Digital Investigation 28 (2019) S30–S39

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Using the object ID index as an investigative approach for NTFS file systems

Rune Nordvik [a, b, *], Fergus Toolan [b], Stefan Axelsson [a, c]

[a] Norwegian University of Science and Technology, Norway
[b] Norwegian Police University College, Norway
[c] Halmstad University, Sweden

ARTICLE INFO

Article history:

Keywords:
User activity
NTFS
Object ID

ABSTRACT

When investigating an incident it is important to document user activity, and to document which storage device was connected to which computer. We present a new approach to documenting user activity in computer systems using the NTFS file system by using the $ObjId Index to document user activity, and to correlate this index with the corresponding records in the MFT table. This may be the only possible approach when investigating external NTFS storage devices, and is hence a valuable addition to the storage forensics toolbox.

Introduction

Users interact with the file system by navigating, creating, moving, renaming, copying or deleting files, or directories. Digital forensic investigators normally use digital forensic tools to investigate criminal cases (Garfinkel, 2010; Gl and Kugu, 2017). When digital forensic tools parse the NTFS file system they often show only selected parts of each MFT record. In order to validate the results of the tools it would be necessary for digital forensic investigators to use hex viewers, or tools such as mftrcrd (Schicht, 2018) to manually interpret the MFT records. In NTFS, metadata about files is mainly found in the system file $MFT (master file table) [2, p.353], but metadata might exist in other system files including $ObjId, $LogFile, $UsnJrnl, $Secure, etc. Typically, file metadata could include timestamps, file names, block allocations (data runs or extents), Object IDs, different indexes, etc [2, chap. 13].

This paper will focus on Object Identifers (OIDs). The Object ID index found in the $ObjId system file can help the investigator to find all allocated files that have an Object ID, which will assist in event reconstruction of user activity. OIDs are created based on typical user activity and are used by Windows in order to track an object (file, directory or link) even if the object changes location or name (Microsoft, 2016). OIDs will be created when a file is opened by the user in Windows File Explorer, or when the file is opened or saved by some applications (Parsonage, 2008). A user can also use the command line tool **fsutil objectid** to create, delete or set OIDs. If a user moves a file to another volume the Object ID might change, however, the Birth Object ID and the Birth Volume Object ID should be preserved (Microsoft, 2016). A volume is a collection of addressable sectors that can be used for storage, and a volume can also be a partition [2, p.70]. In the context of this paper, the volume is a partition using the NTFS file system. According to Microsoft the Windows OS uses OIDs in order to track files (Microsoft, 2016).

A digital forensic tool might show OIDs connected to a file, but different digital forensic tools deviate in how OIDs connected to a file are presented. We performed an experiment to determine if forensic tools display OIDs. Thus we tried X-Ways Forensics and Autopsy on a file which was known to have connected OIDs. The results were that X-Ways Forensics showed only the Object ID key, and Autopsy (Sleuthkit) failed to show any information relating to OIDs. EnCase shows Object IDs and parses the Object ID timestamp, sequence number and the MAC address (Habben, 2018). If a file, directory or link is assigned OIDs, the following will be assigned:

- Object ID (used as a key in the index)
- Birth Volume Object ID (special identifier equal to the Object ID of the $Volume system file from the volume the OIDs were created)
- Birth Object ID (equal to the first Object ID assigned and should not change)
- Domain Object ID (always zeros, reserved)

* Corresponding author. Norwegian University of Science and Technology, Norway.
E-mail address: rune.nordvik@phs.no (R. Nordvik).

https://doi.org/10.1016/j.diin.2019.01.013

It is not enough to just display an artifact, the investigators need to understand what it means. The authors consider OIDs to be important for digital forensics for the following reasons:

- OIDs will show which boot session a file with OIDs belongs too (Leachi et al., 2005), which can assist in timeline creation.
- OIDs can show the node (MAC-address) used by the computer that created the OIDs (Leachi et al., 2005). This means we will be able to determine to which computers the external storage medium has been attached, as long as the user has accessed files and created new OIDs.
- OIDs can show in which sequence files have been assigned OIDs within a boot session. This might assist in detecting manipulation of timestamps and in building timelines.
- The $ObjId index can be used as a triage tool in order to identify files or directories the user has accessed.
- The Birth Volume Object ID might be used to identify the file system volume used when the file was first assigned OIDs.

The Object ID is a unique 16 byte identifier used to identify files on a NTFS volume (Microsoft, 2016). Any file that obtains an Object ID, will also have a Birth Volume Object ID, a Birth Object ID and a Domain ID (Microsoft, 2016). The Birth Volume Object ID (16 bytes) is used for identifying the volume the file was located on when it first obtained an OID (Microsoft, 2016). The Birth Object ID (16 bytes) is the first Object ID assigned to the file. The Object ID may change if the file is moved, but the Birth Object ID should remain constant (Microsoft, 2016). The Domain ID is a 16 byte structure reserved for identifying a domain, and must be 16 bytes of zeros (Microsoft, 2016). Our experiments attempt to observe and assess if the description of **fsutil** by Microsoft is still true in Windows 10.

OIDs are 16 bytes in size and contain a 60 bit timestamp, which is the number of 100 ns intervals since 15th of October 1582 (Leachi et al., 2005; Parsonage, 2008). This timestamp is found in the first 60 bits of the OID and is related to the start of the boot session in which the OID was created (Leachi et al., 2005). The two least significant bytes of this timestamp, when interpreted as Little Endian, are also used as a counter showing the order of OID creation within the specific boot session (Parsonage, 2008). The counter is the only two bytes that separates Object IDs assigned in the same boot session. The timestamp can be converted to FILETIME by subtracting the hex value 0x146BF33E42C000, allowing tools that interpret FILETIME to convert it. The OIDs have a clock sequence which will be identical for all OIDs created in a particular boot session. Finally the last 6 bytes of the OID will normally include the MAC address of the default Network adapter. If no NIC is available this will contain a random number (Leachi et al., 2005). A graphical illustration is shown in Fig. 1.

The Object ID is used as an index key in the $ObjId$O file and this Object ID is also located in the Object ID Attribute (type 0x40) in the corresponding MFT record. We can also find the MFT record number in the $ObjId$O index entry [2, p.335]. This way it is easy to find the correct record in the index, knowing the Object ID key from the MFT record, but also to find all Master File Table (MFT) records that have an Object ID by examining the $ObjId$O index entries. It is the latter approach that is presented in this paper. A prototype tool has been developed which implements this approach and was used during the course of these experiments.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 illustrates the research goals. Section 4 describes the methodology and details about our experimental setup. Section 5 presents the results of our experiment. Section 6 presents the evaluation methodology and results of assessing the feasibility and reliability of the approach. Section 7 discusses and interprets the results. Finally, Section 8 concludes and provides recommendations for future work.

