Chapter 8

HASH-BASED FILE CONTENT IDENTIFICATION USING DISTRIBUTED SYSTEMS

York Yannikos, Jonathan Schluessler, Martin Steinebach, Christian Winter and Kalman Graffi

Abstract A major challenge in digital forensics is the handling of very large amounts of data. Since forensic investigators often have to analyze several terabytes of data in a single case, efficient and effective tools for automatic data identification and filtering are required. A common data identification technique is to match the cryptographic hashes of files with hashes stored in blacklists and whitelists in order to identify contraband and harmless content, respectively. However, blacklists and whitelists are never complete and they miss most of the files encountered in investigations. Also, cryptographic hash matching fails when file content is altered even very slightly. This paper analyzes several distributed systems for their ability to support file content identification. A framework is presented for automated file content identification that searches for file hashes and collects, aggregates and presents the search results. Experiments demonstrate that the framework can provide identifying information for 26% of the test files from their hashed content, helping reduce the workload of forensic investigators.

Keywords: File content identification, hash values, P2P networks, search engines

1. Introduction

One of the principal challenges in digital forensics is the efficient and effective analysis of very large amounts of data. The most popular approach for filtering data in forensic investigations is to compute the cryptographic hashes of files and search for the hashes in large blacklists and whitelists. Since an important property of cryptographic hashes is collision resistance, they are ideally suited for file content identification. The cryptographic hash of a file is calculated solely from the file content – file metadata such as filename and timestamps are not considered. MD5 and SHA-1 are the most commonly used cryptographic hash functions that are used to compute file hashes.

Filtering files by searching for their hash values in blacklists and whitelists is effective and efficient. However, it is impossible to create lists that contain the file hashes of all harmful and harmless files. Moreover, the approach does not work well for multimedia files – changing the format of a picture, changes its file hash, and the new file hash does not match the original file hash.

This paper seeks an alternative to using blacklists and whitelists containing file hashes. It describes a framework that, given file hash values, can automatically engage sources such as P2P file sharing networks and web search engines to find useful information about the corresponding files. The useful information includes filenames because they typically describe the file contents. The framework aggregates, ranks and visually presents all the search results to assist forensic investigators in identifying file content without having to conduct manual reviews. Experimental results are presented to demonstrate the effectiveness and efficiency of the framework.

2. Hash-Based File Content Identification

Cryptographic hashes, especially MD5 and SHA-1, are commonly used in forensic investigations to identify content indexed in whitelists and blacklists. One of the largest publicly available databases is the NIST National Software Reference Library (NSRL), which reportedly contains about 26 million file hashes suitable for whitelisting [12]. Two publicly-available blacklisting databases are VirusTotal [19] and the Malware Hash Registry (MHR) [18] that can be searched for the hashes of malware files. Other databases maintained by law enforcement agencies contain the hashes of contraband files such as child pornography images, but these databases are generally not available to the public.

Tools have been developed to find known files in cloud system software and P2P client software. Examples are PeerLab [9] and FileMarshal [1] that find installed P2P clients and provide information about shared files and search history.

2.1 Advantages

Using cryptographic hashes for file content identification is effective because the hashes are designed to be resistant to pre-image and second pre-image attacks – it is extremely difficult to find the file corresponding to a given hash value and to find another file that has the same hash value as a given file. Calculating file hashes using MD5 and SHA-1 is also efficient – the hash of a 100 MB file can be computed in less than one second on a standard computer system.

Another advantage is that only the file content is used to compute the file hash; no other metadata associated with the file is required. This means that a file that was modified to hide its illegal content, e.g., by giving it an innocuous filename or file extension or by changing its timestamps, can still be identified using its hash if the value is in a blacklist.

2.2 Limitations

Cryptographic hashing does not work well for multimedia files. For example, merely opening and saving a JPEG file results in a new file with a different hash value from the original file. Also, a contraband file that is already blacklisted may be distributed using anti-forensic techniques that make the automated identification of the file extremely difficult, if not impossible.

For example, a criminal entity could create a web server module as a content handler for contraband images. This module could be designed to randomly select and change a small number of pixels in each image before it is downloaded and viewed using a web browser. The modified image would be visually indistinguishable from the original contraband image. However, cryptographic hashing would fail to identify the modified image as being essentially the same as the original image.

