PeekaTorrent: Leveraging P2P Hash Values for Digital Forensics

Sebastian Neuner[a], Martin Schmiedecker[a], Edgar R. Weippl[a]

[a] *SBA Research, Vienna, Austria*

Abstract

Sub-file hashing and hash-based carving are increasingly popular methods in digital forensics to detect files on hard drives that are incomplete or have been partially overwritten or modified respectively. While these techniques have been shown to be usable in practice and can be implemented efficiently, they face the problem that a-priori specific "target files" need to be available. While it is always feasible and, in fact, trivial to create case-specific sub-file hash collections, we propose the creation of case-independent sub-file hash databases. To facilitate hash databases which can be publicly shared among investigators, we propose the usage of data from peer-to-peer file sharing networks such as BitTorrent. Most of the file sharing networks in use today rely on large quantities of hash values for integrity checking and chunk identification, and can be leveraged for digital forensics.

In this paper we show how these hash values can be of use to identify possibly vast amounts of data and thus present a feasible solution to cope with the ever-increasing case sizes in digital forensics. While the methodology is independent of the used file sharing protocol, we harvested information from the BitTorrent network. In total we collected and analyzed more than 3.2 billion hash values from 2.3 million torrent files, and we discuss to what extent they can be used to identify otherwise unknown file fragments and data remnants. Using open-source tools like *bulk_extractor* and *hashdb*, these hash values can be directly used to enhance the effectiveness of sub-file hashing at large scale.

Keywords: sub-file hashing, hash-based carving, file whitelisting, p2p file sharing

1. Introduction

One of the current problems in digital forensics is the vast amount of data to be analyzed, as hard drives with 8 terabytes capacity are readily available and the number of devices per person increases steadily. Both are factors for which the current forensic process model does not scale well [1]. Acquisition of large data drives can take days, and even though optimization techniques were recently introduced in the literature, e.g., sifting collectors [2] or file-based deduplication [3], they are not yet used in practice on a larger scale. Slack space, the general availability of counter-forensic tools and the increasing importance of RAM content for analysis further challenge the current boundaries of digital forensics. While file whitelisting is a common approach to reduce the number of files to be investigated, it is limited in numerous ways: for one, there is currently just one large corpus of hash values which is publicly shared – the NIST National Software Reference Library, containing 43 million file hashes. Secondly, these file hash values rely on hashing an entire file, and are thus unusable to identify files that are partially modified, or files which have been deleted respectively partially overwritten.

To cope with these problems, we present *peekaTorrent*, a methodology to identify files and file fragments based on data from publicly available file-sharing networks. It is based on the open-source forensic tools *bulk_extractor* and *hashdb* and can be readily integrated in forensic processes. It improves the current state-of-the-art on sub-file hashing [4] twofold: for one the hashed sub-file parts are larger than pure sector-based hashes, and thus less prone to false-positives for files that share common data segments. Secondly, we solve the problem that an a-priori sub-file hash database is required by creating one that can be shared openly. Lastly, no participation in file-sharing activity is needed as the torrent metadata or "metainfo", which is stored in the torrent file, already contains all the necessary information including the sub-file hash values. This information can then be used for file and fragment identification and effective file whitelisting, as well as for other use cases. As such, the contributions of this paper are as follows:

- We present a scalable methodology to identify files and file fragments based on sub-file hashing and P2P file sharing information.
- We collect and analyze more than 2.3 million torrent files, rendering up to 2.6 petabyte of data identifiable using that information.
- We identify several use cases for file (fragment) iden-

Email addresses: `sneuner@sba-research.org` (Sebastian Neuner), `mschmiedecker@sba-research.org` (Martin Schmiedecker), `eweippl@sba-research.org` (Edgar R. Weippl)

tification in the context of both file-whitelisting and blacklisting with that data.

- All obtained data and created source code is available online at `https://www.peekatorrent.org`.

The remainder of this paper is structured as follows: Section 2 provides the necessary background for this paper. Section 3 describes our idea of using sub-file hash values from peer-to-peer file sharing networks in the forensic process, and discusses different use cases where this data can be of value. Section 4 describes our collected data, while the possible benefits are described in Section 5. Section 6 discusses limitations and future work, before we conclude in Section 7.

