THE JOURNAL OF DIGITAL FORENSICS, SECURITY AND LAW

Journal of Digital Forensics, Security and Law

Volume 14 | Number 3 Article 1

9-1-2019

Fast Forensic Triage Using Centralised Thumbnail Caches on Windows Operating Systems

Sean McKeown
Napier University, s.mckeown@napier.ac.uk

Gordon Russell
g.russell@napier.ac.uk

Petra Leimich
Napier University, p.leimich@napier.ac.uk

Fast Forensic Triage Using Centralised Thumbnail Caches on Windows Operating Systems

Cover Page Footnote

This research was supported by a scholarship provided by Peter KK Lee.

FAST FORENSIC TRIAGE USING CENTRALISED THUMBNAIL CACHES ON WINDOWS OPERATING SYSTEMS

Sean McKeown, Gordon Russell, Petra Leimich
School of Computing
Edinburgh Napier University, Scotland
{S.McKeown, G.Russell, P.Leimich}@napier.ac.uk

ABSTRACT

A common investigative task is to identify known contraband images on a device, which typically involves calculating cryptographic hashes for all the files on a disk and checking these against a database of known contraband. However, modern drives are now so large that it can take several hours just to read this data from the disk, and can contribute to the large investigative backlogs suffered by many law enforcement bodies. Digital forensic triage techniques may thus be used to prioritise evidence and effect faster investigation turnarounds. This paper proposes a new forensic triage method for investigating disk evidence relating to picture files, making use of centralised thumbnail caches that are present in the Windows operating system. Such centralised caches serve as a catalogue of images on the device, allowing for fast triage. This work includes a comprehensive analysis of the thumbnail variants across a range of windows operating systems, which causes difficulties when detecting contraband using cryptographic hash databases. A novel method for large-scale hash database generation is described which allows precalculated cryptographic hash databases to be built from arbitrary image sets for use in thumbnail contraband detection. This approach allows for cryptographic hashes to be generated for multiple Windows versions from the original source image, facilitating wider detection. Finally, a more flexible approach is also proposed which makes novel use of perceptual hashing techniques, mitigating issues caused by the differences between thumbnails across Windows versions. A key contribution of this work demonstrates that by using new techniques, thumbnail caches can be used to robustly and effectively detect contraband in seconds, with processing times being largely independent of disk capacity.

Keywords: digital forensics, triage, image comparison, image processing, known file analysis, image thumbnails, cryptographic hashing, perceptual hashing

1. INTRODUCTION

The field of digital forensics plays an important role in the modern justice system due to the prevalence of digital devices and digital information in everyday life. However, an increasingly heterogeneous digital environment with increasing volumes of data has seen an end to the *Golden Age* of digital forensics (Garfinkel, 2010). This trend has resulted in multi-year backlogs for law enforcement agencies across the world, in the worst cases reaching up to four years before evidence is processed (Lillis et al., 2016). Such backlogs may damage the course of justice, with offenders being frequently awarded reduced sentences due to time waited, and suicides being committed in some cases (Roussev et al., 2013). It is therefore vital that the discipline of digital forensics adapts to the changing times and allows for digital forensics to be performed much faster, while still retaining acceptable accuracy levels.

In order to address the problem of data volume, new tools and procedures must be developed which place an emphasis on performance, rather than simply correctness (Roussev et al., 2013), though the latter property remains important. A number of solutions have been suggested, such as random sampling, parallelisation, or applying techniques from fields, such as Data Mining, Information Retrieval, and Machine Learning (Beebe, 2009). However, a review of such solutions in Quick and Choo (2014) found that there are still several areas of weakness, necessitating further work in the areas of data reduction and triage. A more recent survey of international practitioners dealing with indecent images of children (Franqueira et al., 2018) indicated that any subsequent advances have not had a significant impact on forensic turnarounds, with reports that processing times are still very long and in some cases analysis of the evidence can take up to 2 months.

This paper uses the centralised thumbnail caches found in Windows operating systems since Vista to perform fast contraband detection. Windows generates image previews by downscaling an image to create a smaller version of the image, a thumbnail, which is then stored in a cache for later use. These thumbnails are typically displayed when utilising a file system browser, such as Windows Explorer. Thumbnail caches are relatively small, irrespective of the storage capacity of the device. Using this approach, only a few hundred megabytes of data may need to be inspected to perform triage on a device, potentially allowing contraband to be detected within large storage devices within a matter of seconds.

The contributions of this paper are:

- The first analysis of the pixel and binary differences between thumbnails produced by various Windows versions, showing that a single cryptographic hash for a thumbnail is insufficient, which has not been previously discussed in the literature.

- An approach for automatically replicating the exact thumbnail binaries used by Windows, the benefits of which are two-fold: *i)* Thumbnails and cryptographic hashes can be deterministically generated for different versions of Windows from a single source image, allowing for wider detection of contraband. This is particularly important as images produce different thumbnails between Windows versions. *ii)* Future experimental work on Windows thumbnail caches can be automatically scaled to large datasets.

- The creation and evaluation of a rapid triage approach which utilises cryptographic thumbnail hashes for fast contraband detection.

- A novel analysis of the potential to use perceptual hashing algorithms to detect contraband in thumbnail caches across different Windows releases, so as to tolerate image compression and thumbnail variations between platforms.

Experiments are carried out primarily on the Windows 10 operating system using the Flickr 1 Million image dataset (Huiskes et al., 2010). Results show that thumbnail caches comprising tens of thousands of images can be acquired and analysed in seconds, allowing investigators to perform rapid forensic triage.

2. RELATED WORK

2.1 Digital Forensic Triage

Roussev and Quates (2012) suggest that the solution to the problem of digital forensics scalability lies in more sophisticated processing techniques, with inexpensive initial assessments being used to gain an understanding of the data prior to committing to more intensive analyses. The authors present a case study of the effectiveness of a bytewise similarity hashing tool, *sdhash*, to correlate evidence across heterogeneous devices. Two data streams are generated from each storage device, the first to generate a forensically sound copy of the disk, with the second being used to generate similarity hashes, allowing for data to be processed as it is acquired. File system overheads are avoided by reading

raw data directly from the disk. While effective, this method requires the use of powerful, multiprocessor, workstations in a lab environment.

Roussev et al. (2013) develop a parallel processing model in an attempt to process data as fast as it is read from disk. Different analyses are handled by a variety of worker nodes, calculating traditional cryptographic and similarity hashes, parsing windows registry and metadata information, indexing text, and decompressing files. The authors note that only traditional cryptographic hashing was computationally inexpensive enough to be performed at disk speed. However, approaches which use small amounts of data, such as registry analysis, are fast enough in practice. It was determined that a typical 8-core workstation is insufficient to do this processing in real-time, necessitating either drastically more computational power, or data reduction techniques.

The *bulk_extractor* tool, described by Garfinkel (2013), uses a heavily parallelised processing model to extract key pieces of information from a disk data stream. This is achieved by implementing multiple scanners, which process images, documents, and other textual data to extract artefacts such as email addresses, telephone numbers, and credit card numbers. Generated histograms provide a visual overview of potentially important information, such as the most frequently occurring email addresses.

While the above methods focus on processing an entire disk, Young et al. (2012) propose reducing the amount of data to read from a disk by means of random sampling. A sample of disk sectors is chosen randomly, with each sector being cryptographically hashed and compared to a database of known sectors for contraband detection. This approach was later expanded by Penrose et al. (2015) to use Bloom filters to reduce memory overheads and allow execution on legacy equipment. Their experiments show that the processing time depends on sample size and computer specifications, ranging from seconds on modern machines with Solid State Drives (SSDs), to less than an hour in most cases with traditional hard disks. By selecting an appropriate number of samples, as little as 4MiB of data can be detected with very high probability on a large disk in a short time.

Grier and Richard (2015) take a different sampling approach which makes use of investigation-type specific filters for acquiring evidence from a digital device. By processing file system metadata, key areas of the evidence can be identified and selectively acquired, with analysis taking place on this evidential subset. This approach was shown to reliably capture a large portion of the relevant evidence while greatly reducing the amount of data needing to be captured from a device.

2.2 Centralised Thumbnail Caches In Windows

In addition to adapting to the increasing volumes of data, digital forensics research must also investigate changes in the structure of the data, particularly as new Operating Systems are released.