Skin color detection algorithms are commonly used to detect child pornography. However, skin color detection is not a reliable technique because it has a high false positive rate. Also, skin color detection alone would not be able to discriminate between child pornography and legal pornography.

Steinebach, *et al.* [16] have proposed an efficient and accurate approach that uses robust hashing to identify blacklisted child pornography. However, robust hashing has not as yet been used for file content indexing by P2P protocols, web search engines and most hash databases. For this reason, we do not consider robust hashing in this paper.

3. Distributed Systems for Hash-Based Searches

This section examines various distributed systems to assess their suitability for file hash searches. In particular, the section focuses on P2P file sharing networks, web search engines and online databases of indexed file hashes.

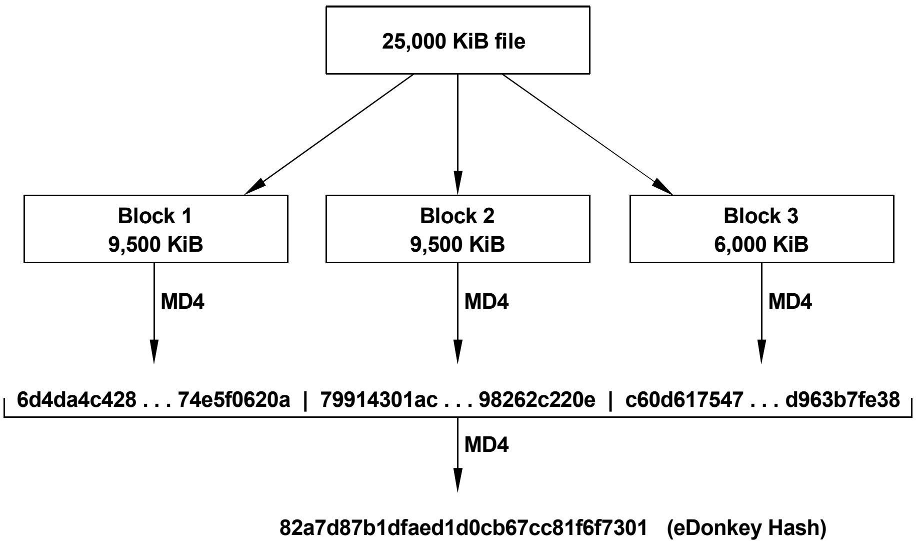

25,000 KiB file
Block 1
9,500 KiB
Block 2
9,500 KiB
Block 3
6,000 KiB
MD4
MD4
MD4
6d4da4c428 . . . 74e5f0620a | 79914301ac . . . 98262c220e | c60d617547 . . . d963b7fe38
MD4
82a7d87b1dfaed1d0cb67cc81f6f7301 (eDonkey Hash)

Figure 1. Creation of an eDonkey hash of a sample 25,000 KiB file.

3.1 P2P File Sharing Networks

P2P file sharing networks are immensely popular. A recent Cisco study [3] reported that P2P networks contributed to about 15% of the global IP traffic in 2011. Another study [15] identified P2P traffic as the largest share of Internet traffic in every region of the world, ranging from 42.5% in Northern Africa to almost 70% in Eastern Europe. The most popular P2P protocols are BitTorrent, eDonkey and Gnutella.

Although BitTorrent [4] is the most popular P2P protocol with 150 million active users per month [2], the protocol does not allow hash-based file searches. BitTorrent uses "BitTorrent info hashes" (BTIHs) instead of cryptographic hashes of the files that are shared. Since BTIHs are created from data other than just file content (e.g., filenames), BitTorrent is not suitable for our hash-based file search approach.

Another popular P2P protocol is eDonkey with 1.2 million concurrent online users as of August 2012 (aggregated from [6]). The eDonkey protocol is well suited to hash-based file searches because the "eDonkey hash" used for indexing is a specific application of the MD4 cryptographic hash [10]. Figure 1 illustrates the process of creating an eDonkey hash. A file is divided into equally-sized blocks of 9,500 KiB and an MD4 hash value is computed for each block. The resulting hashes are then concatenated to a single byte sequence for which the final MD4 hash (eDonkey hash) is computed.

An eDonkey peer typically sends its hash search request to one of several eDonkey servers, which replies with a list of search results. eDonkey also allows decentralized searches using its kad network, a specific implementation of the Kademlia distributed hash table [11]. Interested readers are referred to [17] for additional details about kad.