2. Background

Digital forensics relies on a multitude of information sources to gain knowledge, ranging from hard drives and file system artefacts [5] to the dynamic content of RAM [6] to the user files and programs that store information in log files, SQLite databases, or digital images. This leaves the investigator with a broad spectrum of places where to look, where each investigation depends in its specific context and questions to be answered. The general process outline has been defined in both [7] and [8], whereas a great number of current challenges has been discussed in [1]. Another problem is the increasing spectrum of used devices, ranging from smartphones [9] to smart TVs to numerous other types of devices. Most pressing, however, is the general problem that the average case size is constantly increasing [10]. For one this is due to increasing storage capacities of hard drives, with modern hard drives being able to store many terabytes of data that need to be analyzed with respect to the traditional approach of digital forensics. Secondly, cloud storage services commonly push information automatically from device to device, like pictures taken or files edited, leading to duplicate files across devices. Lastly, the density of digital devices surrounding us is increasing, which is also true for the average number of devices per user.

In recent years, numerous forensic models and publications were specifically targeted to reduce the manual work in investigations with a large amounts of data. Among them is the concept of *forensic triage*, which was initially presented in 2006 [11] and quantified more recently in [10] quantified regarding the expected amount of computational power needed. The basic idea is that instead of analyzing all the data there is, only a specific subset of files which are known to be of interest are inspected. Only recently the concept of *sifting collectors* was proposed [2] in which the amount of data to be analyzed is reduced by ignoring known areas on hard drives that are of no particular interest, while still retaining the ability to create bit-identical images if needed. Our approach is different in that it extends the traditional process of forensic imaging by identifying large volumes of both files and file fragments to be either of particular interest (blacklisting), or not of any interest at all as the file is a known, good file (whitelisting).

Both *bulk_extractor* and *hashdb* are two very powerful open-source tools which were published by Simson Garfinkel. Bulk_extractor [12] recursively scans hard drive content, and is able to retrieve information in compressed as well as embedded files like PDFs. It is extremely fast and can use all available cores on a machine to parallelize the task at hand. Hashdb [4] uses efficient algorithms to build a lookup database of hash values much faster than any relational or NoSQL-style database system. It can reliably identify the presence of a given list of target file hash values, and builds on previous work that showed that there is only a small percentage in shared file content on the sector level [13].

3. P2P Networks for Hash Values

The basic idea of our approach is to extend the existing knowledge and applicability on sub-file hashing and hash-based carving by leveraging vast amounts of publicly available hash values. While hashing was previously mainly used to uniquely identify entire files of arbitrary size, our concept presented here extends this to hashing variable-sized sub-file portions. Sub-file hashing [13] as well as hash-based carving [14] allow investigators to search for file fragments by hashing either each hard drive sector or aligned blocks of data. This can also be used if there is not enough time available to prove stochastically the presence or absence of specific files, e.g., in well below an hour and with only a relatively small error margin. We extend these concepts by mapping sub-file hashes with data from peer-to-peer file sharing networks with variable block sizes, both usable for black- and whitelisting of large volumes of files as well as sampling. We thus extend existing tools and concepts, such as bulk analysis of forensic media using *bulk_extractor* [12] and *hashdb*[4].

Peer-to-peer (P2P) file sharing applications and protocols rely heavily on hashing for integrity and as a foundation for parallelization, i.e., simultaneously downloading multiple parts of a file from different users for increased performance. While we used the popular BitTorrent file format for our evaluation, in many cases any application that uses sub-file hashing is directly usable: Dropbox for example, a popular cloud storage service, hashes blocks of 4 megabytes using SHA-256 and stores them in a local SQLite database [15, 16]. These sub-file hash databases can also be privately created and maintained, for example based on files and information within a company or an investigative bureau, but across cases. Our particular contribution is to propose that these pre-computed hash lists can be used to identify files and sub-files on hard drives. With millions and millions of torrent files publicly shared online, *peekaTorrent* uses the fact that each and every torrent file indexes all files and also contains their corresponding SHA-1 hash values. For efficiency, the files are split into equally sized pieces or chunks, solely depending on the overall size of information to be shared [17] in powers of 2 starting with 16kb. Thus, by splitting the hard drives into equally sized chunks and hashing them using SHA-1, it becomes a matter of comparing hash values to possibly identify hard drive content without relying on file system metadata. Also, this information is freely available without participating in any form of file sharing activities, but leveraging the initial seeders computing power in hashing any form of content.

