set New Attribute Sizes           Set New Attribute Sizes
1085783 Forget Transaction                Compensation Log Record
```

Listing 2: USN record combined with a LogFile transaction - sequence 2

The last operation of sequence 2 was a `delete`, as can be seen in the `Reason` field. Tests pointed out that after this operation the MFT sequence value is incremented by one and the

`flags` field is modified so that it indicates that the MFT entry is not in use. This can be observed from listing 1.

```
USN      : 3856
File name: secret.txt
Timestamp: 2016-01-29 21:28:06.551178
Reason   : FILE_DELETE|CLOSE

LogFile transaction number: 126

LSN     Redo operation                    Undo operation
1082596 Delete Index Entry Allocation     Add Index Entry Allocation
1082620 Delete Index Entry Root           Add Index Entry Root
1082642 Deallocate File Record Segment    Initialize File Record Segment
1082656 Clear Bits in Nonresident Bitmap  Set Bits in Nonresident Bitmap
1082668 Set New Attribute Sizes           Set New Attribute Sizes
1082687 Update Nonresident Value          No-Operation
1082708 Set New Attribute Sizes           Set New Attribute Sizes
1082727 Forget Transaction                Compensation Log Record
```

Listing 3: USN record combined with a LogFile transaction - sequence 1

Listing 3 shows the last USN record (3856) of sequence value 1. This USN shows the deletion of the file. The file name which the MFT entry had before deletion can easily be identified. The timestamp of the USN record can also be combined with the LogFile transaction. LogFile transactions lack timestamping so this is an added benefit. The actual data belonging to this sequence of the MFT entry can not be easily recovered. Extra steps need to be taken to find the operations on the blocks which contained the data of this MFT entry (resident or non-resident). After that perhaps some data can be recovered.

# Conclusion

The UsnJrnl, holds information that's not contained in the LogFile, even though it's more concise. More importantly, however, it holds a combination of information which makes it a very good starting point of an investigation.

Combining the MFT, LogFile and UsnJrnl results in a much more valuable overview of what happened on a system than when taking any of these files alone. Combining them by hand, however, is laborious work. The model that was proposed in this paper suggests how one can go about linking the files together in an automated manner.

With a simple test case we've shown that it is possible to use the model to combine UsnJrnl information with LogFile and MFT information in software. The principles of this proof of concept can be taken further to create much more advanced tooling. As Microsoft hasn't released much formal documentation on the artefacts' structure, however, one should thread carefully when automating tasks as there are no promises on consistency of the data contained in the artefacts.

## Suggestions for future work.

In the developed tool not every MFT attribute is implemented. At the moment of this writing only the "Filename" and "Standard Information" attributes are implemented in the parsing programs. Even though these are the two most important ones, iplementing the other attributes will still add greatly to the insight that can be achieved.

There are some fields in both the LogFile and the UsnJrnl on which no information could be found. These fields may, however, contain very important data. Since low level documentation is sparse this knowledge needs to be gathered by reverse engineering the usage of the fields for different files and structures.

Although our research was not meant to focus on the LogFile it turned out a lot of effort went into finding out how to parse it. Still, parsing the LogFile can be improved especially on the part of interpreting the redo/undo data. This data can be different depending on the redo/undo opcode combination.