The Gnutella protocol uses SHA-1 for indexing shared files. Searching for SHA-1 hashes was initially supported by Gnutella, but it is now disabled in most Gnutella clients for performance reasons. The `gtk-gnutella` client still supports SHA-1 hash searches. However, this client yields very few results because a SHA-1 hash search request is dropped by all other clients that do not support SHA-1 hash searches. Also, in test runs, we have observed that `gtk-gnutella` has very long response times for SHA-1 hash searches. Therefore, Gnutella is not suitable for hash-based file searches for reasons of effectiveness and efficiency.

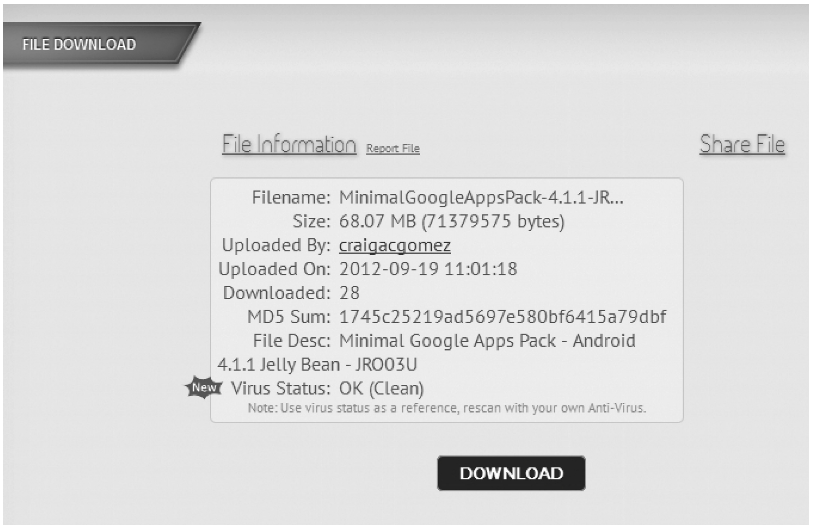
3.2 Web Search Engines

According to [13], Google is by far the most popular web search engine with about 84.4% market share; next come Yahoo! with 7.6% and Bing with 4.3%. Since the success of search engines is based on gathering as much Internet content as possible while applying powerful indexing mechanisms to support fast querying of very large data sets, they provide an excellent basis for hash-based file searches. Because many web sites (e.g., file hosting services) also provide file hashes such as MD5 or SHA-1 as checksums, it is very likely to find valuable information such as a popular filename when searching for a file hash. Figure 2 shows a screenshot of a file hosting website [5] that provides useful information about a file with a specific MD5 file hash value.

3.3 Online Hash Databases

In addition to NIST's National Software Reference Library (NSRL) [12], several other hash databases can be searched online or downloaded and searched locally. One example is the Malware Hash Registry (MHR) [18], which contains the hashes of files identified as malware. Another is SANS Internet Storm Center's Hash Database (ISC) [14], which reportedly searches the NSRL as well as MHR for hashes. MHR and ISC both support search queries via the DNS protocol by creating DNS TXT requests that use specific file hashes as a subdomain of a given DNS zone.

Figure 3 shows an example of querying the ISC database using the DNS protocol. The source "NIST" specifies the NSRL database. The result contains the indexed filename `Linux_DR.066` belonging to the hash.



Filename: MinimalGoogleAppsPack-4.1.1-JR...
Size: 68.07 MB (71379575 bytes)
Uploaded By: craigacgomez
Uploaded On: 2012-09-19 11:01:18
Downloaded: 28
MD5 Sum: 1745c25219ad5697e580bf6415a79dbf
File Desc: Minimal Google Apps Pack - Android 4.1.1 Jelly Bean - JRO03U
Virus Status: OK (Clean)
Note: Use virus status as a reference, rescan with your own Anti-Virus.

Figure 2. File hosting website with useful information about an MD5 file hash.

```
  > dig +short B47139415F735A98069ACE824A114399.md5.dshield.org
               TXT "Linux_DR.066 | NIST"
```

Figure 3. Querying the ISC database for a file hash using `dig`.

4. File Content Identification Framework

When searching for a specific file hash, the most important information directly connected with the hash is a commonly-used filename because it typically describes the file content in a succinct manner. In order to automatically search different sources and collect the results, we developed a prototype that implements the following modules to search P2P file sharing networks, web search engines and hash databases for file hashes:

- **eDonkey Module:** This module is used to search eDonkey P2P networks for eDonkey hashes.
- **Google Module:** This module is used to search Google for MD5 and SHA-1 hashes.
- **Yahoo! Module:** This module is used to search Yahoo! for MD5 and SHA-1 hashes.