Related work and contributions

Previous work on Object IDs has focused on interpreting the meaning of OIDs found in link files (shortcut files), or OIDs from link files found in the NTFS journal. In this section we describe this previous related work and finally we describe our contributions.

Related work

Carrier provided a description of the $OBJECT_ID structure [2, p.367] and the index $ObjId structure [2, pp.386–387]. Carrier describes OIDs as an alternate method of addressing files, which allows for locating the file even if the name and location have changed [2, p.335]. Carrier does not describe the format the OIDs are using or their exact meaning.

In Windows, users can create shortcut files that point to other files. The Windows OS often creates these shortcut files automatically based on user activity. These shortcut files normally have the extension **lnk** and are called link files (Parsonage, 2008). Parsonage describes which OIDs can be found within link files and compares them to the output of the **fsutil** command. Within link files the following OIDs might be stored:

- New VolumeID (corresponds to the Volume Object ID of the $Volume system file, but not found in the $ObjId index if this is from another NTFS volume)
- New File ObjectID (should be identical with the Object ID found in the $ObjId index)

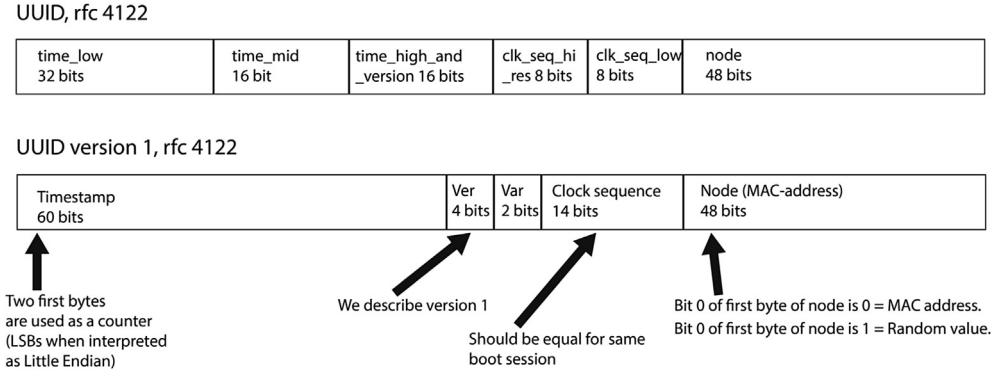

Fig. 1. Structure of an Object ID UUID version 1.

- Birth VolumeID (should be identical with the Birth Volume Object ID found in the $ObjId index entry, but the move bit is not set)
- Birth File ObjectID (should be identical with the Birth Object ID found in the $ObjId index entry)

Parsonage (2008) describes the importance of Link Files, and mentions that there exists an index of Object IDs, however, little use is made of this. This research is based on the description of OIDs from this article, but does not focus on the binary content of link files. Parsonage claims that the OIDs are not preserved on removable media. This paper attempts to determine the veracity of this claim. We will use the $ObjId index as an approach to find all allocated files on a volume which would indicate user activity. The OID structure, described in (Leachi et al., 2005; Parsonage, 2008), can be used to connect the device to one or multiple computer system(s) using the MAC address included in the OID.

In Windows, jump lists are used for saving recently used items for an application or for the OS itself. For instance, the list of recently opened documents is made possible using a jump list. Singh & Singh (Singh and Singh, 2016) describe jump lists, and show how to interpret these for Windows 10, which is different from Windows 7 and 8. Their work shows that OIDs are used in DestList and LNK streams, which includes embedded shortcut files. Within these shortcut files/streams both the new Volume Object ID and the Birth Volume Object ID might be shown, which is helpful for tracking purposes. However, an investigator may have no access to the system volume, meaning they would have no access to the jump lists. In these cases the investigator only has the $ObjId index, the MFT table or other system files to investigate. In this index we find the Object ID, the Birth Volume Object ID, the Birth Object ID, the Domain ID (unused) and the reference to the MFT record [2, pp.386–387].

McGrath & Gladyshev (McGrath and Gladyshev, 2013) describe how to use the NTFS $logfile to find cleartext files after encryption. The authors use **fsutil** to determine the Birth Volume Object ID from the known ciphertext file. They state that the ciphertext file and the fact that the Birth Volume Object ID was found inside the $logfile and conclude that the encryption took place on that volume. They also do the same for the cleartext file found in the $logfile. Even if the file was deleted, the previous dataruns, used by the cleartext file, might still be present within $logfile. In our experience not all encryption software creates Object IDs or link files, and not all cleartext files have OIDs. Some encryption software will create OIDs both for the cleartext file selected and the ciphertext file created.

Cowen (2018a) has performed a few experiments regarding Object IDs. The results of his experiments are not published in a peer-reviewed paper. He is using a python script parsing every MFT record for the OBJECT_ID attribute. He suggests that the last 6 bytes of the Object ID is the MAC address, even for the $Volume. His testing also shows that he does not find valid MAC addresses for the Object ID connected to the $Volume. Further, his script source code shows he has based his parsing on the knowledge from Parsonage (Cowen, 2018b). Cowen's testing shows that there are Object IDs even for some of the files installed on the system, and that their MAC addresses have been preserved. He concludes that there are less Object IDs for pre-installed files on Windows 10 compared to Windows 7. Cowen does not use the system file $ObjId in his experiments, and therefore he only finds the main Object ID key.

Yamazaki (2015) has published a closed source tool, **fte**, that should be able to parse the $MFT, $ObjId system file and other NTFS indexes. When we tested this tool on Windows 10, it was only able to parse the $ObjId system file when selecting a live volume, and only if the Index Allocation Attribute existed. The tool shows correctly the date from the Object ID, but the column describes ctime which easily could be interpreted incorrectly as change time. The tool detected correctly if a file has been moved from another NTFS volume. The fte tool does not parse the Index Root Attribute, when there are just a few files with OIDs on a volume.

Contributions

None of the above related work address the meaning of the Object IDs saved in the $ObjId index. As can be seen, no-one has previously identified what kind of operations update the $ObjId index. Hence, our contributions include novel investigation methods for:

- Event reconstruction of user activity using the Object ID index correlated with the $MFT table.
- Documentation of computer devices to which an external hard drive has been attached.
- Finding the boot times of a computer by investigating the Object ID index of attached NTFS volumes, which could be correlated with external NTFS storage devices that have been attached.
- Creating timelines.
- Detection of manipulation of timestamps by analyzing Object IDs.

Research goals

This research focuses on the feasibility and reliability of using $ObjId$O index to document user activity.

