Forensic analysis of deduplicated file systems

Dario Lanterna*, Antonio Barili

University of Pavia, Via Ferrata, 5, Pavia, Italy

ABSTRACT

Deduplication splits files into fragments, which are stored in a chunk repository. Deduplication stores chunks that are common to multiple files only once. From a forensics point of view, a deduplicated device is very difficult to recover and it requires a specific knowledge of how this technology operates. Deduplication starts from a whole file, and transforms it in an organized set of fragments. In the recent past, it was reserved to datacenters, and used to reduce space for backups inside virtual tape library (VTL) devices. Now this technology is available in open source packages like OpenDedup, or directly as an operating system feature, as in Microsoft Windows Server or in ZFS. Recently Microsoft included this feature in Windows 10 Technical Preview. Digital investigation tools need to be improved to detect, analyze and recover the content of deduplicated file systems. Deduplication adds a layer to data access that needs to be investigated, in order to act correctly during seizure and further analysis. This research analyzes deduplication technology in the perspective of a digital forensic investigation.

© 2017 The Author(s). Published by Elsevier Ltd on behalf of DFRWS. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Introduction

The architecture evolution of deduplicated file systems has been mature for production environment since many years, but now it is ready for office and consumer environment.

Digital forensic analyses are frequently required for many types of crimes, not only for cybercrime. In most cases, the practitioner has to extract some files from file systems, to restore some other from backups, and to analyze a bunch of digital media as USB disks, SD cards, and NAS storages. Analyses involving datacenters are done with the collaboration of data center staff and technology and deduplication is handled transparently. Now that this technology is arriving at a consumer level (Windows 10 Technical Preview-2016), a higher level of awareness is required. Seizing an usb disk of some TB, without knowledge of the presence of deduplicated volumes, makes it difficult and sometimes impossible to extract data.

The use of a deduplicated file system is transparent to the user, and gives optimal results in terms of space saving. The saving improvement estimated from Microsoft (El-Shimi et al., 2012) using basic chunking is of 25.82% for Office-2007 documents (docx), and 9.96% for PDF. These values are calculated using GFS-US dataset.

This analysis explains how deduplication works, how we can identify a particular type of deduplication implementation, and how to reconstruct files for a specific configuration. Traditional data carvers do not recognize the presence of a deduplicated file system. Microsoft implements this feature as an extension of NTFS, adding a reparse point attribute in the file entry. Reading the NTFS Master File Table (SMFT) of a system with deduplication enabled, a forensic tool can show files and folder structures, but cannot extract the files’ content. A similar problem was present the first time NTFS introduced files and folders compression.

Previous work

Deduplication is studied from the point of view of algorithms and their efficiency (Muthitacharoen et al., 2001; Min et al., 2011) and a brief introduction to new storage technologies in a forensics perspective is explained in this article (Carlton and Matsumoto, 2011). The authors indicate the need for thorough study using experimental data, and physical acquisition and underline the importance of markers that help to recognize storage technologies. Deduplication is used in smartphone memory analysis to detect duplicated pages (Park et al., 2012), because flash memory pages are not deleted and rewritten when data is modified, but a new page is created according to Flash Translation Layer (FTL) algorithm (Park et al., 2012), and the updated content is saved in it (Harnik et al., 2010). Deduplication is also considered a useful technology to reduce space needed to archive digital evidence (Neuner et al., 2015; Neuner et al., 2016).
Deduplication

Deduplication is a process that works in order to reduce duplication of data on a device. Data deduplication is used in backup and archive processes, while network deduplication is used to reduce network bandwidth usage in some applications. Deduplication can be done at file-level (SIS Single Instance Storage) or at block-level.

The deduplication process is considered in-line if it is done before saving data on the device, while is considered a post-process, if data is first stored on a device and then deduplicated, according to some parameters as file age, file type and file usage. An example of inline deduplication is OpenDedup, while an example of post-process is the deduplication engine integrated in Microsoft Windows Server (2012) (and 2016) (El-Shimi et al., 2012).