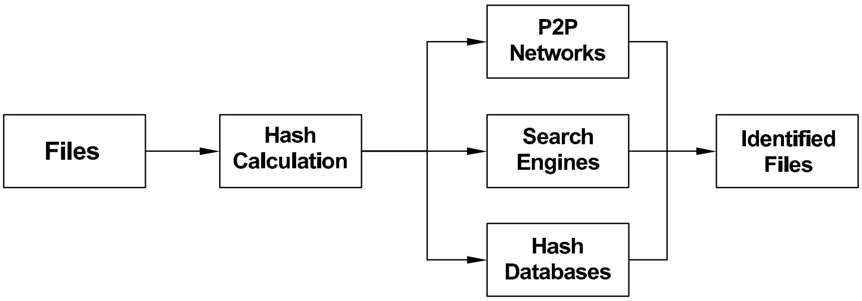


Figure 4. File content identification using cryptographic hashes.

- **NSRL Module:** This module is used to search the NSRL for MD5 hashes. A copy of the NSRL was downloaded in advance to perform local database searches.
- **ISC Module:** This module is used to search ISC for MD5 hashes.
- **MHR Module:** This module is used to search MHR for MD5 hashes.

Figure 4 shows the file content identification process of our framework. The process has five steps:

1. Select a directory containing the input files.
2. Calculate the MD5, SHA-1 and eDonkey hashes for each file in the directory and its subdirectories.
3. Use the hashes to query P2P networks, search engines and hash databases.
4. Collect all the results and aggregate the identical results.
5. Present the results in a ranked list.

5. Evaluation

In order to evaluate our framework, we used a test set of 1,473 different files with a total size of about 13 GB. Although the test set was rather small, it provided a good representation of files that would be of interest if encountered during a forensic investigation. The test set included the following categories of files:

- **Music Files:** A total of 650 audio files, mainly containing music. These files were randomly selected from three large music libraries, several music download websites and sample file sets.

- **Picture Files:** More than 500 pictures, mainly from several public and private picture collections (e.g., 4chan image board).

- **Video Files:** A total of 34 video files, mainly YouTube videos, entire movies and episodes of popular TV series.

- **Document Files:** A total of 240 documents, including e-books in the PDF, EPUB and DOC formats, mainly from a private repository.

- **Miscellaneous Files:** Several archive files, Windows and Linux executables, malware and other miscellaneous files, mainly from a private download repository.

5.1 Search Result Characteristics

Our framework collects metadata found by searching P2P file sharing networks, search engines and hash databases. The search results are presented in a ranked form based on how many different sources found the same metadata. In general, the more specific a filename that is found when searching for a file hash, the higher the filename is ranked. Along with the filename, the framework also collects other metadata (e.g., URLs and timestamps), if available.

Table 1 shows the results of a hash-based search using the framework to identify file content. Specifically, the table shows the filenames found for three sample files from the test set: one audio file, one picture file and one video file. As mentioned above, locally-available metadata such as a local filename is not considered by our framework because it cannot be trusted and may be forged to hide incriminating information. The sample video file turned out to be a good example of the usefulness of the framework. Since the local filename is `Pulp Fiction 1994.avi`, a forensic investigator is likely to conclude that the file contains the popular movie *Pulp Fiction* directed by Quentin Tarantino. However, in eDonkey networks, our framework found several filenames describing files with pornographic content. Most of the filenames found in our evaluation provided enough information to identify the content of the corresponding files.

5.2 Effectiveness

For 382 files (26%) in the test set of 1,473 files, we were able to find a commonly-used filename that helped identify the file content. 220 of these files (almost 58%) yielded unique results, i.e., only a single source produced results for these files. Figure 5 shows the total percentage of

Table 1. Results of a hash-based search.