- **Feasibility:** The selected approach should be feasible for use with new versions of Windows, and therefore we have selected Windows 7 and 10 as our test systems.
- **Reliability:** The selected approach should reliably detect user activity.

Research questions

This paper aims to determine if user activity can be documented from non-OS NTFS volumes using FS metadata from the NTFS file system. It also aims to determine if it is possible to discover what machine(s) a device was connected to by using artifacts present on the device.

When the NTFS system volume is unavailable, investigators can no longer rely upon jump lists, recent link files, registry, event logs and prefetch files in order to determine user activity. Only the artifacts found in the NTFS file system can be relied upon, hence, it is the opinion of the authors that this approach may be the only means of recreating user activity for external NTFS media.

Automation

Forensic tools have basic support for parsing the MFT record attributes, but to the authors' knowledge only two tools, mftrcrd (Schicht, 2018) and fte (Yamazaki, 2015), show all the timestamps from all the File Name attributes (FNAs) within the MFT record. Furthermore, only the fte tool is able to parse the $ObjId index to list all OIDs in the system under investigation. As part of this work an open source tool has been developed that automates the parsing of $ObjId and correlation with the pertinent attributes in the MFT record. The prototype tool, **NTFSObjIDParser** (Nordvik, 2019), was developed in C++ using the graphical QT Libraries. The target users are computer forensic investigators. Users need to export the $MFT table and the $ObjId$O Index Allocation Attribute (type 0xA0) or

the Index Root Attribute (type 0x90), as shown in Fig. 2 and Fig. 4. Using these inputs, the prototype tool will correlate each index entry with the corresponding MFT record.

In NTFS indexes are used for storing $MFT attributes in a sorted order, and a B-tree is used [2, p.290]. The root node is always located in the resident Index Root Attribute [2, p.294]. If all the nodes can not fit resident (7 or more entries) in the Index Root attribute, a non-resident Index Allocation Attribute is used [2, p.294]. The $MFT record of the $ObjId system file contains these two attributes (Index Root and/or Index Allocation), and the indexed attribute is in this case the $MFT $OBJECT_ID Attribute (type 0x40).

Methodology

The **NTFSObjIDParser** prototype tool was used in our experiments. The output was verified using the **xxd** hex viewer and the forensic suite **Sleuthkit** (Carrier, 2017).

Object ID creation

The purpose of this experiment is to determine when an Object ID is created. Multiple tools were evaluated. These included: the command prompt; File Explorer; Notepad; VeraCrypt; and LibreOffice. The scenarios tested on the NTFS filesystem were:

- **File creation:** Using a tool to create a new file.
- **Opening a file:** Using a tool to open an existing file, with or without an Object ID, and test if rebooting impacts the result.
- **Copying a file (same volume):** Using a tool to copy a file, with or without an Object ID, to the same volume, and test if rebooting impacts the result.
- **Copying a file (other volume):** Using a tool to copy a file, with or without an Object ID, to another NTFS volume, and test if rebooting impacts the result.
- **Moving a file (same volume):** Using a tool to move a file, with or without an Object ID, to another directory on the same volume, and test if rebooting impacts the result.
- **Moving a file (other volume):** Using a tool to move a file, with or without an Object ID, to a directory on another volume, and test if rebooting impacts the result.
- **Deleting a file:** Using a tool to delete a file.

The reboot means that after the test, the machine is rebooted, and the test repeated. The reboot was performed to see if the 60 bit timestamps within the OIDs were updated to the last timestamp for the most recent boot time. This was tested for all scenarios where files have an existing OID.

After each test, and after the reboots, the MFT table and the clusters found in the Index Allocation Attribute data runs were exported. Simple Sleuthkit commands (Carrier, 2017) were used for extraction of the MFT, but during the experiments we thought it was necessary to use **dd** to gather the clusters from the Index Allocation Attribute. Sleuthkit v. 4.4.1 to v. 4.6.2 did not show the Index Allocation Attribute, only showing the Index Root Attribute.

```
# List partition table
sudo mmls /dev/rdisk2
# Export the record 0 (MFT Table)
# from volume starting on sector 32
sudo icat -o 32 /dev/rdisk2 0 > MFT.bin
# Showing the 25th MFT record
dd if=MFT.bin bs=512 skip=$(( 25*2 )) count=2 | xxd
```

Fig. 2. Exporting the MFT table, and MFT record number 25.

However, it is possible to extract an existing Index Allocation Attribute using Sleuthkit by combining the MFT record number and the attribute type. The USB device was unmounted from Windows, and mounted in MacOS where Sleuthkit was installed.

The **mmls** command in Fig. 2 was used to show the partition tables, and to find the correct volume. Using this information the MFT table was exported. From the extracted MFT table the $ObjId MFT record (25) was shown in the hex viewer. It should be noted that the 25th MFT record is not always used for the $ObjId file.

Fig. 3 shows the Index Allocation Attribute which commences at offset 0x158 (type 0xA0). Skipping 0×48 bytes, and examining the value at offset 0×1A0, the bytes 0x110123 are seen. This provides the data run for the attribute in question. Interpreting this shows that the contents start at cluster 0x23 and occupy a single cluster. The test disk has 8 sectors per cluster, therefore the data content of the $ObjId$O index is located at sector 280 relative to the start of the volume. Allowing for the 32 sectors before the volume, sectors 312–319 are extracted as shown in the first command in Fig. 4.

The last command skips the index file header (64 bytes) within this file and shows an object ID index entry. The result of this is shown in Fig. 5. We observed that when it was less than 7 entries, there was no Index Allocation Attribute, and all the indexes were, in this case, stored resident in the Index Root Attribute. If there are 7 or more entries, it is necessary to extract the Index Allocation Attribute, and to skip the file header (32 bytes) in order to find the first Object ID index entry.

The structure of an Object ID index entry is shown in Fig. 6. The basic parsing of this index structure is defined by Carrier [2, p.387]. The 8 bytes at offset 0x20 provide a reference to the MFT table. The 16 most significant bits (in this case all multibyte data fields are stored in Little Endian format) are for the sequence number and the remainder is for the MFT record number (0x25). There are a total of 4 universally unique identifiers (UUIDs), but the Domain UUID always has a zero value. The Object ID UUID will also be found in the MFT record Object ID Attribute, but none of the other UUIDs will be present. Both the Object ID and the Birth Object ID will have a 60 bit timestamp, as described in Section 1. The two least significant bytes represent the Object ID order, in other words the order in which OIDs were created. In bytes 8 and 9 of the UUID the clock sequence number is found. Remember to set the two variant bits to 0. Then we read the two bytes in Big Endian order. This sequence number is equal for all UUIDs that were created/updated within the same boot session. The last 6 bytes, when read as an array of bytes rather than a multibyte field, will show the MAC address of the standard NIC used. If no NIC was used a random number appears at this location (Parsonage, 2008). More details on how to parse an Object ID entry are shown in Table 1 and in Fig. 6.