Morris and Chivers (2011a) note that a paradigm shift for storing thumbnails occurred between Windows XP and Windows Vista, with a centralised thumbcache structure for each user replacing the thumbs.db files in each directory. This means that instead of thumbnails being found in the same directory as their source files, they are now located in a per-user store at the path: `[Drive]:/Users/[Username]/AppData/Local/Microsoft/Windows/Explorer`. Separate database files store thumbnails of various sizes, named thumbcache_xxx.db, where xxx specifies the maximum thumbnail dimension for each side. For Windows Vista and Windows 7, these are 32×32, 96×96, 256×256, and 1024×1024, though thumbnails need not be square. Each cache entry contains some metadata and the thumbnail image itself, which is stored in JPEG, BMP, or PNG format. Cache entry IDs are mapped back to their source file via the thumbcache_idx.db file. Images viewed with the Windows Explorer preview pane generate 1024 pixel thumbnails unless the images are small enough to fit a smaller thumbnail size. 256-pixel thumbnails correspond to extra-large icons, with smaller sizes being used for smaller previews. Thumbnail generation is not limited to images, with documents such as PDFs and file

system directories also having thumbnail preview options. Directory previews look like an open folder with multiple pages, each corresponding to an item in the directory. Viewing a directory icon may trigger thumbnail creation for files within the directory without the user previewing them directly. Additionally, entries are created for files on removable media in the centralised thumbnail cache. Thumbnails may be removed from the system using the Disk Clean-up utility built-in to Windows, and thumbnail generation can be disabled using group policies.

Windows 8 expands on the thumbnail dimensions included in previous versions, adding 16×16, 48×48, and WIDE thumbnails to support additional start menu functionality (Quick et al., 2014). Additionally, iconcache_x.db files were added, with corresponding dimensions matching all thumbcache files.

Parsonage (2012) explores the behaviour of thumbnail generation on Windows Vista and Windows 7 further, finding that legacy thumbs.db files are still being generated when accessing files using the Universal Naming Convention (UNC), which specifies a hostname, share name, and optional file path. Additionally, they note that thumbnails may be created in a variety of circumstances which do not involve the original source image being opened or viewed in thumbnail mode. This includes thumbnails being automatically generated based on the two most recently modified files in a directory, for use by the directory preview. Thumbnails are also generated as a result of dragging and copying files, even when such activities are cancelled by the user.

The existing literature has not described the changes introduced to the thumbcache in Windows 10, which we present here in Section 3.1. Additionally, prior work has not discussed the low level details of how Windows thumbnails are represented on disk, and how this differs across Windows versions, both in the binary and pixel domains, which is discussed in Section 4.2.

2.3 Centralised Thumbnail Caches in other Operating Systems

Centralised thumbnail caches are not used exclusively in Windows operating systems. Morris and Chivers (2010) discuss the centralised thumbnail caches used in versions 9.10 and 10.04 of the Kubuntu and Ubuntu operating systems, respectively. Thumbnails are stored in the user's home directory in `~/.thumbnails`[1], which is composed of three sub-directories: fail, large, and normal. All thumbnails are of the PNG format and correspond to 128×128 pixels for normal thumbnails, and 256×256 for large. Thumbnails which cannot be generated are tracked using the 'fail' sub-directory. In contrast to Windows, these thumbnail caches have no index; instead, thumbnails are stored directly in these sub-directories, with MD5 hashes of the source URI for filenames, allowing for fast thumbnail lookups. These thumbnails contain the modified times of the original files, such that updated originals signal that the thumbnail must be rebuilt. Importantly, as the thumbnail cache in Ubuntu is just a directory, it may be used by third party programs, thus there is a possibility that some thumbnails are configured differently to those generated by the operating system.

Thumbnail generation, storage, and extraction for the Android operating system is explored by Leom et al. (2015). The thumbnail cache for the built-in Gallery application is located at `/sdcard/Android/data/com.google.android.gallery3d.cache/imgcache.0`, and is composed of a single file which contains thumbnails of 200×200 and 640×480 pixels in the JPEG format. Larger thumbnails are generated for most pictures when they are created using the camera application, while smaller thumbnails are created when viewing images in the Gallery application.

Finally, Newcomer and Martin (2014) describe the per-user thumbnail caches found on Mac OS X. Thumbnails for the Spotlight and Finder applications are provided by the QuickLook technology and are generated for many file types.

[1] This appears to have moved to `~/.cache/thumbnails` in more recent releases of Ubuntu.

Different thumbnail sizes are used depending on the finder viewing mode. The directory containing the cache is located in `/private/var/folders/<2random>/<30random>/C/com.apple.QuickLook.thumbnailcache` for versions 10.7 and 10.8, with the 'C' in the path being switched for '–Caches–' on the older 10.5 and 10.6 OS. Each user has their own directory in `folders` consisting of two random lowercase characters, potentially including an underscore, followed by a subdirectory of 30 random characters. It is unclear if this is actually stochastic or deterministically generated by a hashing function. An SQLite database, index.sqlite is used to store information on the location of each source file, as well as the location of its thumbnail, which is indexed by referencing its offset in the corresponding thumbnail.data binary file containing thumbnail image data.

2.4 Thumbnail Caches in Forensics Investigations

Using thumbnail caches as part of an investigation is not a new idea. Quick et al. (2014) both explore the structure of the Windows thumbcache across older Windows versions, but also evaluate the thumbcache support for a variety of industry standard forensic tools, such as FTK, Encase and X-Ways, as well as some thumbnail specific tools. The authors note that while many tools support parsing the Windows thumbcache_xxx.db files, there was little support for linking this with the Windows.edb file to match thumbnails to full sized originals. The authors propose a method which unifies existing work on the thumbcache files, Windows.edb, and carving of edb.

Using hash databases directly for triage is discussed by Shaw and Browne (2013), where the thumbnail cache is used as part of a previewing system for quick device assessment. The authors also explicitly discuss the use of cryptographic hash databases for detecting previously encountered thumbnails bearing illegal content.

The research presented in this paper differs from prior work in that it acknowledges the weaknesses of the cryptographic hash approach for detecting illegal media using thumbnails, particularly due to changes in the thumbnailing process across Windows versions, which is explored in Section 4. A method of generating the exact binaries present in the Windows thumbnail cache is also proposed, which helps alleviate this issue by automating thumbnail creation in Section 3.3. An alternative solution which makes use of perceptual hashing is also explored in Section 5, which does not appear to have been considered in the literature.

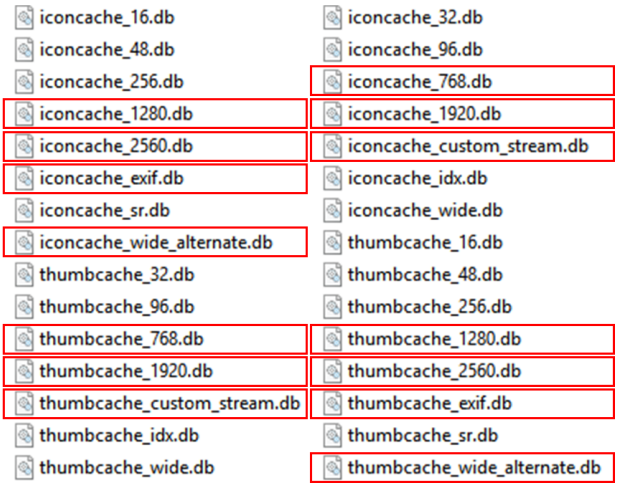


Figure 1: A screenshot of thumbcache files present on Windows 10. Files added in Windows 10 are indicated with boxes. Iconcache and thumbcaches of dimensions 1024×1024 were removed for Windows 10.

3. AUTOMATICALLY EXTRACTING THUMBNAILS

Prior work presented in Section 2.2 has focused on detailing the structure of the Windows thumbnail cache files, with analysis of the triggering mechanisms for populating the cache, and the forensic significance of cached thumbnails. In contrast, this paper seeks to exploit the centralised nature of Windows thumbnail caches, using it as a catalogue for images on the device. This allows contraband to be detected by processing a single directory, rather than an entire disk. Additionally, variations between thumbnails produced by different Windows versions are explored here for the first time.