A deduplicated file system acts the deduplication process against the whole file, to discover duplicated parts. The procedure (Fig. 1) splits the file into fragments called chunks and for each chunk a hash is computed (OpenDedup uses a non-cryptographic hash algorithm named Murmurhash3 (Yamaguchi and Nishi, 2013; Appleby), while Microsoft uses a cryptographic algorithm SHA256 (NIST, 2012)). All new hashes are stored in a database, and the relative chunk is stored in a chunkstore; a pointer to the position in the chunkstore is saved together with the hash. The original file is transformed in a sequence of hashes; each hash is linked to the corresponding chunk. The procedure, that reconstructs original files after deduplication, is called rehydrating. Chunks that are common to multiple files are saved only once. If a very frequent chunk is lost, many files cannot be fully rehydrated. Different techniques are possible to avoid this problem. Microsoft stores multiple copies of the chunks that recur very often; OpenDedup uses SDFS file system that may use multiple nodes and spread each chunk inside more than one node.

The chunks may have a fixed length in the order of some kB (usually 32 kB–128 kB) or variable length. The use of fixed length chunks simplify hash computing, and storage is simple to organize. Using fixed length chunks, a little difference between two files generates a different set of chunks with different hashes: for example later versions of documents or source code. Variable length algorithms extract chunks using fingerprints in the text, in this case little additions to a file affect only the chunk that contains the addition. Fingerprints identification is done using Rabin fingerprints algorithm (Rabin, 1981). This algorithm uses a sliding window of a fixed number of bytes and computes a value (fingerprint) using polynomials. Using specific patterns of fingerprint values, deduplication systems cut original files in chunks. In this way, it is possible to extract common chunks in very similar files isolating the changing parts.

Deduplication is present in many products available for production usage:

- Data Domain File system (DDFS) (Zhu et al., 2008) is used in appliance of EMC DataDomain family;
- Zettabyte File System (ZFS) an open source file system originally designed by Sun Microsystems, now Oracle Corporation. ZFS implements deduplication from 2009;
- B-tree file system (BTRFS) stable from August 2014 can enable out-of-band data deduplication;
- LiveDFS (Ng et al., 2011) implements inline deduplication and is designed for virtual machine storage repositories;
- OpenDedup based on SDFS is an inline file system used for backup purposes;
- Microsoft Windows 2012 file system is a post-process deduplication engine (Debnath et al., 2010; Introduction-to-data-deduplication, 2012).

Each implementation has proper strategies to reduce impact on system performance, and to reduce usage of memory and CPU.

Analysis

Low level file system analysis

The analysis of a real system allows acquiring and recognizing details, about characteristics of these file systems. The analysis of the structure and the acquisition of artifacts give a knowledge of how to operate. The elements analyzed here are present in all deduplicated file systems with different naming conventions and internal organizations.

The scope of the analysis is to detect a series of parameters needed to reconstruct data. Using these parameters, it is possible to infer configurations of the original systems and to run an instance of the original application and recover data automatically.

The knowledge of the structure of deduplicated file systems may help improve off-line tools. Off-line tools, like FTK Imager by AccessData or Autopsy by Basis Technology, can navigate many different file systems, but considering W2012 deduplication these tools can only navigate files and folders structure, but they cannot extract the content of deduplicated files and the same happens with OpenDedup. These tools installed on a W2012 system, cannot open deduplicated files, even if they are accessible from file system.

OpenDedup

The first implementation analyzed in this paper is the SDFS file system, used in OpenDedup. OpenDedup allocates all necessary files in a regular file system using a software interface called File system in Userspace (FUSE). By means of FUSE it can write a virtual file system allocated in user storage space. When SDFS is mounted it operates as a regular file system, and deduplicated files are

![Fig. 1. Deduplication process.](image)
reached through their mount point. Unmounting the file system, it is possible to access the underlying structure (Fig. 2). The basic structure of SDFS (Fig. 2) is replicated for each volume.

The volume under analysis is called “dedup-test”; to navigate this file system, the starting point is the folder “dedup-test/files” (Fig. 3). In this folder, there is a replica of the folders and files structure of the mounted file system; each file and folder has metadata similar to the files in the mounted file system. The attribute relative to file size is not referred to the original deduplicated file, while the content of the file is relative to the deduplicated storage structure. Files in this folder contain the pointers needed to support the deduplication structure. These files are pointer files.