Sample File	**Source**	**#Results**	**Most Common Filenames (sorted by #results in descending order)**
Picture File	Google	1,100	`Lighthouse.jpg` `8969288f4245120e7c3870287cce0ff3.jpg`
	Yahoo!	7	`Lighthouse.jpg`
	eDonkey	7	`Lighthouse.jpg` `sfondo hd (3).jpg`
	ISC	1	`Lighthouse.jpg`
Audio File	eDonkey	31	`Madonna feat. Nicki Minaj M.I.A. Give Me All Your Luvin.mp3` `Madonna - Give me all your lovin [ft.Nicki Minaj & M.I.A.].mp3` `Madonna ft. Nicki Minaj & M.I.A. - Give Me All Your Luvin'.mp3`
Video File	eDonkey	130	`La.Loca.De.la.Luna.Llena.Spanish. XXX.DVDRip.www.365(...).mpg` `La Loca De La Luna Llena (Cine Porno Xxx Anastasia Mayo(...).avi` `MARRANILLAS TORBE...avi` `Pelis porno - La Loca De La Luna Llena (Cine Porno Xxx(...).mpg`

files for which the individual sources successfully found a filename, along with the aggregated numbers.

Most of the results were obtained by searching eDonkey and Google. In comparison, searching Yahoo!, ISC and MHR produced no additional results. Apart from Google and eDonkey, the NSRL was the only other source that produced unique results.

Figure 6 shows the search results for each category. eDonkey was very effective at finding filenames using the hashes of video files – it produced results for 53% of the video files in the test set. Furthermore, eDonkey found filenames for the hashes of 18% of the picture files, 19% of the audio files and 24% of the miscellaneous files.

Google yielded the most results for miscellaneous files; it found filenames for 66% of the hashes. The NSRL and ISC hash databases yielded the filenames of most system files, sample wallpapers and sample videos that are typically whitelisted. Searching for documents such as e-books and Microsoft Word files turned out to be ineffective. Google was still

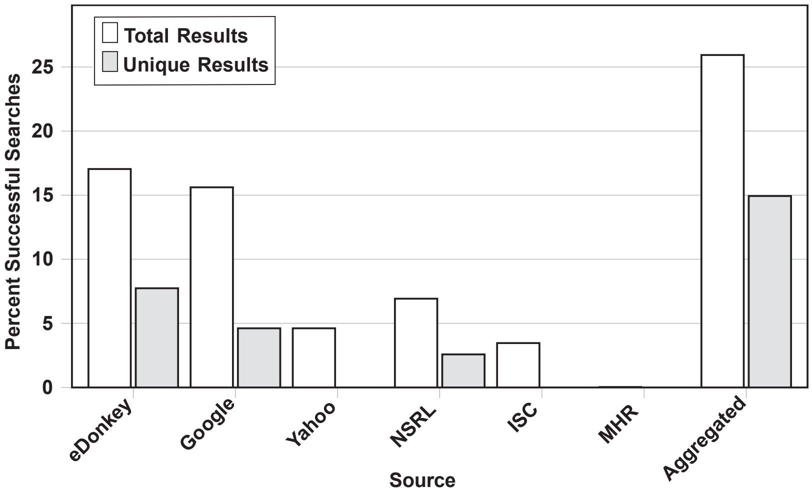


Figure 5. Successful searches for each source and aggregated numbers.

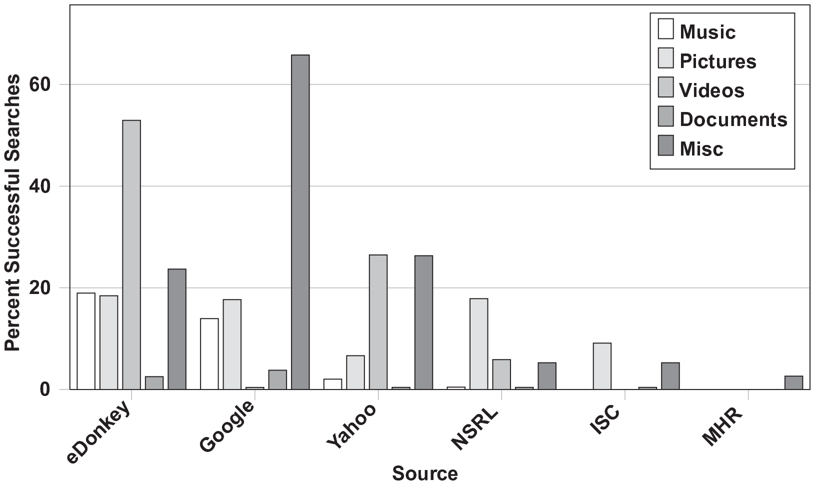


Figure 6. Successful searches for each source by category.

the most successful source, but it found filenames for just 3.8% of the document file hashes.