Manually parsing each Object ID entry is too time consuming when every index entry must be parsed, and therefore the prototype tool, *NTFSObjIDParser*, is used for automation. The prototype correlates Object ID entries with the MFT record found in the entry reference by parsing the MFT record's Standard Information

```
00000150: 0000 0000 0000 0000 a000 0000 5000 0000  ..............P...
00000160: 0102 4000 0000 0300 0000 0000 0000 0000  ..@.............
00000170: 0000 0000 0000 0000 4800 0000 0000 0000  ........H.......
00000180: 0010 0000 0000 0000 0010 0000 0000 0000  ................
00000190: 0010 0000 0000 0000 2400 4f00 0000 0000  ........$.O.....
000001a0: 1101 2300 00c0 ffff b000 0000 2800 0000  ..#.........(...
```

Fig. 3. Hex dump of the Index Allocation Attribute.

```
# Export the Index Allocation non resident content
sudo dd if=/dev/rdisk2 bs=512 skip=312 count=8 of=ObjectID.
    ↪ bin
# Alternative method: It is possible to extract the
# Index Allocation Attribute using Sleuthkit, however
# it's corresponding MFT-fileid-attribute is not shown by
# fls when using Sleuthkit. If $ObjId is inode 25:
sudo icat -o 32 /dev/rdisk2 25-160 > ObjectID.bin
# If no Index Allocation Attribute exist, extract the
# Index Root Attribute
sudo icat -o 32 /dev/rdisk2 25-144 > ObjectID-IR.bin

# Show one of the index entries
sudo dd if=ObjectID.bin bs=1 skip=64 count=88 | xxd
```

Fig. 4. Exporting the Object ID Index Allocation non resident data, and show one Object ID Index Entry.

```
00000000: 2000 3800 0000 0000 5800 1000 0000 0000    .8.....
    ↪ X.......
00000010: 2535 8c37 c7f3 e611 9c55 0800 2737 afb0
    ↪ %5.7.....U..'7..
00000020: 2500 0000 0000 0100 0000 0000 0000 0000
    ↪ %...............
00000030: 0000 0000 0000 0000 2535 8c37 c7f3 e611
    ↪ ........%5.7....
00000040: 9c55 0800 2737 afb0 0000 0000 0000 0000   .U..'
    ↪ 7..........
00000050: 0000 0000 0000 0000                       ........
```

Fig. 5. Hex dump of an Object Index Entry.

Attribute (SIA), all File Name Attributes (FNAs) and the Object Identifier Attribute (OIA). The first column in Fig. 7a shows the MFT record reference for each row. This shows which rows represent the same item. Then the byte offset to the entry or to the MFT record is shown, and the relative offset from each entry where the MFT attribute or Object ID type can be found. Knowing the offsets will allow verification of results by computer forensic investigators.

Next the MFT Header flags are shown if the entry is a MFT record or the Object ID entry flags are shown if the entry is an OID. The MFT header flags will show if the file or directory is allocated or unallocated. It is unlikely that unallocated files or directories will be found, this is due to the fact that deleted files will be removed from the index, however it will be present in the MFT table as long as the record has not been reused. An attempt was made to find patterns describing what actions created the OID: creation; opening; copying; moving; deleting. This is not fully implemented in the prototype. In the Name column we show the OIDs or File Names. The SIA does not have a File Name or an OID, so it is left empty.

For Object ID, Birth Object ID and the MFT OIA attribute the Created timestamps are shown. It should be noted that the time is the system boot time before creating the OIDs. For SIA or FNA the

```
typedef struct _INDX_ENTRY
{
        qint16 OffsetData; // 0x00
        qint16 SizeData;   // 0x02
        qint8 Padding1[4]; // 0x04 - Unused
        qint16 SizeIndexEntry; //0x08
        qint16 SizeIndexKey; // 0x0A Size of the Object ID
        qint32 Flags; // 0x0C - DOS flags
        qint8 ObjectID[16]; // 0x10 - Used as an index key
        quint64 MFTRecord; // 0x20
        qint8 BirthVolumeID[16]; // 0x28 Does not follow
            ↪ the standard for OIDs, as described. Can be
            ↪ correlated to the $Volume Object Attribute.
            ↪ Windows 10 set it to zero for external
            ↪ storage devices!
        qint8 BirthObjectID[16]; // 0x38, Should remain the
            ↪ same
        qint8 DomainID[16]; // 0x48 Not used, set to zero
            ↪ values
} INDX_ENTRY;   // Total of 88 bytes or 0x58 bytes
```

Fig. 6. C structure of an Object ID index entry.

Table 1
Offet table index entry, based on [2, pp.386–387].

Offset	Length	Meaning
0x00	0x02	Offset to data
0x02	0x02	Size of data
0x04	0x04	Padding (Unused)
0x08	0x02	Size of Index Entry
0x0A	0x02	Size of Index Key (Object ID)
0x0C	0x04	Flags
0x10	0x10	Object ID UUID (the key)
0x20	0x08	Reference to MFT record
0x28	0x10	Birth Volume Object ID UUID
0x38	0x10	Birth Object ID UUID
0x48	0x10	Domain ID UUID

Created, Modified, Record Modified and Accessed timestamps are shown, as can be seen from Fig. 7b. Note that these timestamps are approximately real time, but that the Accessed timestamp does not get updated all the time. Then the MAC address computed from the last 6 bytes of the OIDs is shown. In the field Object ID Order the decimal value of the two least significant bytes of the 60 bit timestamp in the OIDs is shown. This is not shown for the Birth Volume Object ID, since this OID does not have a timestamp. The last column shows the clock sequence, which shows which OID entries were created within a boot session.

We used Virtual Box v. 5.1 to virtualize Windows 7 Home Premium SP1 (32bit) and Windows 10 Pro (64bit). The attached SATA USB3 disks were of the type Lacie Porsche Mobile (1 TiB), each using one volume and formatted as NTFS. For Windows 7 we needed to install USB3 drivers. Since we were using virtual machines, the USB disks were automatically released to the host (MacOS High Sierra v 10.13) OS when rebooted. This was also why we observed that the $Volume Object ID was not always set. In the final stages of preparing this paper, we found that we could add the USB device to the USB device filters in the Settings, Ports, USB in Virtual Box. This way we could restart the virtual machine while the USB disk was attached during the reboot. The internal NTFS volumes were created by adding a vmdk disk device using Virtual Box, and then formatting it in Windows. We also tested using different USB thumb drives, however rebooting with the USB thumb drives attached did not assign a $Volume Object ID.