Analyzing one of the pointer files (ex.: The Complete Works of William Shakespeare.txt) the first bytes report the creator, in our case OpenDedup (address 0x08: “org.opendedup.sdfs.io.MetaDataDupFile”) and the version is at the end of the file (address 0x12E: “3.1.9”). The first two bytes in each file pointer (Table 1) (stream_magic: 0xACED) are a typical marker for JavaSerialization protocol. This indicates that this file is a serialization of java data structure.

Inside this file there is the size of the original file (address 0x4A:00:00:00:00:00:00:00:00:55:48:81 → 5,589,889 bytes — Table 2). Using this attribute, the metadata related to the file system are now complete: owner, permission, path and filename are reported in the folder “dedup-test/files” while the size is inside the file pointer. Using pointers, we can rehydrate a file to its original content.

To reconstruct the file we need a link to the structure that contains the sequence of chunks. Inside the file pointer there is a unique-identifier (address 0x68: “$aac2f972-56e4-4fd5-9e1c-8dddec187195” — Table 3) that points to further elements in the structure.

In the folder “dedup-test/ddb” (Fig. 4) there is a set of two characters folders, these are the first two characters of the unique identifiers present in the file pointers. In this case, we have to look in the folder “dedup-test/ddb/aa”: inside this folder there are the folders that map files relative to unique-identifiers starting with “aa”. To access the map for the file under analysis the right folder is “dedup-test/ddb/aa \aac2f972-56e4-4fd5-9e1c-8dddec187195”. The folder contains a file named “aac2f972-56e4-4fd5-9e1c-8dddec187195.map”. This file contains all the chunks hashes of the original file chunks. These hashes are in the proper order and each hash can be used as a pointer into the chunkstore.

The hashes sequence and the chunkstore are two elements common in all deduplication systems. The default hash algorithm in OpenDedup is murmur hash (mmh3) (Appleby) but this can be changed by a configuration parameter and other algorithms can be specified; Microsoft and DataDomain use SHA1/SHA256.

Table 1

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0008 AC ED 00 05 73 72 00 27</td>
<td>~4..Sr.!</td>
</tr>
<tr>
<td>0x0008 6F 72 67 2E 6F 70 65 6E</td>
<td>Org.Open</td>
</tr>
<tr>
<td>0x0010 64 65 64 75 70 70 73 64</td>
<td>Dedup.sd</td>
</tr>
<tr>
<td>0x0018 66 73 2E 6F 6F 28 4D 65</td>
<td>fs.io.Me</td>
</tr>
<tr>
<td>0x0020 74 61 64 61 74 61 44 65</td>
<td>taDataDe</td>
</tr>
<tr>
<td>0x0028 64 75 70 46 69 6C 65</td>
<td>dupFile</td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0048 xx 00 00 00 00 00 55 4B</td>
<td>...Saa2</td>
</tr>
<tr>
<td>0x0050 81 xx xx xx xx xx xx xx</td>
<td>Eic-8dd</td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0068 00 00 00 24 61 61 63 32</td>
<td>...Saa2</td>
</tr>
<tr>
<td>0x0070 66 39 37 32 2D 35 36 65</td>
<td>F972-56e</td>
</tr>
<tr>
<td>0x0078 34 2D 34 66 66 64 35 2D 39</td>
<td>4-4fd5-9</td>
</tr>
<tr>
<td>0x0080 65 31 63 2D 38 64 64 64</td>
<td>Elc-8dd</td>
</tr>
<tr>
<td>0x0088 65 63 31 38 37 31 39 35</td>
<td>ec187195</td>
</tr>
</tbody>
</table>

Fig. 2. SDFS volumes structure.

Fig. 3. SDFS files structure.

Fig. 4. SDFS ddb structure.

Fig. 5. SDFS chunkstore structure.

Mmh3 is a non-cryptographic hash algorithm simple to compute, with a reduced need of memory and computing power; it has a low collision rate, and so it is suitable for inline deduplication. The procedure computes mmh3 with a 128 bit length and with a “seed” (0x192A → decimal 6442), the value of which is written in
the first two characters of each map file. The simplest way to verify mmh3 hash computing is by fixing the chunk length, splitting the file using chunk length and then computing mmh3 (Appleby) using the “seed” 6442.