3.1 Windows 10 Thumbcache

While the Windows 10 thumbnail cache remains in the same location as its predecessors, several changes have been made to the files used to store the thumbnails. A complete list of thumbcache files for a Windows 10 user is depicted in Figure 1. The 1024×1024 thumbnail database was removed, replaced by 768×768 and 1280×1280 pixel thumbnails, with very large 1920×1920 and 2560×2560 thumbnails also being added. Wide thumbnails also have an extra file named thumbcache_wide_alternate.db, while completely new items are included in the thumbcache_exif.db, thumbcache_custom_stream.db, and respective iconcache files. The thumbcache_exif.db database appears to contain thumbnails which are embedded in source JPEGs, converted to use the quantisation tables of the other thumbnail databases. Example thumbcache entries, as displayed by the Thumbcache Viewer application (Kutcher, 2016) are provided in Figure 2.

This paper focuses on the 96×96 and 256×256 thumbnail caches, as they are the most commonly generated, corresponding to small/medium/large and extra-large thumbnail previews. Thumbnails with dimensions of 96 are stored as bitmaps, while those with 256-pixel dimensions are stored as JPEGs. Version 1709 of Windows 10 (October 2017) enforced a limit of less than 500MiB on each of these cache files[2]; this limitation appears to have been lifted on later versions (tested on 1803, April 2018). In place of this file size limitation, the cache is regularly cleared by the SilentCleanup task in Windows (Brinkmann, 2019), which runs the built-in Disk Cleanup utility, resetting the cache and moving old database files to a ThumbCacheToDelete directory, which is discussed further in Section 7. Further work is required to test if old thumbnails which have not been accessed for extended periods of time are removed, but the current assumption is that they are not.

Even where both the 96-pixel and 256-pixel thumbcaches are at the size limits imposed by Windows 10 version 1709, this corresponds to less than 1GiB of data to process per user, and approximately 45,000 thumbnails. That is, for a single user with a 1TB drive, the worst case for this approach requires that only 0.1% of the disk is sequentially read. Assuming a sequential read speed of 100MiB/s[3], this would take approximately 10 seconds to acquire, as opposed to the three hours required to read the entire terabyte disk. This value will rise slightly if additional thumbnail sizes are processed, or the device has multiple user accounts.

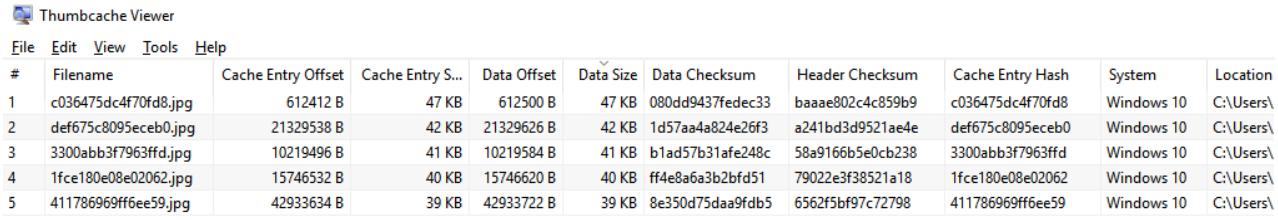
3.2 Test Dataset and Operating Systems

The primary focus of this work is Windows 10, however, thumbnails were also examined for the Windows Vista and Windows 7 operating systems to facilitate comparison between versions. The Flickr 1 Million dataset (Huiskes et al., 2010), composed of 1 million JPEG images, was used to explore thumbcache forensics at scale. However, no existing tool allowed for Windows thumbnails to be generated automatically, nor were there any existing datasets of Windows thumbnails available. In order to address this, a method was developed to create Windows thumbnails for each Flickr image on each platform, as described below in Section 3.3. While this approach allowed for the generation of 96- and 256-pixel thumbnails on Windows 7 and Windows 10, only 256-pixel thumbnails were automatically generated on Windows Vista.[4] To compensate for this, a small number of 96-pixel thumbnails were examined manually for Vista.

[2]Maximum values of 350MiB for 96-pixel thumbnails and 460MiB for 256-pixel thumbnails were observed, which were verified by users on the community forum at `https://answers.microsoft.com/en-us/windows/forum/windows_10-files/after-fall-update-windows-10-puts-a-maximum-size/6ad0a1e7-38c0-4547-9b8b-f7f3906c3a12`).

[3]For reference, a modern consumer HDD can sustain sequential read speeds of approximately 150-200MiB/s, while consumer SATA3 SSDs peak around 500MiB/s.

[4]This appears to be caused by the Windows Vista API not properly supporting the thumbnail dimensions argument.



#	Filename	Cache Entry Offset	Cache Entry S...	Data Offset	Data Size	Data Checksum	Header Checksum	Cache Entry Hash	System	Location
1	c036475dc4f70fd8.jpg	612412 B	47 KB	612500 B	47 KB	080dd9437fedec33	baaae802c4c859b9	c036475dc4f70fd8	Windows 10	C:\Users\
2	def675c8095eceb0.jpg	21329538 B	42 KB	21329626 B	42 KB	1d57aa4a824e26f3	a241bd3d9521ae4e	def675c8095eceb0	Windows 10	C:\Users\
3	3300abb3f7963ffd.jpg	10219496 B	41 KB	10219584 B	41 KB	b1ad57b31afe248c	58a9166b5e0cb238	3300abb3f7963ffd	Windows 10	C:\Users\
4	1fce180e08e02062.jpg	15746532 B	40 KB	15746620 B	40 KB	ff4e8a6a3b2bfd51	79022e3f38521a18	1fce180e08e02062	Windows 10	C:\Users\
5	411786969ff6ee59.jpg	42933634 B	39 KB	42933722 B	39 KB	8e350d75daa9fdb5	6562f5bf97c72798	411786969ff6ee59	Windows 10	C:\Users\

Figure 2: A screenshot of the Thumbcache Viewer application on a Windows 10 thumbcache. Data Checksum is a CRC64 of the entire thumbnail file, while Cache Entry Hash is the ID returned by the `GetThumbnail` method, and also serves internally as the file name. The truncated Location is the file path to the thumbcache file.

3.3 Thumbnail Dataset Generation for Large Datasets

In order to acquire data to analyse at scale, the Windows shell API was utilised to generate thumbnails for images by accessing the `IThumbnailCache` interface and calling the `GetThumbnail` method.[5] This can be used to force the generation of a new thumbnail or obtain one which has already been cached for an image. A parameter allows for the size of the thumbnail to be controlled, though only one size at a time may be requested. The API then returns a memory mapped bitmap, regardless of the format the thumbnail is stored in (BMP or JPEG). As a result, without knowing which function Windows uses to save the memory mapped image to a particular file type, it is non-trivial to recreate the exact binary data stored in the thumbcache from the object the API returns. That is, thumbnails saved to disk in this fashion will not be identical with those stored in the cache itself, and cannot be used directly for cryptographic hash based contraband detection.

To work around this issue, thumbnails can be obtained directly by parsing each thumbcache_xxx.db file after calling `GetThumbnail`. This is achieved by keeping track of the IDs returned by each call to `GetThumbnail` and extracting items in the cache with matching IDs. These IDs appear as *Cache Entry Hash* in Figure 2, and are derived from a hash function using the volume GUID, NTFS FILEID, file extension, and last modified time (Khatri, 2012). For efficiency, the thumbcache_xxx.db should not be parsed after each thumbnail is generated. Instead, several hundred, or thousand, thumbnails should be generated at a time, with all of them being recovered from the thumbcache_xxx.db in a single pass. However, a small enough batch size should be chosen such that thumbnails are not overwritten (on older Windows 10 releases) before they are recovered, with a batch size of 5000 proving effective in this work.

The end result of this processing is a dataset composed of the exact binaries stored by Windows in the thumbcache files. This dataset can then be further processed to generate contraband lookup databases, as described in Section 4 and Section 5.

4. MATCHING EXACT THUMBNAIL BINARIES

With a dataset of thumbnails in the format used by the Windows thumbcache, a lookup database may be created by calculating cryptographic hash digests, such as SHA256, for each thumbnail. When examining a Windows computer, each thumbcache_xxx.db file may then be parsed, and each entry in the cache hashed and checked against this database. This could either be done in memory while parsing thumbcache files or after extracting thumbnail images to a directory.