In addition to MD5 and SHA-1 hashes, we used Google to search for hashes in the SHA-2 family because they are supported by many forensic analysis tools. Figure 7 shows the results. The SHA-224, SHA-256, SHA-384 and SHA-256 hashes only yielded a small number of results with no unique results. This leads us to believe that using MD5 and SHA-1 is adequate for reliable file content identification based on cryptographic hashes.

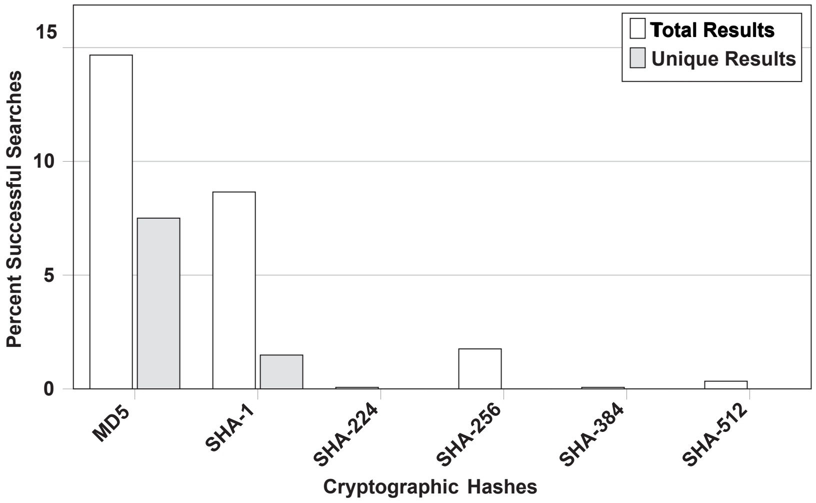


Figure 7. Successful Google searches using cryptographic hashes.

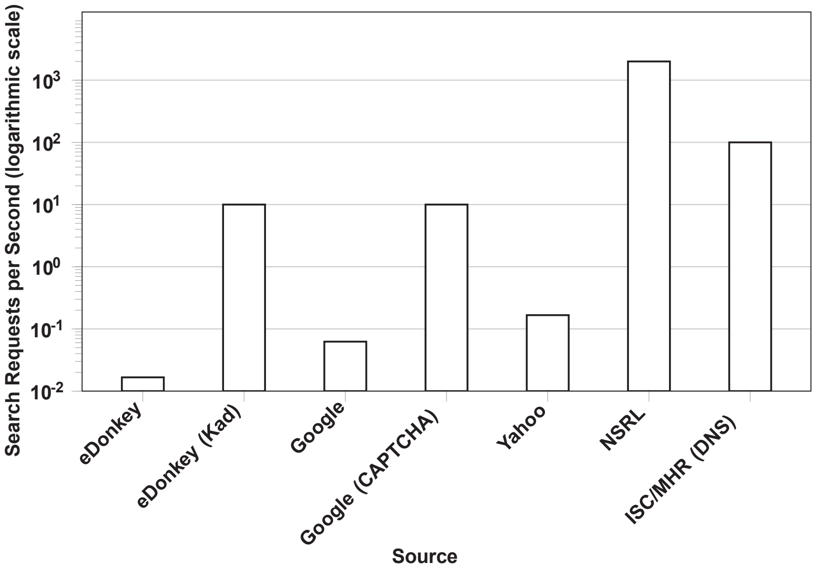


Figure 8. Search requests per second for each source.

5.3 Efficiency

To evaluate the efficiency of our framework, we measured the average number of search requests per second that were sent to each source. Figure 8 shows the results. We found that some sources (e.g., hash databases) could be searched very quickly. Other sources implement mechanisms such as CAPTCHAs to throttle the number of requests received during a given time period.

eDonkey servers, which are used for file search, generally have security mechanisms that prevent request flooding. We discovered that even if just one request was sent every 30 seconds, the servers quickly stopped replying to additional requests. Eventually, we found that sending one request every 60 seconds produced results without being blocked. However, using the decentralized kad network of eDonkey (where peers are directly contacted with search requests), we were able to send requests much faster, at a rate of about 10 requests per second.

The locally-available NSRL hash database was the fastest; it handled about 2,000 requests per second. The MHR and ISC databases were also fast, about 100 search requests per second were possible using their DNS interfaces.