The sequence (Table 4) of hashes can be used to get the chunks. To discover where each chunk is saved some other steps are needed. In folder “\deduptest\chunkstore\[db]” there is a file called hashstore-sdfs-UNIQUEID (example: hashstore-sdfs-97035D0A-E223-D298-AABA-6EE96282B8A6) that contains as set of (key, value) pairs. The keys are the hashes, while the values are pointers in the chunkstore folder structure (Fig. 5). This file contains all the hashes computed for all the files in the deduplicated file system, and operates as an index.

The use of these pointers in the chunkstore structure (Table 5) requires a simple transformation of the pointer from hexadecimal to decimal (ex: 0x162058 7D 6E 75 5F 20 A3 39 F5 requires a simple transformation of the pointer from hexadecimal to signed decimal (ex: 0x7D6E755F20A339F5 = 9038290553609075189). The pointer for the chunk we are looking for, \deduptest\chunkstore\chunks\9038290553609075189.map, is again (key, value): the key is the length of a start marker FFFF FFFF, and the value is the chunk.

The structure of this file (Table 6) is again (key, value): the key is as usual the hash, while the value is the offset in the file “9038290553609075189.map”. This file contains all the hashes and the structure is (key, length, value); the length is the length of a start marker FFFF FFFF, and the value is the chunk.

Table 4
Hashes sequence.

<table>
<thead>
<tr>
<th>Hexadecimal</th>
</tr>
</thead>
</table>
| 0x00000000 19 2A 02 00 00 00 00 00 ...
| 0x000100 00 00 00 35 00 00 00 ...
| 0x000110 01 2D FA E1 3F CE 15 51 ...
| 0x00011A B1 9A A7 55 28 A0 89 99 ...
| 0x00011B 41 00 00 00 00 00 00 ...
| 0x000160 00 35 00 00 00 00 00 00 00 B7 D7
| 0x00016B C2 E3 4B C9 85 T0 C0 1A ...
| 0x000170 34 CE F1 B4 28 EF 00 FE |

Table 5
Chunkstore pointer.

<table>
<thead>
<tr>
<th>Hexadecimal</th>
</tr>
</thead>
</table>
| 0x162048 2D FA E1 3F CE 15 51 B1 ...
| 0x162050 9A A7 55 28 A0 89 99 41 ...
| 0x162058 7D 6E 75 5F 20 A3 39 F5 |

Table 6
Chunkstore.

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Text</th>
</tr>
</thead>
</table>
| 0x0ADD00 00 00 00 10 2D FA E1 3F | ....-~å?
| 0x0ADD00 CE 15 51 B1 9A A7 55 28 | ↓.QaÅU |
| 0x0ADD20 A0 89 41 00 00 10 04 | åMªa....
| 0x0ADD20 FF FF FF FF BB BF S4 | yyyy1-qt
| 0x0ADD20 68 65 20 50 72 6F 6A 65 | be Proje
| 0x0ADD20 63 74 20 47 75 74 65 6E | et Outen |
| 0x0ADD20 62 67 2 65 72 0 45 42 6F | berg EBo |

Windows 2012 deduplication

The Windows 2012 Deduplication (W2012Dedup) (Patentscope) is a feature of the file system, while OpenDedup was a file system in userspace. This implies that the analysis has to access the underlying structure of the file system. To analyze the file system, the tools used are FTK Imager and Autopsy; by means of these tools, it is possible to access all the elements of the NTFS structure.

The file system stores all deduplication data under the “System Volume Information” (Fig. 6); this hidden folder contains the chunkstore structure. The chunkstore contains three elements, the Stream container, the Data container and the Hotspot container. The first two elements are common in deduplication. The Stream contains the hashes sequences; the Data contains the chunks, while the Hotspot is an element added by Microsoft to store most common or frequently used chunks; in this last container there is a controlled level of redundancy.

The analysis of a Windows 2012 file system starts from the Master File Table ($MFT$) and $MFT$ information are stored in little endian. The $MFT$ entry relative to a deduplicated file contains information about chunkstore. These data are saved in a “Reparse Points” attribute ($REPARSE_POINT = 0xC0$).

The function of the “Reparse Point” attribute is to act as a collection of user-defined data. Microsoft or third party software can use this attribute for specific applications. When a reparse point is present, the system needs a specific filter to parse this attribute.