Section 4.1 discusses the possibility of augmenting this approach using CRC64 checksums which are already present in the thumbcache files, while Section 4.2 and Section 4.3 discuss pragmatic concerns for this approach.

[5] `https://msdn.microsoft.com/en-us/library/windows/desktop/bb774628.aspx`

4.1 Intermediate Lookups Using Embedded Checksums

The Windows thumbnail cache stores CRC64 checksums for each thumbnail, depicted as *Data Checksum* in Figure 2. This presents the opportunity to use these embedded checksums in a lookup database in the same manner as cryptographic hashes, except that they need only be parsed as strings from the thumbcache, rather than calculated from binary data at runtime.

The CRC64 checksum stored in the Windows thumbnail cache makes use of an unknown polynomial, however the lookup table used to generate values is stored in the thumbnail.dll file. Utilising this table, the first 1,024 bytes of the thumbnail are processed to calculate an initial CRC64, while the remaining bytes are used to calculate second CRC64 value. These values are then combined using the exclusive-OR (XOR) operator, producing the final checksum which is stored in the cache. This process was identified from the source code of the Thumbcache Viewer (Kutcher, 2016) application and verified manually by comparing the output and embedded checksums in the thumbcache.

However, these checksums are more likely to have hash collisions than cryptographic hashing algorithms (Dandass et al., 2008). While no collisions were observed for Windows 10's 256-pixel thumbnails, they did occur for the 96-pixel bitmaps. Three pairs and a triplet of non-identical 96-pixel thumbnails possessed the same CRC64 checksum while producing different SHA256 digests.

As the CRC64 algorithm produces collisions at the million image scale, CRC64 matches should be verified using a cryptographic algorithm, such as SHA256. This is faster than computing SHA256 hashes for all thumbnails in the thumbnail cache, but requires a hash and checksum pair to be stored for each thumbnail in the lookup database. In this work CRC64 checksums and SHA256 hashes were stored separately in two C++ unordered map structures.

4.2 Thumbnail Differences Between Windows Operating Systems

Thumbnails from the same version of Windows were shown to be consistent across two computers for both Windows 7 and Windows 10, which also held when using virtual machines. This was verified by extracting thumnbnails for all images in the Flickr 1 Million dataset for multiple computers and verifying that SHA256 image hashes were identical between machines. As such, hardware differences should not cause thumbnails to be generated differently. However, an examination of the thumbnails produced by Windows Vista, Windows 7, and Windows 10, showed that binary identity cannot be relied upon across different Windows versions, even when produced on the same computer. Differences can be explained in terms of the Windows API used to generate the thumbnails by the operating system, which can undergo change over time, such as when a new operating system is released. Based on prior Windows API version numbering, major changes to the API occur when a new operating system is released, and as such, thumbnail cache behaviour is likely to remain stable within a given version of Windows.

The following discussion examines thumbnail differences between Windows versions in some detail, with findings that could potentially have implications for everyday investigations.

4.2.1 96-pixel Thumbnail Differences

All 96-pixel thumbnails are stored in the BMP format in the cache, however, they are not generated in an identical fashion between Windows versions. Following the standard 64 byte BMP file header, Windows Vista and 7 use the 40-byte `BITMAPINFOHEADER` for the Bitmap Information Header, while Windows 10 uses the longer 124 byte `BITMAPV5HEADER`. Additionally, while Windows Vista and Windows 7 use no compression, Windows 10 makes use of bitfields compression, meaning that the raw binary data will not be directly comparable to prior Windows releases. Pixel data for thumbnails, after extraction from the BMP format, was shown to be frequently identical across Windows versions, however, this

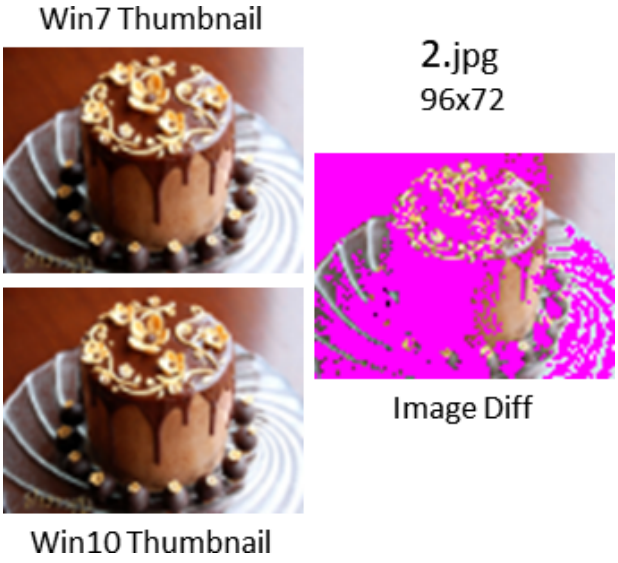


Figure 3: An image diff of the Windows 7 and 10 thumbnails (96-pixel) generated for 2.jpg in the Flickr 1 Million dataset. Diff produced using the resemble.js library. Pixel differences in pink highlight that each Windows version may create slightly different thumbnail outputs.

was not the case for the entire dataset. Figure 3 depicts an instance where two Windows versions produce different bitmap data for the same input image, with differences being highlighted by the resemble.js library (Cryer, 2017).

A further complication is the portion of the image used to produce the thumbnail in the first place. Windows 7 was observed to derive the 96-pixel thumbnails from embedded EXIF thumbnails when available, rather than from the full sized image. This can produce a thumbnail preview which is cropped differently to the full sized image, such that the thumbnail may contain elements which are no longer present in the full sized image. This can occur because the embedded EXIF thumbnail is not always updated when an image is cropped or otherwise edited (Kuksov, 2016). An example of this phenomenon is provided in Figure 4, where elements of the uncropped original are present in the EXIF and Windows 7 thumbnails but are no longer present in the full sized version. This behaviour was present for thumbnails generated by both the API calls and manual inspection of 96-pixel thumbnails on Windows 7, but was not reproduced in Windows Vista or Windows 10. Curiously, the 256-pixel thumbnails in Windows 7 are not generated from the embedded EXIF thumbnail, meaning that a user would see different image previews when switching between thumbnail sizes in Windows Explorer.

Figure 4: A comparison of 49530.jpg from the Flickr dataset with its embedded EXIF thumbnail, Windows 7 thumbnail, and Windows 10 thumbnail. Image dimensions provided. Windows 7 uses embedded EXIF thumbnails to generate the 96-pixel thumbcache entries, which are not necessarily updated when an image is cropped or otherwise modified.

4.2.2 256-Pixel Thumbnail Differences

256-pixel thumbnails are stored in the JPEG format, which allows for varied compression parameters. Images across all three operating systems were found to use the default Huffman tables provided in the JPEG specification, however there are differences in the quantisation tables used. Windows Vista and Windows 7 share the same quantisation tables, while Windows 10 uses a table with finer quantisation on the higher DCT frequencies, resulting in higher image quality. This means that binary identity is lost as the thumbnails effectively have different quality settings. However, despite using the same quantisation tables, Windows Vista and Windows 7 also produce different binaries, with no images in the Flickr 1 Million dataset producing the same SHA256 digest across versions due to pixel differences. The reason for this difference can-

not be attributed to the header and compression settings, which are identical on Windows 7 and Vista. The difference, then, must be the thumbnail API, introduced either in the rescaling of the image, or in the encoding of the JPEG data stream.

A comparative overview of both thumbnail types is provided for the three tested Windows versions in Figure 5.

4.3 Dealing With Differences: Version Specific Databases

As the thumbnails produced by different versions of Windows frequently contain different binary data, it is not possible to create a universal lookup database from thumbnails generated on a single Windows release. For traditional cryptographic hash based lookups, one or more databases need to be created by generating thumbnails for each Windows variant. This would mean either a single unified lookup database or a set of OS-specific thumbnail databases, which may also contain CRC64 checksums. Additionally, thumbnails of each dimension must be hashed separately, increasing the total number of databases and fingerprints.