Torrent files have a rather simple structure [17]: they contain generic information, e.g., when the torrent was created, which software was used and the specific information of the data to be shared. This includes the size of the blocks, their SHA-1 hash values, and how many there are. During the creation of the torrent file, all containing files are concatenated, and this stream of data is then split into equally sized blocks (except for the last one which does not need to be aligned with the block length). By default, the data is split into 256-kilobyte blocks, but the user can specify arbitrary block sizes during the creation of the torrent file. The size of the torrent file depends mostly on the number of blocks, because it contains an SHA-1 hash value of 20 Bytes for each block. To uniquely identify the torrent for clients and trackers, an SHA-1 hash value is calculated over a subset of the torrents' stored information: the so-called *info_hash*. Figure 1 shows a graphical representation of the file format as well as an example from a specific torrent file. The dashed line is the information which is hashed to obtain the info_hash value, while for each file the dictionary *files* contains the relative path and the length of the file. *Piece length* is the block size in which the data is split (in the order specified in the *files* field), and the field *pieces* contains the concatenated SHA-1 hash values.

3.1. Problem of Non-Aligned Files

One of the problems when using torrent files is the way these files are created: prior to hashing all chunks, the files are concatenated (in arbitrary order). If a chunk contains parts of two files, we cannot use the resulting hash value. This means that only files which are larger than the piece length can be identified, thus biasing the general applicability towards large files (which is obvious when looking at content from file sharing networks). Figure 2 shows a representation of block hashes in torrents, with the same content as Figure 1: the SHA-1 value of the first piece is usable, as codec.exe spans into the second piece. As such, it can be used to uniquely identify that this file has been stored on the hard drive by hashing any hard drive with the same hashing window as the piece_length of the torrent. This can be readily integrated into bulk_extractor, which already facilitates the necessary requirements by default. If the first file is longer then in our example, and spans, e.g., n pieces in the torrent file, any of these areas on disc can identify the file as long as the data is consecutively stored somewhere. The second piece in Figure 2 is not usable for our proposed methodology, as it contains content from both the first and the second file. While it could theoretically happen that the operating system allocates the information in such a way that the hash value could be used, this is not necessarily the case as the files can be stored at different locations on the hard drive and in different orders. The third piece (i.e., the second piece that contains content from movie.mkv in our example) is usable if the missing length of the file in the beginning is used for *offset hashing* – it is no longer the piece_length which can be used for chunk hashing during acquisition, but rather aligned to the hard drive sectors, which tremendously increases the hash values to be calculated during analysis. Again, this is already integrated in bulk_extractor and the problem remains CPU-bound, which means it is solvable if enough computation power is at hand. The hash value for the last piece is unusable, as it must not be of the same length as the others [17], i.e., there is no padding for torrent files.

In the following, we discuss the different use cases where such a vast amount of file fragment information can be of use in the particular context of digital forensics. Other protocols are probably equally suitable, but have not been investigated in detail for this work, e.g., Kademlia [18] as well as distributed hash tables in general [19] often use SHA-1 hash values for searching.

3.2. Use Case 1: File Whitelisting

File whitelisting is a well-known technique to identify files that are common and of no particular interest during an early phase in digital investigations. One of the most commonly used databases of hash values is the NIST National Software Reference Library (NSRL) reference data set[1] which comprises at the time of writing of more than 43 million file hash values. Most of these hash values include binaries and program libraries for software on Windows, whereas our collected data contains information of relevance independent of the used operating system, and of much larger file size. While NIST also releases block hash values for the first 4k and 8k of about 13 million files, our dataset is able to identify popular files like movies, TV episodes or other commonly shared files on file sharing networks, even if they are deleted and some sectors were already overwritten by the file system.

