Using Approximate Matching to Reduce the Volume of Digital Data

Frank Breitinger, Christian Winter, York Yannikos, Tobias Fink, Michael Seefried

Chapter 11

USING APPROXIMATE MATCHING TO REDUCE THE VOLUME OF DIGITAL DATA

Frank Breitinger, Christian Winter, York Yannikos, Tobias Fink and Michael Seefried

Abstract Digital forensic investigators frequently have to search for relevant files in massive digital corpora – a task often compared to finding a needle in a haystack. To address this challenge, investigators typically apply cryptographic hash functions to identify known files. However, cryptographic hashing only allows the detection of files that exactly match the known file hash values or fingerprints. This paper demonstrates the benefits of using approximate matching to locate relevant files. The experiments described in this paper used three test images of Windows XP, Windows 7 and Ubuntu 12.04 systems to evaluate fingerprint-based comparisons. The results reveal that approximate matching can improve file identification – in one case, increasing the identification rate from 1.82% to 23.76%.

Keywords: File identification, approximate matching, `ssdeep`

1. Introduction

Traditional physical media such as books, photos, letters and long-playing records (LPs) have been replaced by digital media in the form of e-books, digital photos, email and MP3 files. The result is that forensic investigators are faced with overwhelming amounts of digital data even in routine cases. Investigators need automated processes that can quickly classify and filter terabytes of data, yielding only the relevant files that can then be inspected manually.

A common approach is to employ cryptographic hash functions to automatically classify files: an investigator computes the hash values of all the files residing on digital media and compares these fingerprints

against file hash values stored in a reference database. A matching hash confirms that the referenced file is present in the storage media. Depending on the hash values stored in the reference database, the files can be classified as relevant (blacklisting) and non-relevant (whitelisting). Files not found in the database are not classified.

White [18] and Baier and Dichtelmuller [1] have analyzed the impact of using cryptographic hash functions for file classification in digital forensic investigations. White obtained identification rates as high as 70%. However, Baier and Dichtelmuller measured identification rates between 15% and 52%; these lower identification rates were due to small changes in files that typically occur during computer use.

As a consequence, the digital forensic research community has proposed the use of approximate matching that maps similar files to similar hash values. This paper shows that approximate matching can increase the file identification rate. In one case, the identification rate increased from 1.82% with traditional cryptographic hash value comparisons to 23.76% with approximate matching.

The principal contribution of this paper is the quantitative evaluation of identification rates for approximate matching on complete disk images. While reference results are available for cryptographic hashing [1, 18], approximate matching has not been evaluated on such a large scale. Also, the paper analyzes the error rates for approximate matching and establishes similarity score thresholds for use in real-world digital forensic investigations.

2. Background

Two strategies exist for filtering known content: (i) blacklisting, which compares a file to a reference database containing the fingerprints of illegal and suspicious files (e.g., child pornography) that should be retained for further analysis; and (ii) whitelisting, which compares a file to a reference database containing fingerprints of innocuous files (e.g., operating system files) that should be eliminated from consideration.

The most prominent reference database is the National Software Reference Library (NSRL) [11] maintained by the U.S. National Institute of Standards and Technology (NIST). NIST regularly publishes reference data sets containing the cryptographic hashes of software products. The current reference data set, RDS 2.42 of September 2013, covers approximately 115 million files.

Known file filtering predominantly utilizes cryptographic hash functions. However, the security properties of cryptographic hash functions imply that only identical files can be matched using these hash func-

tions – a difference in just one bit produces a completely different hash value. Although this property is desired for cryptographic purposes, it complicates forensic investigations. For example, an investigator is often interested in locating similar files, file fragments and embedded objects. Therefore, it is helpful to have algorithms that provide approximate matches to correlate related versions of data objects.

Approximate matching techniques find matches based on bytewise similarity or semantic similarity. Bytewise matching operates at the byte level: two inputs are similar if they have similar byte structures. Semantic matching, also called perceptual hashing or robust hashing, attempts to understand the input format and is, therefore, bound to specific media types, images or movies. While a semantic hashing solution is highly domain specific, bytewise approximate matching has the advantage of generality.