While this may add to overall maintenance overheads, its use for triage means that less commonly encountered operating systems need not be accommodated. Instead, the database could be maintained for the most popular, or most recent, Windows releases, which would make up the bulk of a typical investigator's workload. Figures for 2019 show that Windows 10 makes up close to 60% of Windows installations and is still increasing, with Windows 7 being the second most popular at approximately 30%[6]. Assuming the distribution of Windows versions in typical forensic investigations is roughly the same, this means that maintaining a database for the top two releases would accommodate 9 out of 10 investigations on Windows machines seized in 2019. Of course, investigative backlogs mean that computers being analysed today may have been seized several years ago. Looking back to 2016 (three years at the time of writing), Windows 7 and Windows 10 made up approximately 40% each of all Windows computers, meaning that 8 out of 10 Windows PCs were running the top two releases[7].

[6] `http://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide`

5. ROBUST THUMBNAIL MATCHING

Cryptographic hashes are simple to calculate and have constant time database lookups. However, their rigidity in only matching exact binary content proves troublesome for the thumbnail variations found across Windows operating system versions. One method for de-coupling thumbnail lookup databases from specific Windows versions is to focus on detecting thumbnails which are visually identical, rather than checking for identity in the binary domain. This visual robustness can be achieved by using perceptual hashing techniques (Hadmi et al., 2012), which aim to generate robust signatures from visual features, providing tolerance to content-preserving changes in the binary data. This would allow a full sized image to be compared directly to its corresponding thumbnail, regardless of thumbnail dimensions, compression differences, or source Windows version.

Perceptual hashing approaches typically perform well even when images have been rescaled. This means that images in the thumbcache_xxx.db files may be compared to a database of perceptual hashes generated from the original, full sized, contraband images. No intermediate thumbnails, or databases, need to be generated, which frees this approach from the operating system API. Indeed, assuming a robust perceptual hashing technique, this method should be completely operating system independent, performing equally well on Windows and non-Windows platforms.

5.1 Choices of Perceptual Hash

While there are many approaches to perceptual hashing, with their own properties and weak-

[7] Taking into account that all Windows operating systems had a combined market share of approximately 85% in 2016, this means that 7 out of 10 of ALL computers seized in 2016 would have been either Windows 7 or Windows 10.

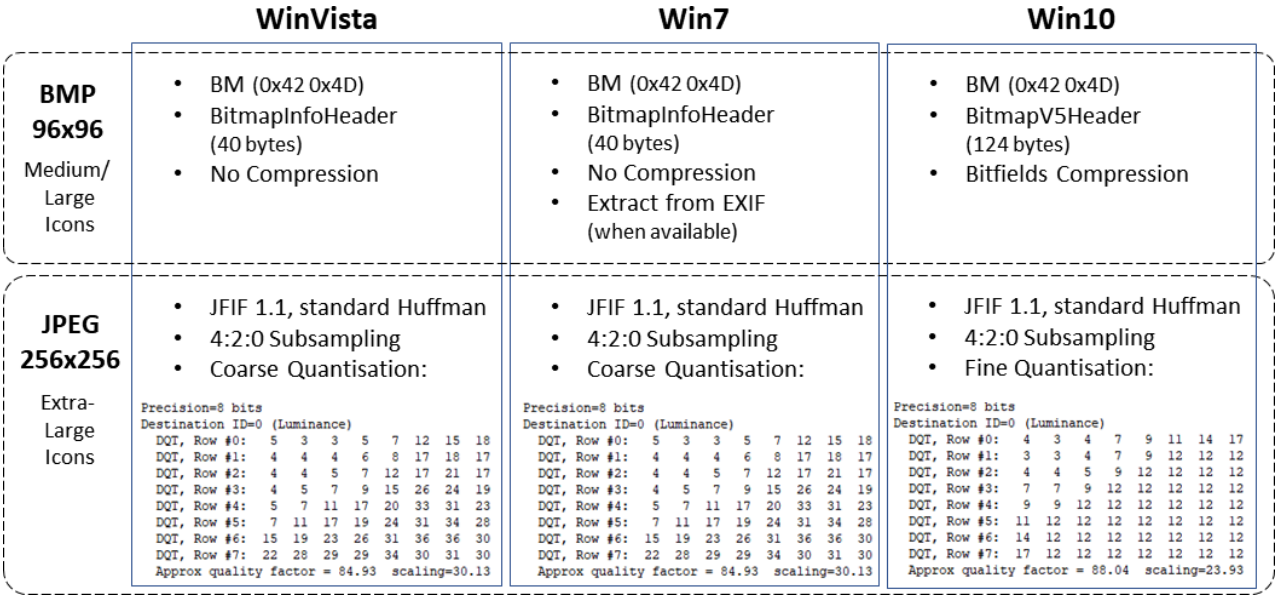


Figure 5: A comparison of thumbnail images for Windows Vista, Windows 7, and Windows 10. Chrominance quantisation tables are omitted for brevity.

nesses (Breitinger et al., 2013), many of the published approaches do not provide implementations. As such, a pragmatic approach was taken where two popular perceptual hashing algorithms with open source implementations were chosen for this work. A brief description of each perceptual hashing method is provided below.

Phash: The Phash library (Klinger & Starkweather, 2012) is an open source perceptual hashing library for performing image comparisons. However, the original codebase has not been updated in some time. As a result, a similar, more recent, implementation was chosen in the Python ImageHash library (Buchner, 2017). This library contains several perceptual hashing approaches, including ahash (average colour hashing), dhash (gradient tracking), whash (discrete wavelet transform), and a modification of the original Phash (discrete cosine transform). Based on initial testing, the modified Phash algorithm was chosen as it had the best performance on the thumbnail datasets. Frequency transformations, such as the DCT used in Phash, are able to capture essential properties of an image and have proven to be effective in the literature (Hadmi et al., 2012).

Blockhash: The Blockhash algorithm (Yang et al., 2006) breaks an image into blocks and compares the mean colour values between blocks to create a signature. As the original paper does not provide an implementation, a third party derivative implementation was used (Commonsmachinery, 2018).

Default hash sizes for both algorithms were used (64-bit for Phash, 256-bit for Blockhash). The distance between two hashes was calculated using the Hamming distance, which is simply the sum of bit differences between signatures. The normalised Hamming distance was then calculated by dividing this sum by the length of the hash digest. This produces a distance between 0 and 1, where 0 indicates an identical perceptual hash, and 1 indicates that all bits are different. Reported distances below refer to this normalised Hamming distance.

5.2 Determining Distance Thresholds For Matching Images

Ideally, visually identical images should produce the same perceptual hash digest, and hashes for almost identical images should only differ by a small number of bits. As the thumbnails for a single source image can differ across Windows

versions, having a matching algorithm which tolerates small variations is necessary. One way to achieve this is to set distance thresholds for what constitutes an image match, while making sure that this threshold is small enough to avoid visually dissimilar images from being considered a match. For example, setting the threshold to $t = 0.3$ would mean that images with a perceptual hash bit difference of less than 3-in-10 would be considered a match, while anything above this is not a match.

In order to determine an appropriate distance threshold for image matches it was necessary to explore typical distances between unrelated images. Thresholds were evaluated for the full sized images in the Flickr 1 Million dataset by calculating pairwise distances from each image to 50 random images in the dataset, with no repeated pairings. Duplicate binary files as determined by the SHA256 algorithm were not included, resulting in a sample of slightly less than 50 million pairwise comparisons, of the potential 500 billion comparisons.

For both perceptual hashing algorithms, the mean and median normalised hamming distances were almost exactly 0.5, and appear to be normally distributed (see Figures 6 and 7). As images in the Flickr dataset should be unrelated[8], these comparisons should hold for any heterogeneous dataset. This result likely reflects design choices behind the algorithms, where two unrelated images should produce hashes which are around 50% different on average.

Based on this data, it is possible to derive false positive rates for various distance thresholds for unrelated Flickr 1 Million images. In this context, a false positive occurs when two non-identical images in the dataset register as matches for a given distance threshold. If this occurs often, then the distance threshold is too high, however, setting the threshold too low may exclude some legitimate matches, generating false negatives.

False positive rates in this dataset are provided for various thresholds in Table 1. As there are

[8]Though there is at least one case where an image has identical pixel data but a different SHA256 hash digest.