[1] Online at `http://www.nsrl.nist.gov/`

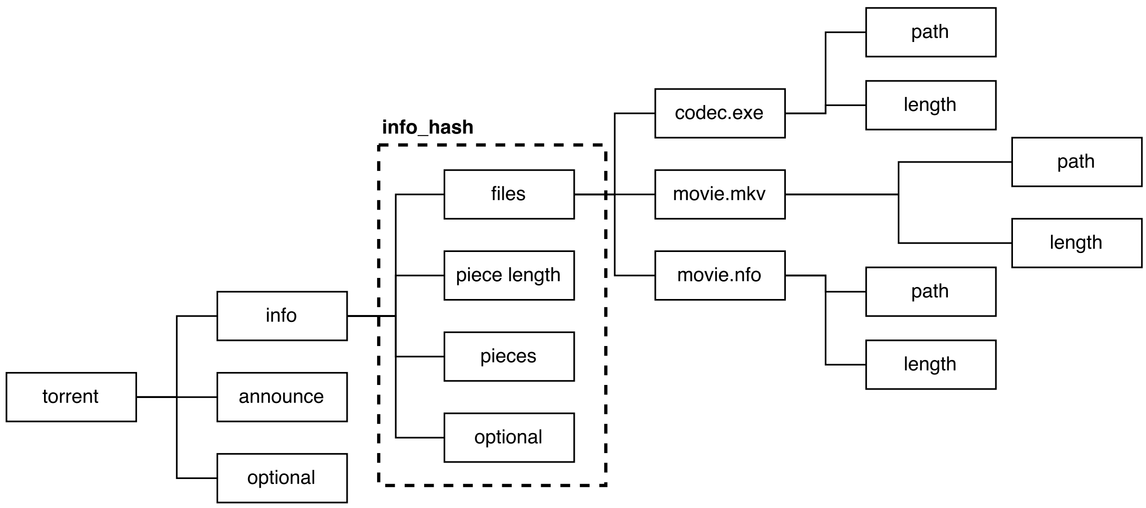

Figure 1: File content in a torrent file

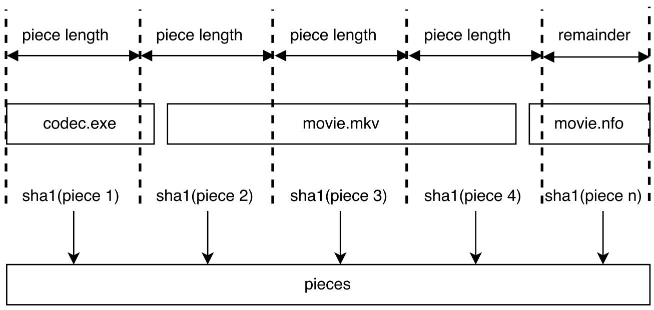

Figure 2: Chunk hashes

3.3. Use Case 2: File Blacklisting

File blacklisting is used to find and identify files of particular interest for a specific investigation. While in our evaluation the usability of our data is mostly limited to cases of copyright infringement, it is still of use for investigations in general and might lead to new insights. Nonetheless, building a private sub-file hash database is always a possibility if a script can be used to hash blocks of arbitrary length of, e.g., all e-mail attachments in a company, all files on a Sharepoint server or source code within a company. This could also include outright illegal material like pictures and videos related to child pornography. Instead of using perceptional hashing [20] – as used by online services like Twitter and Facebook to detect such files [21] – sub-file hash values of variable block length can further identify files like these without access to such perceptionally hashed data.

3.4. Use Case 3: File Fragment Identification

By default, file systems in modern operating systems do not overwrite files once they are deleted, but rather delete the index pointing to the data or mark the affected storage areas as free-to-use [5]. Depending on the operating system and the file system in use, as well as the actual user behavior, it is usually unpredictable when a specific area will be overwritten. Both methods in our approach described so far work for partially overwritten files, as they do not rely on file system metadata. This was already argued in [13] for sector hashing. As long as the data on a disc is not completely overwritten and leaves at minimum the piece length of the torrent files untouched, peekaTorrent will find it.

3.5. Shifting the Bottleneck

Considering these three use cases, the overall performance scales linearly with the number of available CPU-cores, similar to bulk_extractor. Sub-file hashing can leverage multi-core CPUs and scales with the number of available cores. As the file system metadata is not needed, there is also no need for disk seek operations. All the data from the hard drive can be split in constantly sized chunks, and processed recursively using the hashdb scanner within bulk_extractor.

4. Evaluation

To evaluate our methodology we implemented all the steps of the processing outline described above. This includes software we wrote to collect torrent files from the Internet and tools to process and use them within the context of a forensic investigation, see `https://www.peekatorrent.org`. This section shows and underlines the applicability of the proposed approach and the methods applied for gathering torrents on a large scale.

4.1. Data Collection

Collecting a large number of torrents from the Internet is non-trivial, as new torrents are added constantly and older torrents become unavailable once they are no longer shared.