In the case of Google and Yahoo!, the average rates for search requests were one request in sixteen seconds and one request in six seconds, respectively. Sending faster requests caused Yahoo! to completely ban our IP addresses, with increasing ban durations of up to 24 hours. On the other hand, sending faster requests to Google required the solution of CAPTCHAs.

For Google, we observed that the search request rate affected the number of search requests allowed during a specific time period before a CAPTCHA was shown. When requests were sent more often than one every sixteen seconds, Google showed a CAPTCHA after about 180 requests. When the request rate was increased to one request every 0.2 to two seconds, Google showed a CAPTCHA after about 75 requests. However, when requests were sent faster than one every 0.2 seconds, Google surprisingly allowed up to 1,000 search requests until a CAPTCHA was shown. By adding an automatic CAPTCHA-solving service, which typically requires less than 20 seconds per CAPTCHA, we could theoretically increase the throughput of Google searches to about ten requests per second. However, Google eventually started to ban our IP addresses, so we did not examine this issue any further.

Figure 9 shows the number of Google search requests (with error bars) before a CAPTCHA was shown in relation to the duration between search requests.

6. Use Case Scenario

A good use case scenario for our framework is a forensic investigation where seized storage devices have to be analyzed. After creating forensically-sound images of the seized devices, an investigator

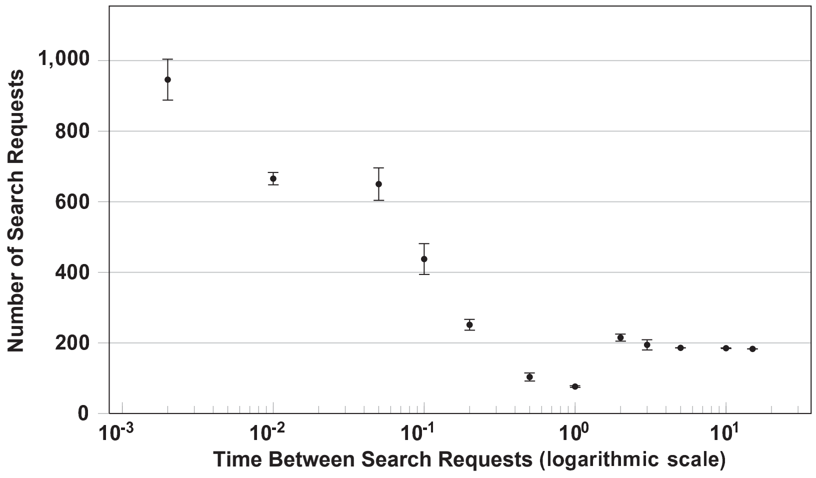


Figure 9. Google search requests.

would typically use a standard forensic tool to perform filesystem analysis. Next, data recovery would target the unallocated space to find previously-deleted files that have not been overwritten. The original filenames of recovered files are often missing, especially when a file carver has been used (e.g., because filesystem information may no longer be available). In the worst (but not atypical) case, the forensic investigator would end up with a large amount of files with non-descriptive filenames and almost no information about their content.

The recovered data must then be reviewed in order to find incriminating and exonerating material. Filtering techniques using whitelists and blacklists of cryptographic hashes would then be used to reduce the amount of data to be reviewed. This is where our framework can significantly support the forensic investigator – instead of only filtering based on incomplete hash databases, the investigator can employ our framework and feed all the data to be reviewed into it.

Our framework would automatically start searching multiple sources for information about the data. Whitelisted files such as system files and common application files would be automatically filtered. Blacklisted files would be marked as such and would be presented to the investigator together with aggregated and ranked information about files that are neither whitelisted nor blacklisted, but for which information such as a common filename has been found by searching P2P file sharing networks or using search engines.

By going through the ranked list of search results that contain, for example, content-describing filenames, a forensic investigator can find valuable information about the files of interest and their content without having to do a manual review. The investigator can then identify the files that should be eliminated due to irrelevant content and the files to be reviewed first because the results indicate that they have incriminating material. Searching for file hashes in distributed systems with large amounts of dynamically-changing content can greatly complement static filtering techniques based on whitelists and blacklists.

7. Conclusions

Distributed systems such as P2P networks and search engines can be used to gather information about files based on their file hashes. The framework created to automate file content identification is very effective at searching these distributed systems and aggregating, ranking and presenting the search results. This could greatly assist forensic investigators in gaining knowledge about file contents without conducting manual reviews.