Results

We observed that for the Index Allocation Attribute (type 0xA0) to be used as an attribute in the $ObjId MFT record, it is necessary to

	MFT Record	Byte Offset	Attribute or Type	MFT Header Flags	Volume Action	Name
8	41	ObjectID$O offset = 152 + 16	ObjectID	0		4b87f0a4fef4e6119c5608002737afb0
9	41	ObjectID$O offset = 152 + 40	BirthVolumeID	0		00000000000000000000000000000000
10	41	ObjectID$O offset = 152 + 56	BirthObjectID	0		4b87f0a4fef4e6119c5608002737afb0
11	41	ObjectID$O offset = 152 + 72	DomainObjectID	0		00000000000000000000000000000000
12	41	MFT offset = 41984+56	SIA	Allocated File		
13	41	MFT offset = 41984+152	FNA	Allocated File		\.\testin2.odt
14	41	MFT offset = 41984+264	OIA	Allocated File		4b87f0a4fef4e6119c5608002737afb0

(a)

	Created	Modified	Record Modified	Accessed	MAC Address	ObjectID Orde
8	Fri Feb 17 10:48:40 2017				8-0-27-37-af-b0	34635
9						
10	Fri Feb 17 10:48:40 2017				8-0-27-37-af-b0	34635
11						
12	Sun Feb 19 21:44:47 2017	Sun Feb 19 21:44:47 2017	Sun Feb 19 21:44:47 2017	Sun Feb 19 21:44:47 2017		
13	Sun Feb 19 21:44:47 2017	Sun Feb 19 21:44:47 2017	Sun Feb 19 21:44:47 2017	Sun Feb 19 21:44:47 2017		
14	Fri Feb 17 10:48:40 2017				8-0-27-37-af-b0	34635

(b)

Fig. 7. NTFSObjIDParser output. Results are split between (a) and (b).

have more than 6 entries on a newly NTFS formatted volume. This might also depend on the number and size of the attributes within the $ObjId MFT record. If we have less entries, the indexes will be found in the Index Root Attribute (type 0x90). In the experiment a NTFS formatted USB disk was used. Since MFT record 3 ($Volume) did not get an Object ID attribute, the value used for Birth Volume Object ID UUID was zero. This result was unexpected, as all documentation consulted described that the Birth Volume Object ID should be assigned a unique value identifying the volume (Microsoft, 2016; McGrath and Gladyshev, 2013; Singh and Singh, 2016; Parsonage, 2008). The missing Birth Volume Object ID UUID was also observed when using Windows 7. In these cases the $Volume Object ID attribute was also not present in the MFT table. This was observed on recently created volumes on internal disks, and on removable disks. Whenever the $Volume Object ID attribute was available in the MFT table, then a non-zero Birth Volume Object ID UUID was present in the $ObjId index. We were only successful in creating an Object ID for the $Volume system file if we performed formatting of an internal or external disk using Windows 7 or 10. We also observed that a reboot might be necessary after the formatting in order for the $Volume Object ID to be assigned, and that the disk must be attached during the boot process. When this internal or external disk was quick reformatted again, the Object ID for the $Volume system file was normally preserved.

In the following tables (2–8) the tests that were performed are summarized. The following abbreviations have been used: W7 (Windows 7); W10 (Windows 10); OID (Object ID); BOID (Birth Object ID); and BVOID (Birth Volume Object ID). The **OS** column contains the operating system used. **Impact** contains the different OIDs that the action might impact. **Existing OID** has the value Yes if the file had existing OIDs before the operation was performed, **Preserved OID** contains Yes if previous OIDs from the source file were preserved after the operation. **New OID** has the value Yes if the action created a new Object ID. **Tool** describes the tool used for the operation.

File creation

Table 2 shows that creating a file makes an entry in the $ObjId$O index if File Explorer is used on Windows 10, but not when using Windows 7. If LibreOffice is used to create a new file a new entry is created in the $ObjId$O index on both versions of Windows. If the command prompt is used and the output is redirected to a file, no entry is made in the Object ID index. When using Notepad to create a file, no entry is made in the Object ID index. If File Explorer is used to extract a zip container (including a directory and a file), this creates the directory with an Object ID entry in Windows 10, but not in Windows 7. However, the file that was extracted did not get any entry in the Object ID index. A 100 MiB container, created using VeraCrypt, did not result in any OIDs.

Opening a file

Table 3 shows that if a file with no Object ID entry was opened by double clicking on it in File Explorer, then it received an Object ID entry. Identical UUIDs for Object ID and Birth Object ID were created. When double clicking a file with existing OIDs in File Explorer, the OIDs were preserved. This was also the case if we rebooted the system first. If LibreOffice was used to open a file without OIDs, a new entry was added to the Object ID Index. Using Libreoffice to open a file with existing OIDs preserved the OIDs. The same behavior was observed when rebooting the system first. No deviations were observed between Windows 7 and 10 when opening files. Both Notepad and the Command Prompt failed to create an OID after opening a file. They did, however, preserve existing OIDs.

Copying a file (same volume)

In Table 4 File Explorer is used to drag and drop a file while holding CTRL (this ensures the file is copied) (Microsoft, 2001). The original file did not have any Object ID before the operation. Both the original and the copy did not get any entry in the Object ID index after this operation. If the source file had OIDs before, these are preserved for the source file, but no OIDs were found for the new copy. Using LibreOffice Save As created new OIDs for the copy. If the **copy** terminal command was used to copy a file to the same volume, it did not create new OIDs for the copy. Notepad was used to create a copy using Save As. In Windows 7 OIDs were created for the copy, but not in Windows 10. Copying a file also is creation of a file based on an existing file. A new entry will be created in the MFT table, and therefore this is also a part of the copy operation.

Copying a file (other volume)

Table 5 shows the results of copying a file to another volume using File Explorer's drag and drop functionality while holding the CTRL key. The results, regarding OIDs, were the same as when the file is copied to the same volume. We also show the result when using LibreOffice's Save As feature, which created new OIDs for the target file. When using Notepad's Save As feature only Windows 7 created new OIDs for the target file. Using the command prompt **copy** command did not create OIDs for the target file.

Moving a file (same volume)

In Table 6 File Explorer's drag and drop functionality is used while holding the SHIFT key (to ensure the file was moved) (Microsoft, 2001). The file did not get an entry in the Object ID index after this operation. If the file had existing OIDs, then these were preserved. The same was observed when using the **move** command from the CMD prompt.

Table 2
Experiment 1 - Test 1: File creation.