Only a minority of websites hosts the torrent files containing all the sub-file hash values themselves, but rather rely on sharing magnet links that point to the information in the completely decentralized *distributed hash tables* (DHTs) [22].

To collect torrent files we focused on the following three main sources: (i) The Pirate Bay[2], (ii) kickassTorrents[3], and (iii) various data dumps, e.g., from *openBay*[4]. For (i) and (ii) we implemented a crawling framework which recursively crawls and parses both websites for every magnet link listed there. After that we extracted the torrent info_hashes from the magnet links and constructed a download link for the torrent cache website `https://torcache.net/`. For (iii) and those torrent files which weren't hosted at torcache.net we implemented a DHT lookup service, similar to the one Wolchok et al. used in their work [23]. The crawlers for (i) and (ii) were crawling the entire websites, including all subcategories to get the full archive for a specific point in time (January 2016 in our case).

From the various openBay dumps we were able to extract close to 30 million info_hashes. The dataset from isohunt contained 7.8 million info_hashes, while the complete archive for openBay included 23.5 million hashes. Both data sets were created after the police raid against Pirate Bay in December 2014 caused the website to be shut down. Previously generated data sets also include one notable xml dump of the Pirate Bay from February 2013 (about 2 million info_hash values). Not all of these files were retrievable using the DHTs, in fact only a small fraction and in particular only newer files. The biggest fraction of torrent files we collected came from kickassTorrents and torcache.net, as torcache.net is used by default to distribute torrent files on behalf of kickassTorrents. So far we have collected 2.3 million torrent files, which we share with the reviewers and will later release them publicly. Our data collection is still going on, and as such the data we collected can only be considered a snapshot in time. Further processing was then done using Python as well as *hashdb*, which was used to efficiently store and query the sub-file hash values.

4.2. Theoretic Evaluation

Fragmentation of files can be a limiting factor using real cases, as for each time a file is fragmented one chunk (of arbitrary length) is no longer identifiable. Since there is no public instance of a SHA-1 pre-image attack, finding a small number of chunks using peekaTorrent has a very small likelihood to be coincidental and can be used for further analysis steps during the investigation. Compared to previous work [13, 4], the number of false positives is greatly reduced, as the block length used for hashing is larger than the previously used sector/cluster size of 512 or 4096 bytes. Hashing a larger file block, e.g., 256 kilobytes, drastically reduces the probability of resulting in the same hash value (for all files independent of each other). This also implies that shared file content across files, such as the ramping structure for Microsoft Office files as discussed in [13], is evaded as the block length increases.

[2]`https://thepiratebay.se/` and its alternative TLDs

[3]`https://kat.cr/`

[4]`https://github.com/isohuntto/openbay-db-dump`

5. Results

Overall, we collected and analyzed more than 2.3 million torrent files. These torrents comprise 3.3 billion block hash values. From these 3.3 billion block hash values, approximately 48% (or 1.62 billion hash block values) are usable to identify millions of files using various block length. Another 50% (or 1.66 billion hash block values) are usable even though the files do not align with the torrent chunking. 1.1% of the 3.3 billion hash values (or 39 million hash block values) are not usable for our approach, as the blocks and their corresponding hashes comprise content of two or more files. The exact numbers for the most popular torrent block lengths of 2^n (for various n) is shown in Table 1, with exotic chunk sizes omitted (n=2,871) for the sake of brevity.

From the 2.3 million torrent files we are able to identify 2.6 petabytes of data using TeekaTorrent, or 32 million files. Regarding only the most common chunk sizes with 100,000 or more torrent files found using our methodology, we are left with 2.1 million torrents. The pre-computed hashdb databases as well as the raw torrent files and the source code used for this paper can be found on our website `https://www.peekatorrent.org`.

5.1. hashdb

We then imported the usable sub-file hash values for all torrents with a piece length of 256k into hashdb [4]. As it can be seen in Table 1, this sums up to 631 million hash values. From these 631 million only 474 million are unique, because of duplicate sub-file hash values. This is due to the fact that the same files can be contained in different torrents, e.g., duplicates for each kickassTorrents and Pirate Bay. Torrent files that became repackaged with different files or file ordering can be another reason to cause this rather large discrepancy. hashdb can then be used to deny that a given sub-file hash value is part of the database using Bloom filters. Otherwise the database is queried, and both filename and info_hash are returned if a corresponding hash value is found. All the features and APIs provided by hashdb are thus fully usable, and the entire project is well documented and active[5].