As discussed in this paper, one can obtain an MFT entry number from the LogFile by means of equation 1. During testing, however, it quickly became apparent that the equation doesn't always return a valid MFT entry number. For example, in case the LSN record contains an embedded USN record in its `redo data` field, this equation cannot be used.

It would be good to have an exhaustive list of LSN redo/undo pairs for which equation 1 *does* returns a valid MFT entry number.

## Appendix

$$MFT\ entry\ number = VCN \times Cs + \frac{mft\ cluster\ index \times Bs}{MFTes} \tag{1}$$

Where:

$$Cs = cluster\ size\ (KB)$$
$$Bs = block\ size$$
$$MFTes = MFT\ entry\ size$$

In many cases the parameters have standard values:

$$Cs = 4\ KB$$
$$Bs = 512\ bytes$$
$$MFTes = 1024\ bytes$$

In which case 1 can be simplified to:

$$MFT\ entry\ number = VCN \times 4 + \frac{mft\ cluster\ index}{2} \tag{2}$$

## References

[1] C. Lees, "Determining removal of forensic artefacts using the usn change journal," *Digital Investigation*, vol. 10, no. 4, pp. 300–310, 2013, ISSN: 1742-2876. DOI: http://dx.doi.org/10.1016/j.diin.2013.10.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287613001084.

[2] G.-S. Cho and M. Rogers, "Finding forensic information on creating a folder in $logfile of ntfs," English, in *Digital Forensics and Cyber Crime*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, P. Gladyshev and M. Rogers, Eds., vol. 88, Springer Berlin Heidelberg, 2012, pp. 211–225, ISBN: 978-3-642-35514-1. DOI: 10.1007/978-3-642-35515-8_18. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35515-8_18.

[3] D. Cowen. (Jan. 7, 2013). Ntfs triforce - a deeper look inside the artifacts, [Online]. Available: http://www.hecfblog.com/2013/01/ntfs-triforce-deeper-look-inside.html (visited on 01/02/2016).

[4] J. Oh. (Jan. 25, 2015). Ntfs log tracker, [Online]. Available: https://sites.google.com/site/forensicnote/ntfs-log-tracker (visited on 01/27/2016).

[5] Z. Weger. (Nov. 13, 2015). Ntfs-linker, [Online]. Available: https://strozfriedberg.github.io/ntfs-linker/ (visited on 01/27/2016).

[6] C. Brian, English. Addison Wesley Professional, 2005, ISBN: 0-32-126817-2.

[7] (Nov. 13, 2012). Everything i know about ntfs, [Online]. Available: http://www.kes.talktalk.net/ntfs/ (visited on 01/27/2016).

[8] R. Russon and Y. Fledel, *Ntfs documentation*, English, 2004.

[9] J. Medeiros, *Ntfs forensics: A programmers view of raw filesystem data extraction*, Grayscale Research, Jun. 2008. [Online]. Available: http://grayscale-research.org/new/pdfs/NTFS%20forensics.pdf.

[10] J. Schicht. (Oct. 27, 2015). Logfileparser, [Online]. Available: https://github.com/jschicht/LogFileParser (visited on 01/27/2016).

[11] P. Polakovic. (Jan. 20, 2016). Ntfs logfile parser, [Online]. Available: http://www.codeproject.com/Articles/1072219/NTFS-LogFile-parser (visited on 01/27/2016).

[12] D. Cowen. (Aug. 13, 2013). Daily blog #51: Understanding the artifacts usnjrnl, [Online]. Available: http://www.hecfblog.com/2013/08/daily-blog-51-understanding-artifacts.html (visited on 01/27/2016).

[13] C. Harrell. (Jan. 1, 2013). Re-introducing $usnjrnl, [Online]. Available: http://journeyintoir.blogspot.nl/2013/01/re-introducing-usnjrnl.html (visited on 01/27/2016).

[14] Microsoft. (). Usn_record_v2 structure, [Online]. Available: https://msdn.microsoft.com/en-us/library/aa365722(v=vs.85).aspx (visited on 01/27/2016).

[15] ——, (). Change journals, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/aa363798(v=vs.85).aspx (visited on 01/27/2016).

[16] ——, (). Usn_record_v3 structure, [Online]. Available: https://msdn.microsoft.com/en-us/library/hh802708(v=vs.85).aspx (visited on 01/27/2016).

[17] ——, (). Usn_record_v4 structure, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/dn302075(v=vs.85).aspx (visited on 01/27/2016).

[18] D. Cowen. (Jun. 14, 2014). Daily blog #359: Carving usn records, [Online]. Available: http://www.hecfblog.com/2014/06/daily-blog-359-carving-usn-records.html (visited on 01/27/2016).

[19] NTFSparse. (). Ntfs_parse on github, [Online]. Available: https://github.com/NTFSparse/ntfs_parse (visited on 02/04/2016).