OS	Impact	Reboot	Existing OID	Preserved OID	New OID	Tool
W7, W10	OID, BOID, BVOID	No	No	–	Yes	LibreOffice
W7	OID, BOID, BVOID	No	No	–	No	File Explorer (File or Directory)
W10	OID, BOID, BVOID	No	No	–	Yes	File Explorer (File or Directory)
W7	OID, BOID, BVOID	No	No	–	No	Extract directory from zip (File Explorer)
W10	OID, BOID, BVOID	No	No	–	Yes	Extract directory from zip (File Explorer)
W7, W10	OID, BOID, BVOID	No	No	–	No	Extract file from zip (File Explorer)
W7, W10	OID, BOID, BVOID	No	No	–	No	CMD prompt, Notepad
W7, W10	OID, BOID, BVOID	No	No	–	No	Veracrypt

Table 3
Experiment 1 - Test 2: Opening a file.

OS	Impact	Reboot	Existing OID	Preserved OID	New OID	Tool
W7, W10	OID, BOID, BVOID	No	No	–	Yes	File Explorer (double click)
W7, W10	OID, BOID, BVOID	No	Yes	Yes	No	File Explorer (double click)
W7, W10	OID, BOID, BVOID	Yes	Yes	Yes	No	File Explorer (double click)
W7, W10	OID, BOID, BVOID	No	No	–	Yes	LibreOffice (File Open)
W7, W10	OID, BOID, BVOID	No	Yes	Yes	No	LibreOffice (File Open)
W7, W10	OID, BOID, BVOID	Yes	Yes	Yes	No	LibreOffice (File Open)
W7, W10	OID, BOID, BVOID	No	No	–	No	CMD prompt, Notepad (File Open)
W7, W10	OID, BOID, BVOID	No	Yes	Yes	No	CMD prompt, Notepad (File Open)
W7, W10	OID, BOID, BVOID	Yes	Yes	Yes	No	CMD prompt, Notepad (File Open)

Table 4
Experiment 1 - Test 3: Copying file to the same volume.

OS	Impact	Reboot	Existing OID	Preserved OID	New OID	Tool
W7, W10	OID, BOID, BVOID	No	No	–	No	File Explorer (CTRL drag)
W7, W10	OID, BOID, BVOID	No	Yes	No	No	File Explorer (CTRL drag)
W7, W10	OID, BOID, BVOID	Yes	Yes	No	No	File Explorer (CTRL drag)
W7, W10	OID, BOID, BVOID	No	Yes	No	Yes	LibreOffice (Save As)
W7, W10	OID, BOID, BVOID	Yes	Yes	No	Yes	LibreOffice (Save As)
W7, W10	OID, BOID, BVOID	No	Yes	No	No	CMD prompt (copy)
W7, W10	OID, BOID, BVOID	Yes	Yes	No	No	CMD prompt (copy)
W7	OID, BOID, BVOID	No	Yes	No	Yes	Notepad (Save As)
W7	OID, BOID, BVOID	Yes	Yes	No	Yes	Notepad (Save As)
W10	OID, BOID, BVOID	No	Yes	No	No	Notepad (Save As)
W10	OID, BOID, BVOID	Yes	Yes	No	No	Notepad (Save As)

Table 5
Experiment 1 - Test 4: Copying file to another volume.

OS	Impact	Reboot	Existing OID	Preserved OID	New OID	Tool
W7, W10	OID, BOID, BVOID	No	No	–	No	File Explorer (CTRL drag)
W7, W10	OID, BOID, BVOID	No	Yes	No	No	File Explorer (CTRL drag)
W7, W10	OID, BOID, BVOID	Yes	Yes	No	No	File Explorer (CTRL drag)
W7, W10	OID, BOID, BVOID	No	Yes	No	Yes	LibreOffice (Save As)
W7, W10	OID, BOID, BVOID	Yes	Yes	No	Yes	LibreOffice (Save As)
W7, W10	OID, BOID, BVOID	No	Yes	No	No	CMD prompt (copy)
W7, W10	OID, BOID, BVOID	Yes	Yes	No	No	CMD prompt (copy)
W7	OID, BOID, BVOID	No	Yes	No	Yes	Notepad (Save As)
W7	OID, BOID, BVOID	Yes	Yes	No	Yes	Notepad (Save As)
W10	OID, BOID, BVOID	No	Yes	No	No	Notepad (Save As)
W10	OID, BOID, BVOID	Yes	Yes	No	No	Notepad (Save As)

Table 6
Experiment 1 - Test 5: Moving file to the same volume.

OS	Impact	Reboot	Existing OID	Preserved OID	New OID	Tool
W7,W10	OID, BOID, BVOID	No	No	–	No	File Explorer (SHIFT drag)
W7, W10	OID, BOID, BVOID	No	Yes	Yes	No	File Explorer (SHIFT drag)
W7, W10	OID, BOID, BVOID	Yes	Yes	Yes	No	File Explorer (SHIFT drag)
W7, W10	OID, BOID, BVOID	No	No	–	No	CMD prompt (move)
W7, W10	OID, BOID, BVOID	No	Yes	Yes	No	CMD prompt (move)
W7, W10	OID, BOID, BVOID	Yes	Yes	Yes	No	CMD prompt (move)

Moving a file (other volume)

The behaviour when moving a file from one NTFS volume to another depends on the OS used, and if the volume is an internal volume or an external volume. All our observations show that external disks have an Object ID equal to zero for the $Volume system file when these external disks have not been connected during reboot. This was observed for both Windows 7 and 10, regardless of the format method. However, internal disks when formatted normally (not quick) will have an Object ID for the $Volume file. If the same internal or external drive is reformatted, then the Object ID for the $Volume system file is preserved.

Table 7 shows File Explorer's drag and drop, while holding the SHIFT key, being used to move a file to a different volume. The file did get an entry in the Object ID index after this operation. The least significant bit, when reading the timestamp location as Little Endian, was set in the Birth Volume Object ID (the move bit). The Object ID and the Birth Object ID were preserved in this new index entry. We also observed an exception if the volume was an NTFS volume without an Object ID Attribute in the $Volume system file (external disk). In this case we observed that the moved file got a new Object ID and Birth Object ID in Windows 10, but no Object IDs

Table 7
Experiment 1 - Test 6: Moving file to another volume.