Reparse Point (Table 7) starts with the NTFS attribute type 0xC0; in our example, the full length of this section is 0x00A0. The length of the original file is written at the relative offset 0x28 (Len 4 bytes); at offset 0x38 (Len 16) there is the unique identifier of ChunkStore ([2EE490E5-44F0-4F9A-AD59-D6D8A2B5652C],.ddp). Inside this folder, we have the “Data” and “Stream” folders. Inside the Data folder, there are .ccc files that are chunks containers, while inside the Stream folder the .ccc files contain the hashes sequences for the deduplicated files. At offset 0x78 (Len 30) there is the sequence of bytes that are the stream header; this value identifies the stream of a particular file in the stream container.

In the Stream folder, a .ccc file has three type of sections: “Cthr” called file header, “Rrtl” or redirection table, “Ckhr” or stream map element. The syntax of the file is:

- `<Stream Container> ::= <file header> < redirection table>`
- `<stream maps> ::= <stream map> <stream maps> | <stream map>`
- `<stream map> ::= <stream header> <metadata> <hash values> | <hash values>`

The data we are looking for is in Ckhr (0x 43 68 68 72) sections. Each Ckhr section (Table 8) contains the full sequence of hashes relative to a file also called “stream map”; each section reports the
When a file is deleted, the $MFT$ entry $\$REPARSE\_POINT$ is cleared, but the chunk hashes sequence, and the chunks in the chunk container are preserved until the first “garbage collection” job runs.

The chunks may be compressed, depending on the system configuration, and some types of file are excluded from compression because they already contain compressed data. The compression used (sometimes called LZNT1+) is very similar to LZNT1 (introduction-to-data-deduplication, 2012; MS-XCA, 2015). LZNT1 is a Microsoft algorithm inspired to LZ77 (Ziv and Lempel, 1977). The difference between LZNT1 and this compression algorithm is that the flag bytes are 32 bits long (4 bytes) instead of 16 bits (2 bytes) used by LZNT1. The syntax is the same:

```
<compressed chunk> ::= <Flag group>
<Flag group> ::= <Flag data> <Flag group> | <Flag data>
<Flag data> ::= Flag-byte <data block> (1-32)
<data block> ::= Char | Len-displacement
```

There is no official documentation about this element, but this compression algorithm seems an evolution of LZNT1.

W2012Dedup hash algorithm outputs values 256 (32 bytes) bits long. According to documentation (Microsoft Open Specifications Program) many Microsoft protocols use SHA-1, for example Windows Distributed File System (DFS) replication (MS-FRS2) and Remote Differential Compression Algorithm (MS-RDC). In this case, the length of the hash value (256 bits) indicates another algorithm. To verify this hypothesis we tested some algorithms but without success: the W2012Dedup hash algorithm is a Microsoft algorithm inspired to LZ77 (Ziv and Lempel, 1977). The difference between LZNT1 and this compression algorithm is that the flag bytes are 32 bits long (4 bytes) instead of 16 bits (2 bytes) used by LZNT1.

Forensic analysis

The seizure of storage devices containing a deduplicated file system is an activity that requires some care. During a seizure, we need to check the presence of deduplication in storage devices and if it is present, there are a few solutions applicable. The first is to seize the whole system and not only the deduplicated storage. The second is to conduct a live forensic analysis and extract the documents of interest for investigation (if known) on-site. The third method is to write down all installation details and replicate the system.

```
```
the same configuration in laboratory, because to recover a deduplicated volume we can mount it using an operating system that runs the same configuration as the original installation.

However, if during seizure no check was done for the presence of deduplication, we have to extract information directly from storage devices. In this case, recovering of the volume requires a little more effort. The first step is to recognize the file system type; the second is to infer the configuration parameters. We must pay specific attention at data carver's results, because at the date we wrote this article, popular data carvers do not recognize the presence of a deduplicated file system, and do not know how to rehydrate original files. This article is a beginning of investigation of these file systems, to improve awareness about the problem.

OpenDedup

The previous explanation of how this file system works, gives us the way to recover it using the chunks repository and the hash sequences. Usually a deduplicated file system is used to support backup of huge quantity of data. The idea to manually rehydrating a file system is nonsensical, but a clear understanding of the process is the basis to create procedures to automate the process. The direct analysis of the storage support is reserved to recovering of corrupted volumes. The fundamental elements in recovering procedure are the chunkstore and the relative hash sequences. To see if the volume under analysis contains a working file system, we can analyze the structure of the deduplicated file system. It allows checking integrity of data; the integrity check is done using the information available in the chunkstore. The chunkstore contains a set of ordered pairs (hash, chunk), and we can use the algorithm murmurhash3 and the “hash-key” (default 0x6442 in case of default configuration) to verify integrity of chunks. To verify that all the chunks are present, we must use the hash sequences. This procedure gives a granularity of check corresponding to the chunk size.