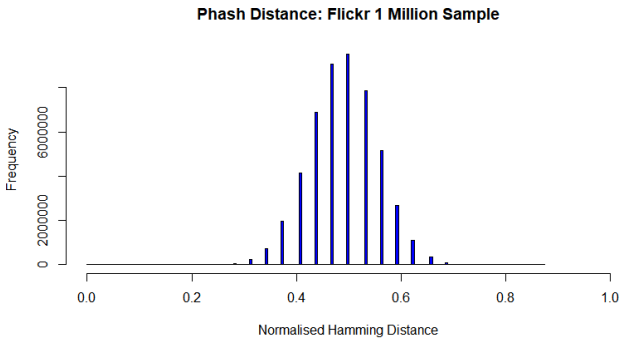


Figure 6: Phash normalised Hamming distance distribution for the 50 million sample comparisons of the original Flickr 1 Million dataset. Distances values appear to have fewer discrete values than those for Blockhash, resulting in small gaps between bars.

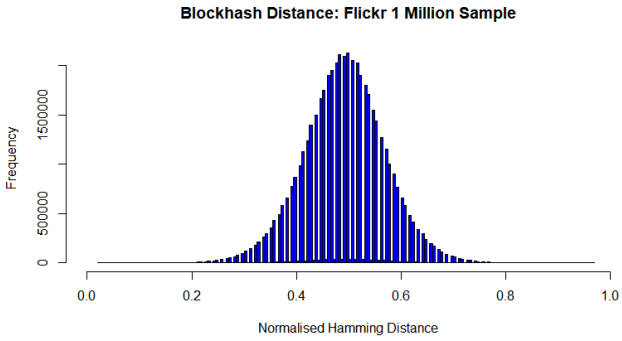


Figure 7: Blockhash normalised Hamming distance distribution for the 50 million sample comparisons of the original Flickr 1 Million dataset.

only expected to be tens of thousands of potential thumbnails on a device, given the SilentCleanup task (Brinkmann, 2019), a relatively high false positive rate may be acceptable. A false positive rate of 0.01% would generate approximately five false positives for every 50,000 thumbnails, which places little burden on a human examiner during manual verification. Despite having a nearly identical mean and standard deviation, the distance thresholds for each perceptual hash are quite different. This can be explained by the lower utilisation of the normalised Hamming space by Phash, partially due to it using a lower number of bits per hash (64-bit vs 256-bit). However, this is a result of using

the default hash size for Phash, and, as such, is not corrected for statistically.

Using a false positive rate of 0.01%, the effective false negative rates for detecting image thumbnails were calculated. A false negative occurs when a full sized image and its thumbnail is not considered to be a match, and is caused by the distance threshold being set too low. In the context of an investigation, this would mean that an item of contraband was not automatically detected, even though it is present in the thumbnail cache. False negative rates for Windows 10 are provided in Table 2, with values for Windows Vista and Windows 7 being almost identical. Both algorithms were found to miss 1-in-2000–4000 thumbnails. This can be attributed to both weaknesses in the algorithms and characteristics of the thumbnailing process. Phash was found to be particularly poor when detecting thumbnails with fractals or repeated patterns (Figure 8), while Blockhash was poor when images possessed large areas with little to no variation in colour (Figure 9).

The weakness of each algorithm may be mitigated by using both algorithms simultaneously, calculating the distance for each perceptual hash and only utilising the lower of the distances. This effectively decreases the false negative rate to approximately 0.002%, or 1-in-50,000. As this method is designed for rapid triage, and already makes assumptions about images being in the thumbnail cache, this level of performance should be acceptable in most cases. However this may be unacceptable in scenarios where there may be only a single image present in the cache and no manual verification is performed.

Unfortunately, the false negative rate cannot be reduced by simply increasing the distance threshold, as some thumbnails were observed to have a hamming distance greater than 0.5 to the full sized version, which is larger than the mean distance to a completely unrelated image. It is conceivable that there exists a perceptual hashing algorithm which performs better in this use case and would provide a higher degree of confidence that no thumbnail has been overlooked. An ideal algorithm should primarily address the problem of scale invariance, such that the same fingerprint is generated from an image regardless of resizing. Ideally, all thumbnails would fall within some well defined distance of their full sized counterparts, such that the false negative rate is effectively zero. However, a sufficiently low false negative rate may be tolerable, as long as it is unlikely to affect any investigative decisions (James & Gladyshev, 2013). One further consideration is the performance of evaluating hash lookups in Hamming space, which is not as fast as the constant time lookups of traditional hashing mechanisms. However, this problem has solutions in the literature, such as multi-hash indexes (Norouzi et al., 2012).

6. BENCHMARKS

To be effective, thumbnail based triage must be very fast. As such, timed benchmarks were executed to assess the lookup performance of the cryptographic hash based approach. Perceptual hashes were omitted as the two algorithms tested were considered to be insufficiently accurate.

The first 10 images (0.jpg to 9.jpg) from the Flickr 1 Million collection were chosen to serve as known lookup items and were converted to 96 and 256-pixel thumbnails. These thumbnails were then processed to produce SHA256 and CRC64 fingerprints for automatic detection. The database was loaded into memory and populated with five million randomly generated CRC64 and SHA256 values to provide appropriate scale.

Two thumbnail cache files were then populated using the `GetThumbnail` method (as discussed in Section 3.3), with the first 10,000 images in the Flickr 1 Million dataset being cached for thumbcache_96.db and the first 25,000 for thumbcache_256.db. These numbers were chosen as they are near the maximum observed capacities of these files on older versions of Windows 10, and as such would avoid thumbnails being overwritten or deleted. This resulted in cache files of 257MiB and 362MiB, respectively, which were then copied to avoid further manipulation.

Two machines were selected to perform the comparison. A workstation (Core-i5 4690k, 16GiB DDR3, 525GB Crucial MX300 SSD, 4TB Western Digital Red HDD) and netbook (Atom

Figure 8: Sample images where Phash performs poorly when comparing original image to 256-pixel thumbnails. Distance is normalised Hamming distance.

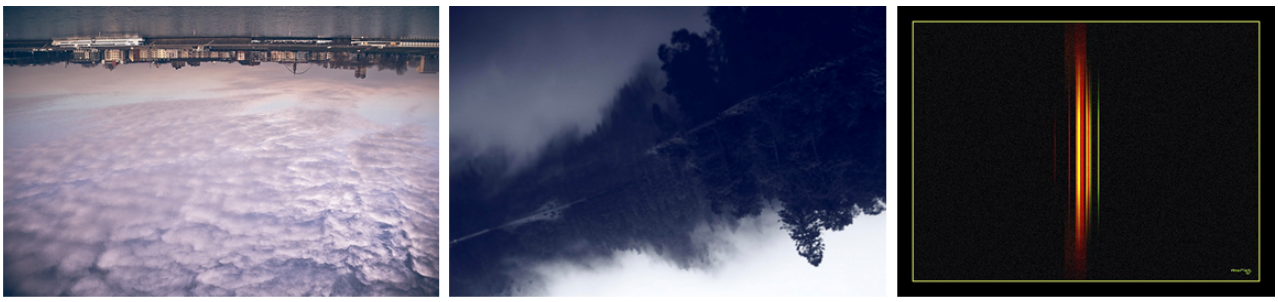


Figure 9: Sample images where Blockhash performs poorly when comparing original image to 256-pixel thumbnails. Distance is normalised Hamming distance.

	96px Thumbnails		**256px Thumbnails**	
	CRC+	**SHA**	**CRC+**	**SHA**
Workstation HDD	2.22s	2.20s	3.03s	3.02s
Workstation SSD	0.60s	0.81s	0.85s	1.28s
Netbook HDD	4.80s	6.66s	5.07s	11.16s

Table 3: Benchmarks for the parsing and lookup times for thumbcache_96.db and thumbcache_256.db, containing 10k and 25k images, respectively. CRC+ verifies initial CRC64 hits with SHA256 lookups. Reported values are the median across 30 iterations of the experiment.

False Positive Rate	0.00001%	0.0001%	0.001%	0.01%
Phash Distance	0.1250	0.1875	0.2188	0.2500
Blockhash Distance	0.0469	0.0781	0.1172	0.1641

Table 1: False positive match rates and their corresponding Phash and Blockhash distance thresholds for pair-wise comparisons in the full sized Flickr 1 Million dataset.