OS	Impact	Reboot	Existing OID	Preserved OID	New OID	Tool
W7, W10	OID, BOID, BVOID	No	No	–	No	File Explorer (SHIFT drag)
W7,W10	OID, BOID, BVOID	No	Yes	OID, BOID	BVOID LSb = 1	File Explorer (SHIFT drag)
W7	OID, BOID, BVOID	No	Yes	No	No	File Explorer (SHIFT drag). Target BVOID = 0
W10	OID, BOID, BVOID	No	Yes	No	Yes + (BVOID = 0)	File Explorer (SHIFT drag). Target BVOID = 0
W7, W10	OID, BOID, BVOID	Yes	Yes	OID, BOID	BVOID LSb = 1	File Explorer (SHIFT drag)
W7, W10	OID, BOID, BVOID	No	No	–	No	CMD prompt (move)
W7, W10	OID, BOID, BVOID	No	Yes	OID, BOID	BVOID LSb = 1	CMD prompt (move)
W7, W10	OID, BOID, BVOID	Yes	Yes	OID, BOID	BVOID LSb = 1	CMD prompt (move)
W7, 10	OID, BOID, BVOID	No	Yes	No	No	CMD Prompt (move). Target BVOID = 0
W7, 10	OID, BOID, BVOID	Yes	Yes	No	No	CMD Prompt (move). Target BVOID = 0

were created in Windows 7. For Windows 10 the Birth Volume Object ID was also set to 0.

For internal disks with an Object ID in the $Volume system file using the command prompt and the **move** command will preserve the Object ID and the Birth Object ID. The Birth Volume Object ID is also preserved, but the least significant bit is set to 1. If this bit is already set, then the Birth Volume Object ID is preserved. If the Object ID of the $Volume of the target volume was zero (external disk), then the OIDs were not preserved and no new OIDs were created even if the **move** command was used.

Deleting a file

If a file is deleted that has an entry in the $ObjId index, then the B-tree index will re-organize, and the result is often that the previous entry will be overwritten. The same was observed when using the **del** command in the CMD prompt. This is also shown in Table 8.

Evaluation

To evaluate the results, we focus on the two research goals described in section 3, Feasibility and Reliability.

Feasibility

Is it feasible to use the file $ObjId to document User Activity? Only the operations that actually create OIDs will be detected. Creating a new file (W10) or opening a file from File Explorer (W7 and W10), LibreOffice or other applications using the same API will be detected. We will not detect all user activity on the NTFS File System by only scrutinizing the Object ID index and the MFT records. However, it is feasible to assume that files with an entry in the Object ID index are there because of user activity. In many real cases, when the investigator only has access to a removable disk, this approach might be the only method of documenting user activity. It can also be used to map possible hosts to which the removable device has been attached.

Reliability

To answer the question of the reliability of this approach it is necessary to focus on what it does not detect. When a file is deleted, the B-tree $ObjId index is re-organized, and the previous content in the object index is normally overwritten. However, the Object ID in the MFT record can still be found, as long as the MFT record is not reused. This is an indication that the file has been opened, created or saved by the user or a software tool. Using the command line shell will normally go undetected, except when moving a file to another NTFS volume that has an Object ID assigned to the $Volume system file. However, if the $Volume Object ID is zero, moving a file to this volume will go undetected. When copying a file from one NTFS volume to another, the target file will not get OIDs. Creating an encrypted container will not generate OIDs when VeraCrypt is used.

It seems that all applications that use the Windows API FileOpen or FileSave dialogs will create OIDs. We tested this by creating a very simple tool that used the IFileOpenDialog interface, and OIDs were created when we used it to open a file that did not have OIDs. Normal users tend to use graphical user interfaces when using Windows, and therefore it is possible to detect a significant portion of user activity by utilizing the OIDs.

Discussion

Since OIDs are created based on typical user activities on NTFS volumes, using the $ObjId index will be a very efficient way to detect which files were accessed by the user. Not all Object IDs will have a LNK (shortcut) file in its Recent folder or as a LNK stream in a Jumplist. The user can even create their own LNK files, which could be stored in a selected directory. We do not claim that the Object ID index will find all user activity, but users using the File Explorer or other Graphical User Interface (GUI) tools have little control over index entry creation. Windows tool developers often use the Windows API instead of creating their own FileOpen or FileSave dialogs, meaning that Object ID creation will be enabled regardless of the programmer's awareness.

As observations show, normal user activity will create entries in the $ObjId index file. The $ObjId file is not directly accessible by the normal user, as it is a system file. This makes it more difficult to hide

Table 8
Experiment 1 - Test 7: Deleting a file.

OS	Impact	Reboot	Existing OID	Preserved OID	New OID	Tool
W7, W10		No	No	–	No	File Explorer (SHIFT delete)
W7, W10		No	Yes	No	No	File Explorer (SHIFT delete)
W7, W10		Yes	Yes	No	No	File Explorer (SHIFT delete)
W7, W10		No	No	–	No	CMD prompt (del)
W7, W10		No	Yes	No	No	CMD prompt (del)

the traces. It is easy to hide traces by deleting LNK files or eventlog entries or by using a tool to clear UserAssist and RecentDocs in the Registry. It is possible to delete entries in the $ObjId index by using the **fsutil** tool, or by deleting files. The latter will still preserve the Object ID attribute in the MFT record, as long as the MFT record is not reused. This is because only a flag in the corresponding MFT record header is changed when deleting a file (Carrier, 2005). It is currently possible to change the $ObjId index file from user space by using fsutil in Windows 10 (not in Windows 7) or by utilizing the correct API when developing new anti-forensic tools. It is not possible to set new OIDs using fsutil if there exists a set of OIDs for the particular file. In order to set new OIDs it is necessary to delete the existing OIDs first, then create new ones. In NTFS there are other system files that will be updated when using fsutil to change the OIDs, for instance the $UsnJrnl have entries that describe the type of change [2, p.394]. Manipulation of OIDs can easily be detected if they do not follow the same format as Windows. If the MFT record SIA created timestamp is manipulated to a future date within another Object ID session, analyzing the previously assigned Object ID will normally detect this manipulation. This because the Object ID identify the boot session it belongs to, and therefore the MFT SIA created date should not be in the time range of a later Object ID boot session.

An interesting question is if all OIDs are only created based on User Activity? The answer depends on how we define user activity. In this study any process that behaves on behalf of a user, as a user agent or a chain of user agents, is user activity (Buchholz and Spafford, 2004). For instance a process is normally executed by a user or the OS. Even though the user started the OS, we do not count automatic OS activity not initiated by the user as user activity. A malicious program is started somehow by a user, not necessarily the local user, and we consider this user activity.

The Object ID index can be used to find all allocated files that have an Object ID. The Object ID keys found in the $ObjId file can also be compared with the unallocated entries in the MFT table which contain an OID. This will indicate that the user did more than just delete the file, and the file should therefore be recovered for further investigation.