The experimental results demonstrate that searching eDonkey networks is very effective for multimedia data such as video files, audio files and pictures. Google and the NSRL are also well suited to hash-based file identification. However, Yahoo! and the ISC and MHR hash databases did not produce any unique search results, and may be eliminated from consideration in favor of eDonkey, Google and the NSRL.

The hash databases were the fastest to search, followed by eDonkey's kad network and then Google (albeit with an automatic CAPTCHA solver). The experiments also demonstrated that MD5 and SHA-1 are still the most commonly used file hashes and are, therefore, well suited to hash-based file searches. Other hashes, such as SHA-2, are rarely used and may be eliminated from consideration at this time.

Our future research will identify other distributed systems that are suitable for hash-based file searches (e.g., P2P file sharing networks like DirectConnect). Also, databases providing specific multimedia content, such as Flickr [20], Grooveshark [7] and IMDb [8], will also be investigated. Future improvements to the framework include creating a client-server model for parallelizing search tasks and leveraging a cloud service infrastructure for faster distributed file hash searches.

Acknowledgement

This research was supported by the Center for Advanced Security Research Darmstadt (CASED), Darmstadt, Germany.

References

[1] F. Adelstein and R. Joyce, File Marshal: Automatic extraction of peer-to-peer data, *Digital Investigation*, vol. 4(S), pp. S43–S48, 2007.

[2] BitTorrent, BitTorrent and μTorrent software surpass 150 million user milestone; announce new consumer electronics partnerships, Press Release, San Francisco, California (www.bittorrent.com/intl/es/company/about/ces_2012_150m_users), January 9, 2012.

[3] Cisco Systems, Cisco Visual Networking Index: Forecast and Methodology, White Paper, San Jose, California (www.cisco .com/en/US/solutions/collateral/ns341/ ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf), 2012.

[4] B. Cohen, Incentives build robustness in BitTorrent, *Proceedings of the First International Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[5] Dev-Host, The ultimate free file hosting/file sharing service, Los Angeles, California (d-h.st).

[6] eMule-MODs.de, Server List for eDonkey and eMule (www.emule-mods.de/?servermet=show).

[7] Escape Media Group, Grooveshark, Gainesville, Florida (www.grooveshark.com).

[8] IMDb.com, Internet Movie Database, Seattle, Washington (www.imdb.com).

[9] Kuiper Forensics, PeerLab – Scanning and evaluation of P2P applications, Mainz, Germany (www.kuiper.de/index.php/en/peerlab).

[10] Y. Kulbak and D. Bickson, The eMule Protocol Specification, Technical Report, School of Computer Science and Engineering, Hebrew University of Jerusalem, Jerusalem, Israel, 2005.

[11] P. Maymounkov and D. Mazieres, Kademlia: A peer-to-peer information system based on the XOR metric, *Proceedings of the First International Workshop on Peer-to-Peer Systems*, pp. 53–65, 2002.

[12] National Institute of Standards and Technology, National Software Reference Library, Gaithersburg, Maryland (www.nsrl.nist.gov).

[13] Net Applications, Desktop Search Engine Market Share (www.netmarketshare.com/ search-engine-market-share.aspx? qprd=4&qpcustomd=0), October 2012.

[14] SANS Internet Storm Center, Hash Database, SANS Institute, Bethesda, Maryland (`isc.sans.edu/tools/hashsearch.html`).

[15] H. Schulze and K. Mochalski, Internet Study 2008/2009, ipoque, Leipzig, Germany (`www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf`), 2009.

[16] M. Steinebach, H. Liu and Y. Yannikos, Forbild: Efficient robust image hashing, *Proceedings of the SPIE Conference on Media Watermarking, Security and Forensics*, vol. 8303, 2012.

[17] M. Steiner, T. En-Najjary and E. Biersack, A global view of kad, *Proceedings of the Seventh ACM SIGCOMM Conference on Internet Measurement*, pp. 117–122, 2007.

[18] Team Cymru, Malware Hash Registry (MHR), Lake Mary, Florida (`www.team-cymru.org/Services/MHR`).

[19] VirusTotal Team, VirusTotal, Malaga, Spain (`www.virustotal.com`).

[20] Yahoo! Flickr, Sunnyvale, California (`www.flickr.com`).