False Negative Rate (Distance: 0.2500 Phash, 0.1641 Blockhash)		
	Phash	**Blockhash**
Win10 96-pixel	0.0494	0.0467
Win10 256-pixel	0.0227	0.0404

Table 2: False negative rates for each perceptual hash algorithm when comparing full sized Flickr 1 Million images to their Windows 10 thumbnails. Distance thresholds are set for a 0.01% false positive rate.

N450, 1GiB DDR2, 160GB Western Digital HDD (OS)) were used, which allowed for the exploration of the performance using a relatively high specification machine and a slower, legacy system. These machines were then used to parse each thumbcache_*xxx*.db using a single threaded approach, with each item being looked up in the corresponding hashset.

Benchmarks were performed on the workstation using both the HDD and SSD, while the netbook used only its internal hard drive. Two lookup modes were tested for each thumbnail size: *i)* initial CRC64 lookups, with positive hits being verified with SHA256 (CRC+), and *ii)* SHA256 only (SHA), as described above in Section 3. The overall execution time does not include the time taken to read lookup databases into memory, as it is assumed that they would be contained in the forensic application's executable in practice. Each run was repeated 30 times, with the memory cache being cleared each time using the `EmptyStandbyList` utility (Liu, 2016).

The median benchmark times[9], shown in Table 3, indicate that this approach is very fast, regardless of the storage media used, taking approximately three seconds in the worst case on the workstation, and 11 seconds on the netbook. Initial CRC lookups offered no benefit when using the Workstation's hard drive, but reduced times by approximately 25% on the SSD, and 30–50% on the netbook. As parsing the SHA-only approach is sufficiently fast, it is likely not worth the extra memory overhead, or database upkeep, to perform CRC lookups.

While there is no guarantee that contraband on a device will have a thumbnail in one of the corresponding thumbcache_*xxx*.db files, this technique is fast enough to be used as an initial check in a forensics investigation. Additionally, this extraction and processing time does not change with the size, or number, of disks present in a device, as the time taken is related only to the number of entries present in the individual thumbcache_*xxx*.db files, and number of users on each device.

All thumbnail cache extraction and parsing code used in this work are available online at `https://github.com/smck1/shellthumbs`.

[9]Median rather than mean, to account for small variance due to background processes in Windows, particularly on the netbook.

7. DISCUSSION

The thumbnail cache on Windows operating systems can be used as a centralised catalogue of images on a device. However, it should be noted that there is no guarantee that an image which resides on a computer will be present in the thumbnail cache. The literature discusses various triggering mechanisms for caching, in some cases showing that images need not even be viewed to be present in the cache. Despite this, the Windows thumbnail cache is a user specific image preview store which does not appear to pro-actively cache all images on the device. However, it does represent the most recently viewed items in Windows Explorer.

Morris and Chivers (2011a) previously investigated deletion behaviour of thumbnails on Windows 7, finding that when the original image is deleted, the cache entry is not necessarily purged. When the thumbnail cache is cleared using the Disk Cleanup utility (Cleanmgr.exe) built-in to Windows, all thumbcache_xxx.db files are temporarily moved to a subdirectory named 'ThumbCacheToDelete', with fresh databases taking their place (Morris & Chivers, 2011a). This behaviour was also present in our experiments on Windows 10, including purges initiated by SilentCleanup (Brinkmann, 2019), and indicates that entire cache databases may be found in unallocated disk space, even after an attempt to purge them. As such, during triage it would be wise to check the thumbnail cache directory for deleted versions of ThumbCacheToDelete with a forensic preview tool or to parse the NTFS Master File Table (MFT) for deleted cache databases.

When using traditional cryptographic hashing it is necessary to store multiple hashes for each source image, for both different thumbnail dimensions and operating system versions. This will likely mean that the hash database is too large to store in main memory, however, Bloom filters have been shown to be an effective mitigation tactic in the literature (Penrose et al., 2015). Bloom filters can also be used for perceptual hash databases, which may include hashes for more than one perceptual hashing algorithm to provide robustness.

While this paper discusses centralised thumbnail caches for the Windows operating system, the literature indicates counterpart centralised caches on Ubuntu, Android, and MacOS, which could be similarly exploited. When perceptual hashes are employed, fingerprints should be representative of the thumbnail regardless of operating system, and can be compared directly to the thumbnails present in the cache. However, further work is required to explore the false positive and negative rates for thumbnails on non-Windows devices, as the thumbnail generation process may impact detection performance. It is also recommended to create perceptual hashes for embedded EXIF thumbnails, as these may be used to create thumbnails, as with the 96-pixel thumbnails on Windows 7. When utilising traditional cryptographic hashing, the corresponding thumbnail generation APIs for each operating system must be utilised to generate the thumbnails for hashing. Further work would also need to verify if changes occur between operating system versions, though there is the possibility of applying the same hashes to a large number of Linux distributions. However, as the traditional hashing approach entails a lot of maintenance and is heavily reliant on operating system APIs, the perceptual hashing method is recommended for operating system interoperability.

The process for conducting thumbcache based triage on Windows is as follows: After mounting the device read-only, User directories on the device are enumerated, and the thumbcache_xxx.db files for each user are parsed in turn. Each thumbnail binary in the database file is then hashed and compared to the contraband hash set. After parsing allocated thumbcache files, the existence of allocated or deleted ThumbCacheToDelete directories should be explored in order to recover previous versions of the cache. If records in the database files match hashes for known contraband, the investigator can then attempt to locate the original file by cross referencing the thumbnail in the Windows search index stored in Windows.edb (Morris & Chivers, 2011b), though there may not be an entry. This process is summarised in Algorithm 1.

Algorithm 1: Thumbnail Triage - Pseudocode to count number of thumbnail database hits on a Windows machine and check for the full file path

```
Input: Evidence Drive
Output: Hit Count
hit_count = 0;
media = openReadOnly(disk);
for user in media/Users/ do
    thumbcache_dir =
     media/Users/user/Appdata/Local
    /Microsoft/Windows/Explorer/;
    for thumbcache in thumbcache_dir do
        // thumcache_dir should also
         include ThumbCacheToDelete
        for record in thumbcache do
            thumbhash =
             hash(record.image);
            if thumbhash in contraband_db
             then
                hit_count++;
                thumbID = record.ID;
                path = lookupID(thumbID,
                 Windows.edb);
                if path then
                    print(path);
                end
            end
        end
    end
end
return hit_count
```

8. CONCLUSIONS AND FUTURE WORK

This paper has shown that centralised thumbnail caches offer an opportunity to perform rapid forensic triage and contraband detection, potentially saving a huge number of investigator hours during the on-site portion of an investigation. This can help focus the investigation and quickly identify devices of interest. Cryptographic hash analysis of the Windows 10 thumbcache can be performed in a matter of seconds, even on low end legacy equipment. The cache itself serves as a sample of recent images on the device, reducing processing of entire disks to a few hundred megabytes. The automatic thumbnail generation approach using Windows APIs allows for thumbnails to be generated and hashed for multiple Windows versions, facilitating detection across a wider range of Windows versions without having come across each thumbnail directly in an investigation. Similarly, the use of perceptual hashing allows for detection to be generalised across all Windows version without needing to generate intermediate thumbnail images.

While this paper focuses on Windows 10, further research would allow the technique to be expanded to other operating system and mobile devices. This can be achieved by further refining the application of perceptual hashing techniques to detect target thumbnails of any dimensions and compression ratio. Flexible thumbnail fingerprint matching will allow for a single fingerprint database to be maintained, which may be used to detect contraband swiftly across any device using centralised thumbnail stores.

Future work includes empirically evaluating thumbnail evidence in historical cases in order to determine a base rate of detection, and likelihood of key evidence being found in the cache. It would also be useful to conduct an empirical analysis of the operating system distribution of seized machines relative to the market share of all operating systems at the time of seizure. This would enable law enforcement to make informed decisions about which databases to maintain. Additional work is also required to explore the fine grained behaviour of the thumbnail cache, such as cache sizes which trigger a cache reset, and whether or not old items are ever removed from the cache. Thumbnail analysis is fast and may be a key mechanism in reducing the large forensics backlogs suffered by modern law enforcement agencies across the world.