While the majority of sub-file hash values are unique within the data we collected (474 million), the long tail of duplicates can be seen in Figure 3. The x-axis accounts

[5]`https://github.com/NPS-DEEP/hashdb`

block length	torrents	chunks	usable chunks		offset chunks		unusable chunks
16k	75k	146m	123m	84%	22m	15%	305k
32k	95k	171m	112m	65%	58m	34%	662k
64k	335k	217m	124m	57%	90m	41%	2m
128k	201k	227m	115m	50%	109m	48%	2m
256k	669k	1.329b	631m	47%	690m	51%	8m
512k	297k	401m	201m	50%	194m	48%	5m
1024k	307k	357m	165m	46%	187m	52%	5m
2048k	170k	201m	75m	37%	121m	60%	4m
4096k	161k	229m	58m	25%	162m	70%	8m
8192k	18k	27m	8m	30%	17m	65%	975k
16384k	2k	3m	315k	9%	2m	84%	198k
Sum:	2.3m	3.314b	1.615b	48%	1.658b	50%	39m

Table 1: Results of data collection for 2.3 million torrent files

for the number of duplicates found, starting from hash values with 10 duplicates or more. Note that the y-axis is log-scale. In the data there are also 17.8 million distinct sub-file hashes that occur twice, 2.5 million that occur three times, and about 440,000 that occur four times. We speculate that these hashes are again caused by some form of release group information or an embedded URL. The by-far largest number of duplicates observed was caused by one particular hash that occurs 8,462,788 times. We would speculate that this is caused by the “null” hash, for data areas that contain only zeros.

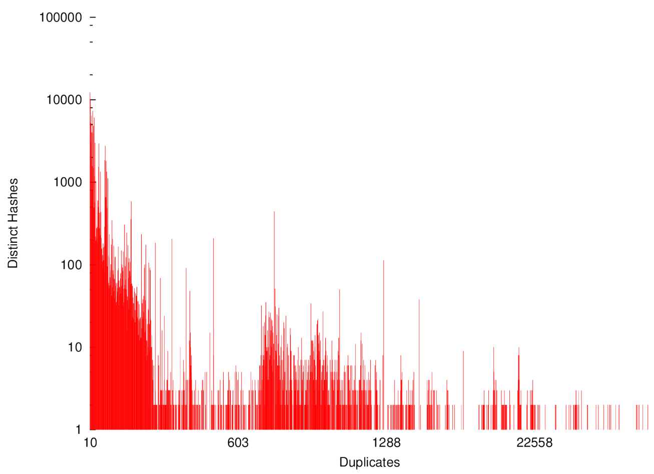

Figure 3: Distribution of sub-file hash duplicates

5.2. Real Runtime on Limited Hardware

To evaluate our approach further, we took a 5-year old notebook and created a one gigabyte image from a USB thumb drive. The notebook was a Lenovo X200s, with a Core 2 Duo processor (L9400), 4GB of RAM and a regular hard drive. On the thumb drive we stored the ISO file for the current version of Ubuntu Desktop, which we downloaded over BitTorrent. We created a fresh hashdb database, and seeded it with the extracted SHA-1 hashes of the torrent file. Overall, we extracted 1158 hash values for the Ubuntu image, the chunk size was 512k. We then used a custom module for bulk_extractor to generate SHA-1 hashes of all blocks bulk_extractor processes, and disabled all other plugins.

Running bulk_extractor with solely the SHA-1 plugin activated on the notebook took 220 seconds to process the 1GB image file. Since the CPU has two cores, two threads were spawned to process the image. From the 1158 chunks, 1154 were successfully identified using peekaTorrent. Three chunks could not be found since the file was stored fragmented in three fragments (verified manually using *fiwalk*), and the last hash value is unusable as it has a different chunk length. Running the same analysis on a modern Xeon with 8 cores plus Hyper-Threading took less than 23 seconds. Running the same image against the hashdb database of all 474 million chunk hashes took 38 seconds. Since we do not aim to evaluate the performance of either bulk_extractor or hashdb, we do not go into details of further performance numbers. Also, the average fragmentation on hard drives depends heavily on the type of usage, size and operating system. Measuring this for the average case is beyond the scope of this paper.