If we have a well-functioning OpenDedup volume, we can install the same version of the file system and configure it to mount the volume under analysis. The configuration requires parameters inferable from data present in the volume. The basic parameters are chunk type and size, the hash algorithm, the “hash-key”, the position of the hash-db and of the chunkstore. To identify chunk type and size we can analyze the chunks length: if all chunks have the same size, the chunk type is “fixed-size” and the chunk size is easily computed, while if chunks have different length the chunk type is “variable-size” and “min-size” and “max-size” need to be estimated analyzing all the chunks. The hash-key is in the first byte of all the map files. The hash-type (or hash algorithm) can be detected using the length of the hash value, the hash value, the chunk content, and the hash-key. We must compute the hash value of the chunk using the possible algorithms and identify the right one; the default algorithm is murmurhash3 (Appleby).

Windows 2012 R2

Windows 2012 uses a post-process deduplication method. Files are first stored in the file system as regular files, and only after a configured period (parameter: fileMinimumAge) are processed for deduplication. After deduplication, the file system removes the original files. However, until the disk area is overwritten, the artifacts of deleted files remain on the volume. Since they were regular files, they can be recovered using a data carver.

W2012 does not deduplicate all the files: it filters files according to “excludeFileExtensionsDefault” configuration parameter, that indicates which file extensions are excluded. The excluded files are saved as regular files and no deduplication is applied. Other files are deduplicated and stored in the volume as previously explained.

W2012 stores chunks in a compressed form, but compression is not applied to all files, there is a configuration parameter that excludes the compressed formats (parameter: noCompressionFileExtensions, default values: asf, mov, wma, wmv, ace, arj, bkh, bz2, cab, gz, gzip, hpk, lha, lzh, lzx, pak, pit, rar, sea, sit, tgz, z, zip, zoo). The excluded files are deduplicated, but chunks are not compressed. These files can be recovered concatenating all chunks as they are in the chunkstore, following the order specified in the stream map; no deflate process is required after chunks extraction.

The simplest method, to recover a well-functioning W2012 deduplicated volume, is to mount it on a system with the same operating system version with the deduplication engine enabled.

Tools like FTK Imager or Autopsy can analyze many different file systems, reading directly the file system data structure. However, when you try to read a deduplicated file system, it starts from $MFT of the deduplicated volume, reads all the entry, shows files and folders with their metadata and when it tries to inspect the content of the files, they result empty. This happens because these tools do not know how to use reparse point information. Therefore, in case of a damaged device, we must recover the fundamental files that are the chunk-container and the stream-container; these two elements are the bearing structure of the chunkstore. Then, following a stream map, we can concatenate chunks to compose a whole file. When a file is rehydrated, if the $MFT is available, we can recover the file name and metadata, otherwise we can analyze header and structure of the file and recognize the file type.

Considering a non-deduplicated volume, when we delete a file, a data carver can recover the whole file, until the allocated space is overwritten. A deduplicated volume instead splits files in chunks, and stores each chunk in a repository. When a file is removed from a deduplicated volume, the entry in $MFT is immediately removed, but the stream map and the chunkstore remains unchanged. Therefore, immediately after deletion, it is possible to recover a file. A deduplicated volume runs a optimization process regularly, but a “regular” optimization has no effects on stream maps and chunk repositories. Only when a garbage collection (GC) job runs, it removes the chunks that are part of deleted elements from the chunkstore. A “regular” GC deletes only part of the unreferenced chunks from the chunkstore, while a “full” GC eliminates all traces of deleted files in deduplicated volume structure. Nevertheless, analyzing unallocated space after a GC we can find artifacts left by this process. During GC, the system creates a new version of the stream container and the chunk container, then deletes the previous version of the stream container and the chunk container files, but they remain for a while in the file system as artifacts. We can recover fragments of stream map and chunk container, and sometimes whole files. To recognize stream-container and chunk-container, we can use their internal structure reported in Tables 10 and 11. The particular structure of files that support a deduplicated file system gives high confidence to the recovered file, because the stream map is a sequence of hashes, and so it is possible to verify the completeness of the chunks repository.