Even if users wipe their system drive, the computer used can be discovered by analyzing a previously attached removable NTFS volume. This is because the MAC address is usually contained within the OIDs. If OIDs are created during multiple sessions on different computers, the removable NTFS volume can also yield different boot times for the computers to which it has been attached.

We can not depend on the move flag (least significant bit in the timestamp when read as LE (Parsonage, 2008)) of the Birth Volume Object ID when the target Birth Volume Object ID is 0. In this case other Object ID and Birth Object ID are created, which makes it look as if the file was not moved. In these cases the file can only be connected to a computer using the MAC address found in the Object ID and the Birth Object ID. When a user moves a file from one volume to another, the move flag will only be set if the target Birth Volume Object ID is not zero.

Conclusions and future work

Users will use File Explorer or other software tools to create, open, copy, move and delete files. In the cases in which OIDs are created, it will yield user activity. Even if the system volume is not available, we know that the OIDs are artifacts from some form of user activity. On external drives the $ObjId is one of the very few artifacts found that can yield user activity.

Our experiments using Windows 7 and 10 show that a Birth Volume Object ID is not always created, even if Birth Object ID and Object ID are created. Previous research has documented that Birth Volume Object IDs are created or updated (Parsonage, 2008), but our results show Birth Volume Object IDs with only zeros. This means that we are even more dependent on the MAC address found within the Birth Object ID to connect the computer used to create the OIDs. If an external disk with a NTFS volume is attached while rebooting, our experiments show that $Volume system file is assigned a new Object ID if the existing one is not set. However, we have observed exceptions to this when using USB thumb drives.

Analyzing the $ObjId index is important in order to:

- create timelines
- connect NTFS volumes to one or more computers by using the MAC address found within the Object ID
- select which files to analyze (data reduction or triage)
- detect boot sessions and the order of OIDs creation
- detect MFT created date manipulation

For further work we suggest to determine if correlation with other system files can be used to validate the interpretation of the $ObjId system file. In this context, $UsnJrnl system file [2, p.343] and the $logfile [2, p.340] is known to be useful for event reconstruction. However, the $logfile is normally very small (64 MiB) and the transactions will start overwriting the oldest transactions when necessary (Zareen and Aslam, 2014). This means the NTFS $logfile journal transactions are very volatile and will only document file activity for a particular time range, with that range dependent on the degree of volume activity. It would also be interesting to expand this study by correlating the $ObjId index with other system files in order to see if it is possible to reliably detect what kind of operation created the Object IDs.

More research could be performed on which APIs implement the use of $ObjId system file. We have documented that the IFileOpenDialog API will create OIDs. Even if our work shows similarities between Windows 7 and 10, it also shows differences. This was expected, since programmers change their software tools regularly, and they decide which APIs they want to use in each release. The APIs themselves could also change in the future.

More experiments should be performed to determine what a change of the $Volume Object ID can have on existing OIDs in the $ObjId index. Further experiments should be performed in order to see if adding a partitioning scheme on USB thumb drives will impact the creation of $Volume Object ID.

References

Buchholz, F., Spafford, E., 2004. On the role of file system metadata in digital forensics. Digit. Invest. 1, 298–309. http://www.sciencedirect.com/science/article/pii/S1742287604000829. https://doi.org/10.1016/j.diin.2004.10.002.

Carrier, B., 2005. File System Forensic Analysis. Addison-Wesley Professional.

Carrier, B., 2017. The Sleuth Kit (TSK) is a library and collection of command line tools that allow you to investigate disk images. Available at: https://www.sleuthkit.org/sleuthkit/. . (Accessed 30 August 2018).

Cowen, D., 2018a. Forensic lunch test kitchen 9/13/18. Available at: https://www.hecfblog.com/2018/09/daily-blog-491-test-kitchen-92718.html. https://www.hecfblog.com/2018/09/daily-blog-491-test-kitchen-92718.html. (Accessed 1 January 2019).

Cowen, D., 2018b. Objectidscannerv2. Available at: https://github.com/dlcowen/TestKitchen/blob/master/OBjectIDScannerV2.py. https://github.com/dlcowen/TestKitchen/blob/master/OBjectIDScannerV2.py. (Accessed 1 January 2019).

Garfinkel, S.L., 2010. Digital forensics research: The next 10 years. Digit. Invest. 7, S64–S73. http://www.sciencedirect.com/science/article/pii/S1742287610000368. https://doi.org/10.1016/j.diin.2010.05.009 (the Proceedings of the Tenth Annual DFRWS Conference).

Gl, M., Kugu, E., 2017. A survey on anti-forensics techniques. In: 2017 International Artificial Intelligence and Data Processing Symposium. IDAP), pp. 1–6. https://doi.org/10.1109/IDAP.2017.8090341.

Habben, J., 2018. Ntfs object ids in encase. Available at: https://4n6ir.com/2018/09/20/ntfs-object-ids-in-encase/. (Accessed 18 January 2019).

Leachi, P., Mealing, M., Salz, R., 2005. A Universally Unique Identifier (UUID) URN

Namespace. Available at: https://www.ietf.org/rfc/rfc4122.txt. (Accessed 30 August 2018).
McGrath, N., Gladyshev, P., 2013. Investigating File Encrypted Material Using NTFS $logfile. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 183–203. https://doi.org/10.1007/978-3-642-39891-9_12. https://doi.org/10.1007/978-3-642-39891-9%5c%5f12.
Microsoft, 2001. Will dragging a file result in a move or a copy? Available at: https://blogs.msdn.microsoft.com/oldnewthing/20041112-00/?p=37323. (Accessed 30 August 2018).
Microsoft, 2016. Fsutil objectid. Available at: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/cc788098(v=ws.11. (Accessed 30 August 2018).
Nordvik, R., 2019. Ntfs object id parser. Available at: https://github.com/RuneN007/NTFSObjectIDParser. . (Accessed 7 January 2019).
Parsonage, H., 2008. The meaning of linkfiles in forensic examinations. http://computerforensics.parsonage.co.uk/linkfiles/linkfiles.htm. (Accessed 1 June 2017).
Schicht, J., 2018. Command line $mft record decoder. https://github.com/jschicht/MftRcrd. (Accessed 27 September 2018).
Singh, B., Singh, U., 2016. A forensic insight into Windows 10 Jump Lists, vol. 17. Elsevier - Digital Investigation, pp. 1–13.
Yamazaki, T., 2015. Filetime extractor. Available at: http://www.kazamiya.net/en/fte. . (Accessed 8 January 2019).
Zareen, M.S., Aslam, B., 2014. $logfile of ntfs: A blueprint of activities. In: 17th IEEE International Multi Topic Conference 2014, pp. 305–310. https://doi.org/10.1109/INMIC.2014.7097356.