6. Discussion

Our results show that a rather large number of block hash values is usable to identify files based on the data we collected from BitTorrent files, somewhere close to 98%. Due to the nature of file sharing networks and the content distributed there we assume that this is possibly biased, that these networks commonly share large files like movies in high quality. We did not investigate the distribution of filenames and file sizes to what extend one can expect that the largest file is the first in the torrent file. We assume that this is specific to the application that created the torrent, as this is not specified in the file format of BitTorrent [17].

Half of the usable chunk hashes come with an arbitrary offset due to the placement of the affected files. This is caused by the particularities of BitTorrent files. However, since bulk_extractor processes pages of memory without any file system information, these artefacts are also retrievable (as long as the file is larger than the chunk size). Other sources for sub-file hashing have to be investigated, like other P2P protocols or cloud storage solutions such as Dropbox. We expect similar functionality from other cloud storage solutions like Google Drive, OwnCloud or Microsoft OneDrive as well, where the local data structures could be used as a source for history hash values. Still, using the data we collected we can identify up to 2.6 petabytes of data for 3.3 billion chunks. We expect these values to increase, as we will keep collecting data and publishing it on our website.

Regarding the forensic application and typical use case, many scenarios come to mind. First, it depends on the data sources used for seeding the sub-file hashing – this can be for example all sent e-mail attachments in a company, a stack of sensitive corporate documents or encrypted data blobs in the corporate context. Secondly, this can be easily enlarged by investigators via adding data from private repositories of interesting files, file archives or any other data source at hand – like USB thumb drives – or portable hard drives, and hashing it in sub-file chunks. Another example could be the cross-linking of files between hard drives: if any of the hard drives during an investigation is hashed with a particular chunk size, all other related drives can use this information to identify non-fragmented overlaps. After all, this was obviously the original motivation behind the tight connection between bulk_extractor and hashdb. Foremost, peekaTorrent allows for hard drives without any meta information at all to find clues on the content – as long as the hard drive is not encrypted.

6.1. Limitations

While 2.6 petabytes of identifiable files sounds like a lot, its usefulness depends on the particular kind of investigation. If the goal is to whitelist as many files and file fragments as possible on a diverse set of machines, our approach looks promising. As always in digital forensics, it depends, however, on the specific context of the investigations and the questions of interest. For more specific investigations it depends on the type and volume of data – creating sub-file hash values of variable block length is easily scriptable, so if a large repository of files is available, our methodology is applicable. This can be, for example all attachments from an e-mail server, malicious files like malware from anti-virus companies, or even smaller sets of files with a direct connection to an investigation.

Another limitation is the behavior of storage devices, operating systems and file systems: SSDs regularly delete artefacts within the free space using the TRIM command [24], and depending on the operating system and file system, fragmentation can occur. There are no current numbers on the amount of fragmentation happening, with the latest study on file system metadata being already close to a decade old [25]. Also, the approach only works for files which have at least a file size bigger then the hashing window respectively the torrent piece length. Based on our findings with peekaTorrent, only files with a minimal size of 16 kilobytes are identifiable, while a vast amount of files needs to have at least 256 kilobytes due to the nature of the seeding data.

6.2. Future Work

For future work we plan to evaluate our approach using real hard drives and/or cases. It is generally hard to find representative cases or hard drives, but measuring the applicability of peekaTorrent is our next step. Furthermore, we plan to investigate the usage of GPUs for variable block length hashing. We also plan to make our tools and data collections more readily applicable, by releasing tools for creating and querying sub-file hash values easy as part of the forensic process. Lastly, our data collection could be enhanced by focusing on popular file torrents and by collecting more files over time (which is expected to continue for the near future) from additional torrent websites as well as from DHT crawlers.

7. Conclusion

In this paper we have demonstrated how vast amounts of sub-file hash values can be of use in digital forensics. We evaluated the idea of using torrent files from popular file sharing platforms and collected more than 2.3 million torrent files for our analysis. Based on these torrent files we extracted more then 3 billion SHA-1 sub-file hash values and were able to identify up to 32 million files or 2.6 petabytes of information using this data set. Both the collected data and the written software tools are available under open source licenses.

Acknowledgements

We thank our shepherd Judson Powers for guiding us to a highly improved version of the initial paper. We owe particular thanks to our student Daniel Gasperschitz for writing the SHA-1 module for bulk_extractor. This research was supported by the Austrian Research Promotion Agency (FFG) through the Bridge Early Stage grant P846070 (SpeedFor) and the COMET K1 program.