The importance of Hash-db

Suppose you recover a chunk container, without the hash sequences, and the chunks are not compressed. Without the knowledge of chunks concatenation sequence, it is impossible to do an accurate reconstruction because of the deduplication algorithms used to create chunks. An efficient chunking strategy uses a system based on the Rabin algorithm (Rabin, 1981). This algorithm locates the point where to break a file creating chunks; the localization of cut points happens where the original file shows a predefined “fingerprint”. When systems use this algorithm to process files
containing documents like contracts or invoices, the resulting chunks are very similar, because these documents are usually based on models, and their chunks can be concatenated to generate files never existed in the original file system. The existence of a flawless hash sequences container is the only way to be sure of the accuracy never existed in the original

Table 10
Stream container format.

<table>
<thead>
<tr>
<th>Address</th>
<th>Hexadecimal content</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000</td>
<td>43 74 68 72 01 04 04 01 Cthr....</td>
</tr>
<tr>
<td>0x000020</td>
<td>last stream entry</td>
</tr>
<tr>
<td>...</td>
<td>43 74 68 72 01 04 04 01 Cthr....</td>
</tr>
<tr>
<td>0x001000</td>
<td>52 72 74 6C 01 03 03 01 Rrtl....</td>
</tr>
<tr>
<td>0x002000</td>
<td>52 72 74 6C 01 03 03 01 Rrtl....</td>
</tr>
<tr>
<td>0x003000</td>
<td>52 72 74 6C 01 03 03 01 Rrtl....</td>
</tr>
</tbody>
</table>

Table 11
Data container format.

<table>
<thead>
<tr>
<th>Address</th>
<th>Hexadecimal content</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000</td>
<td>43 74 68 72 01 04 04 01 Cthr....</td>
</tr>
<tr>
<td>0x000020</td>
<td>last stream entry</td>
</tr>
<tr>
<td>...</td>
<td>43 74 68 72 01 04 04 01 Cthr....</td>
</tr>
<tr>
<td>0x001000</td>
<td>52 72 74 6C 01 03 03 01 Rrtl....</td>
</tr>
<tr>
<td>0x002000</td>
<td>52 72 74 6C 01 03 03 01 Rrtl....</td>
</tr>
<tr>
<td>0x003000</td>
<td>52 72 74 6C 01 03 03 01 Rrtl....</td>
</tr>
<tr>
<td>end of</td>
<td>...</td>
</tr>
<tr>
<td>file</td>
<td>FF FF FF FF FF FF FF FF</td>
</tr>
</tbody>
</table>

Fig. 7. Recovering using chunks without hashes sequence.

property, you can concatenate chunks, to create a new well-formatted file, but this file was not present in the original volume.

The procedure we used to generate files to demonstrate this hypothesis is very similar to the procedure used to create documents in a company, where employees start always from the same document model to create new documents, and then they modify only part of header and part of the body (examples are invoices and contracts).

The problem exposed, enforces the rule that we must have the hash sequences to rehydrate files present in the chunkstore. The hash sequences are a crucial element of a forensics acceptable reconstruction of a deduplicated file system.

Conclusion

New storage technologies need to be investigated from a forensic point of view, because manufacturers rarely give detailed documentation about low-level implementation. Storage technologies knowledge is central in digital forensic analysis and this work gives a first look inside deduplication implementations.

This paper addresses deduplication technologies, it analyses a post process deduplication (Microsoft Windows, 2012) and an in-line implementation (OpenDedup). The knowledge of deduplication implementations helps to identify the presence of this kind of file system on devices, and to recover files and folders content from them. In case of damaged devices or file systems, this work proves that, without the structures containing the hash sequences, the file reconstruction can generate never-existed files. The hash sequences are crucial for a “forensically sound” recovery.

Future works will analyze from a forensic point of view other implementations of storage deduplication and storage technology.

References


Ng, Chun-Ho, Ma, Mingcao, Wong, Tsz-Yeung, Lee, Patrick P.C., Lui, John C.S., December 2011. Live deduplication storage of virtual machine images in an