9. ACKNOWLEDGEMENTS

This research was supported by a scholarship provided by Peter KK Lee.

REFERENCES

Beebe, N. (2009). Digital forensic research: The good, the bad and the unaddressed. In *IFIP International Conference on Digital Forensics* (pp. 17–36). Springer.

Breitinger, F., Liu, H., Winter, C., Baier, H., Rybalchenko, A., & Steinebach, M. (2013). Towards a process model for hash functions in digital forensics. In *International Conference on Digital Forensics and Cyber Crime* (pp. 170–186). Springer.

Brinkmann, M. (2019, March). *How to block the automatic cleaning of Windows 10's Thumbnail Cache - gHacks Tech News.* Retrieved 2019-07-09, from `https://www.ghacks.net/2019/03/04/how-to-block-the-automatic-cleaning-of-windows-10s-thumbnail-cache/`

Buchner, J. (2017). *ImageHash.* Retrieved 2018-08-24, from `https://pypi.org/project/ImageHash/`

Commonsmachinery. (2018, July). *Contribute to blockhash development by creating an account on Github.* Commons Machinery. Retrieved 2018-08-24, from `https://github.com/commonsmachinery/blockhash` (original-date: 2014-09-02T17:46:34Z)

Cryer, J. (2017, August). *Resemble.js: Image analysis and comparison.* Huddle. Retrieved 2017-08-25, from `https://github.com/Huddle/Resemble.js` (original-date: 2013-02-21T14:25:27Z)

Dandass, Y. S., Necaise, N. J., & Thomas, S. R. (2008, April). An Empirical Analysis of Disk Sector Hashes for Data Carving. *J. Digit. Forensic Pract.*, *2*(2), 95–104. doi: 10.1080/15567280802050436

Franqueira, V. N., Bryce, J., Al Mutawa, N., & Marrington, A. (2018, December). Investigation of Indecent Images of Children cases: Challenges and suggestions collected from the trenches. *Digital Investigation*. doi: 10.1016/j.diin.2017.11.002

Garfinkel, S. L. (2010, August). Digital forensics research: The next 10 years. *Digital Investigation*, *7*, S64–S73. (DFRWS USA 2010. The Proceedings of the Tenth Digital Forensics Research Workshop. Philadelphia. Aug 2-4, 2010.). doi: 10.1016/j.diin.2010.05.009

Garfinkel, S. L. (2013, February). Digital media triage with bulk data analysis and bulk_extractor. *Computers & Security*, *32*, 56–72. doi: 10.1016/j.cose.2012.09.011

Grier, J., & Richard, G. G. (2015, August). Rapid forensic imaging of large disks with sifting collectors. *Digital Investigation*, *14*, S34–S44. doi: 10.1016/j.diin.2015.05.006

Hadmi, A., Ouahman, A. A., Said, B. A. E., & Puech, W. (2012). *Perceptual image hashing.* INTECH Open Access Publisher. Retrieved 2016-08-23, from `http://cdn.intechopen.com/pdfs/36921/InTech-Perceptual_image_hashing.pdf`

Huiskes, M. J., Thomee, B., & Lew, M. S. (2010). New trends and ideas in visual concept detection: the MIR flickr retrieval evaluation initiative. In *Proceedings of the international conference on Multimedia information retrieval* (pp. 527–536). ACM.

James, J. I., & Gladyshev, P. (2013, September). A survey of digital forensic investigator decision processes and measurement of decisions based on enhanced preview. *Digital Investigation*, *10*(2), 148–157. doi: 10.1016/j.diin.2013.04.005

Khatri, Y. (2012). *Windows 7 Thumbcache hash algorithm.* Retrieved 2017-11-09, from `http://www.swiftforensics.com/2012/06/windows-7-thumbcache-hash-algorithm.html`

Klinger, E., & Starkweather, D. (2012, October). *pHash the open source perceptual hash library.* Retrieved 2017-08-24, from `http://www.phash.org/apps/`

Kuksov, I. (2016). *What EXIF can tell about the photos you post online.* Retrieved 2018-03-22, from `https://www.kaspersky.co.uk/blog/exif-privacy/7893/`

Kutcher, E. (2016, October). *Thumbcache Viewer - Extract thumbnail images from the thumbcache_*.db and iconcache_*.db database files.* Retrieved 2017-

08-24, from https://thumbcacheviewer.github.io/

Leom, M. D., DOrazio, C. J., Deegan, G., & Choo, K.-K. R. (2015, August). Forensic Collection and Analysis of Thumbnails in Android. In (pp. 1059–1066). IEEE. doi: 10.1109/Trustcom.2015.483

Lillis, D., Becker, B., O'Sullivan, T., & Scanlon, M. (2016, April). Current Challenges and Future Research Areas for Digital Forensic Investigation. *arXiv:1604.03850 [cs]*. (arXiv: 1604.03850)

Liu, W. J. (2016, January). *Empty Standby List.* Retrieved 2017-08-31, from https://wj32.org/wp/software/empty-standby-list/

Morris, S., & Chivers, H. (2010). A comparative study of the structure and behaviour of the operating system thumbnail caches used in Kubuntu and Ubuntu (9.10 and 10.04). *Proc. of the 4th Cybercrime Forensics Education & Training. Canterbury Christ Church University, Canterbury, UK*.

Morris, S., & Chivers, H. (2011a). An analysis of the structure and behaviour of the Windows 7 operating system thumbnail cache. In *Proceedings from 1st Cyberforensics Conference.*

Morris, S., & Chivers, H. (2011b). Forming a Relationship between Artefacts identified in thumbnail caches and the remaining data on a storage device. *Cybercrime Forensics Education and Training*.

Newcomer, S., & Martin, L. (2014). Determining User Actions In Os X Based On Quicklook Thumbnail Cache Database Entries. *Issues in Information Systems*, *15*(2).

Norouzi, M., Punjani, A., & Fleet, D. J. (2012). Fast search in hamming space with multi-index hashing. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on* (pp. 3108–3115). IEEE.

Parsonage, H. (2012). *Under My Thumbs Revisiting Window s thumbnail databases and some new revelations about the forensic implications.* Retrieved 2017-06-14, from http://computerforensics.parsonage.co.uk/downloads/UnderMyThumbs.pdf

Penrose, P., Buchanan, W. J., & Macfarlane, R. (2015, March). Fast contraband detection in large capacity disk drives. *Digital Investigation*, *12, Supplement 1*, S22–S29. (DFRWS EU 2015. The Proceedings of the Tenth Digital Forensics Research Workshop. Dublin. March 23-26, 2015.). doi: 10.1016/j.diin.2015.01.007

Quick, D., & Choo, K.-K. R. (2014, December). Impacts of increasing volume of digital forensic data: A survey and future research challenges. *Digital Investigation*, *11*(4), 273–294. doi: 10.1016/j.diin.2014.09.002

Quick, D., Tassone, C., & Choo, K.-K. R. (2014). Forensic Analysis of Windows Thumbcache files. *Quick D, Tassone C and Choo KK R.*

Roussev, V., & Quates, C. (2012, August). Content triage with similarity digests: The M57 case study. *Digital Investigation*, *9*, S60–S68. (DFRWS USA 2012. The Proceedings of the Tenth Digital Forensics Research Workshop. Washington DC. August 6-8, 2012.). doi: 10.1016/j.diin.2012.05.012

Roussev, V., Quates, C., & Martell, R. (2013, September). Real-time digital forensics and triage. *Digital Investigation*, *10*(2), 158–167. doi: 10.1016/j.diin.2013.02.001

Shaw, A., & Browne, A. (2013, September). A practical and robust approach to coping with large volumes of data submitted for digital forensic examination. *Digital Investigation*, *10*(2), 116–128. doi: 10.1016/j.diin.2013.04.003

Yang, B., Gu, F., & Niu, X. (2006). Block mean value based image perceptual hashing. In *Intelligent Information Hiding and Multimedia Signal Processing, 2006. IIH-MSP'06. International Conference on* (pp. 167–172). IEEE.

Young, J., Foster, K., Garfinkel, S., & Fairbanks, K. (2012). Distinct sector hashes for target file detection. *Computer*, *45*(12), 28–35.