References

[1] S. L. Garfinkel, Digital forensics research: The next 10 years, Digital Investigation 7 (2010) S64–S73.

[2] G. G. Richard III, J. Grier, Rapid forensic acquisition of large media with sifting collectors, Digital Investigation 14 (2015) S34–S44.

[3] S. Neuner, M. Schmiedecker, E. Weippl, Effectiveness of file-based deduplication in digital forensics, Security and Communication Networks.

[4] S. L. Garfinkel, M. McCarrin, Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb, Digital Investigation 14 (2015) S95–S105.

[5] B. Carrier, File system forensic analysis, Addison-Wesley Professional, 2005.

[6] M. H. Ligh, A. Case, J. Levy, A. Walters, The art of memory forensics: detecting malware and threats in windows, linux, and Mac memory, John Wiley & Sons, 2014.

[7] D. Brezinski, T. Killalea, Guidelines for Evidence Collection and Archiving, RFC 3227 (Best Current Practice) (Feb. 2002).

[8] K. Kent, T. Grance, H. Dang, Nist special publication 800-86, Guide to Integrating Forensic Techniques into Incident Response.

[9] A. Hoog, Android forensics: investigation, analysis and mobile security for Google Android, Elsevier, 2011.

[10] V. Roussev, C. Quates, R. Martell, Real-time digital forensics and triage, Digital Investigation 10 (2) (2013) 158–167.

[11] M. K. Rogers, J. Goldman, R. Mislan, T. Wedge, S. Debrota, Computer forensics field triage process model, in: Proceedings of the conference on Digital Forensics, Security and Law, Association of Digital Forensics, Security and Law, 2006, p. 27.

[12] S. L. Garfinkel, Digital media triage with bulk data analysis and bulk_extractor, Computers & Security 32 (2013) 56–72.

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

[14] S. Garfinkel, A. Nelson, D. White, V. Roussev, Using purpose-built functions and block hashes to enable small block and sub-file forensics, digital investigation 7 (2010) S13–S23.

[15] D. Kholia, P. Wegrzyn, Looking inside the (drop) box., in: 7th USENIX Workshop on Offensive Technologies (WOOT), 2013.

[16] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, E. R. Weippl, Dark clouds on the horizon: Using cloud storage as attack vector and online slack space., in: USENIX Security Symposium, San Francisco, CA, USA, 2011, pp. 65–76.

[17] B. Cohen, The bittorrent protocol specification, bep-3, online at http://www.bittorrent.org/beps/bep_0003.html.

[18] P. Maymounkov, D. Mazieres, Kademlia: A peer-to-peer information system based on the xor metric, in: Peer-to-Peer Systems, Springer, 2002, pp. 53–65.

[19] M. Steiner, T. En-Najjary, E. W. Biersack, Long term study of peer behavior in the kad dht, IEEE/ACM Transactions on Networking (TON) 17 (5) (2009) 1371–1384.

[20] F. Breitinger, B. Guttman, M. McCarrin, V. Roussev, D. White, Approximate matching: definition and terminology, URL http://csrc. nist. gov/publications/drafts/800-168/sp800_168_draft. pdf.

[21] T. Ith, Microsoft's photodna: Protecting children and businesses in the cloud, online at https://news.microsoft.com/features/microsofts-photodna-protecting-children-and-businesses-in-the-cloud/ (2015, July 15th).

[22] C. Zhang, P. Dhungel, D. Wu, K. W. Ross, Unraveling the bittorrent ecosystem, Parallel and Distributed Systems, IEEE Transactions on 22 (7) (2011) 1164–1177.

[23] S. Wolchok, J. A. Halderman, Crawling bittorrent dhts for fun and profit., in: 4th USENIX Workshop on Offensive Technologies (WOOT), 2010.

[24] G. Bonetti, M. Viglione, A. Frossi, F. Maggi, S. Zanero, A comprehensive black-box methodology for testing the forensic characteristics of solid-state drives, in: Proceedings of the 29th Annual Computer Security Applications Conference, ACM, 2013, pp. 269–278.

[25] N. Agrawal, W. J. Bolosky, J. R. Douceur, J. R. Lorch, A five-year study of file-system metadata, ACM Transactions on Storage (TOS) 3 (3) (2007) 9.