This paper investigates bytewise approximate matching, in particular, using the `ssdeep` tool [9, 10]. This was motivated by the prominence of `ssdeep`, the fact that NIST has published a reference set of `ssdeep` hashes, and the ability to process large file volumes in reasonable time using the F2S2 tool [19].

It is important to note that the concept of approximate matching should not be confused with locality sensitive hashing. Approximate matching reduces large files to small digests such that similar files are mapped to similar digests. On the other hand, locality sensitive hashing is an indexing strategy, which places items into hash buckets such that similar items have a high probability of being in the same bucket. Hence, locality sensitive hashing could speed up approximate matching depending on the matching approach and the locality sensitive hashing method that are employed.

2.1 ssdeep

Context triggered piecewise hashing was proposed by Kornblum [9, 10] and implemented in the `ssdeep` tool. It originates from the spam detection algorithm of Tridgell [17] implemented in `spamsum`.

The `ssdeep` tool divides a byte sequence (file) into chunks and hashes each chunk separately using the Fowler-Noll-Vo (FNV) algorithm [12]. Context triggered piecewise hashing then encodes the six least significant bits of each FNV hash as a Base64 character. All the characters are concatenated to create the file fingerprint.

The trigger points for splitting a file into chunks are determined by a rolling hash function. This function, which is a variation of the Adler-32 algorithm [8], is computed over a seven-byte sliding window to generate

a sequence of pseudorandom numbers. A number r in the sequence triggers a chunk boundary if $r \equiv -1 \pmod{b}$. The modulus b, called the block size, correlates with the file size.

Kornblum suggests dividing a file into approximately $S = 64$ chunks and using the same modulus b for similar-sized files. The modulus b is a saltus function:

$$b = b_{\min} \cdot 2^{\lfloor \log_2(N/S/b_{\min}) \rfloor} \tag{1}$$

where $b_{min} = 3$ and N is the input length in bytes. Since two fingerprints can only be compared if they were generated using blocks of the same size, `ssdeep` calculates two fingerprints for each file using the block sizes b and $2b$ and stores both fingerprints in one `ssdeep` hash.

The similarity of two fingerprints is calculated in two steps. First, the fingerprints are treated as text strings and compared with each other using an edit distance function. An edit distance of zero indicates matching strings and the distance increases with the dissimilarity of strings. In the second step, the computed distance of the two fingerprints is converted into a similarity score in the range 0 to 100, where a higher score indicates greater similarity.

The `ssdeep` similarity measure defines a score boundary for small fingerprints. Hence, small files that are identical often do not receive a score of 100, although this is expected. The score boundary is enforced in lines 600 through 604 of `fuzzy.c` (`ssdeep` version 2.10). Tridgell's comment in the source code says that this is done so as not to "exaggerate the match size" for a small block size. However, we eliminated this constraint in our research because the score boundary introduces uncertainty when interpreting the match results.

While `ssdeep` is a pioneer in the domain of approximate matching, several improvements to `ssdeep` hashing have been proposed in the literature [7, 16].

2.2 F2S2

The task of comparing files to a reference list can be very time consuming. In particular, the comparison of similarity digests cannot be performed as efficiently as the exact matching of cryptographic hashes. Approximate matching tools – including `ssdeep` – typically compare every hash computed for the investigation target to every hash in the reference list.

Unlike exact matching, approximate matching cannot be accelerated using classical indexing strategies developed for relational databases. However, it is possible to find suitable indexing strategies. The F2S2 tool [19] implements one such solution that involves n-gram indexing.

F2S2 has been applied to `ssdeep` hashes and has achieved a speedup of 2,000 times in realistic test cases.

3. Experimental Methodology

The experiments were performed on an Ubuntu 12.04 host using `ssdeep` 2.7-1. The drive images came from computers running the Windows and Linux operating systems. Since Windows is the most commonly used operating system, the experiments analyzed two Windows versions: Windows XP (Professional, 32 bit) and Windows 7 (Professional, 32 bit). The experiments also analyzed the Linux Ubuntu operating system (version 12.4, 32 bit).

3.1 Overview

The experiments focused on determining the appropriateness of SHA-1 and `ssdeep` for known file identification with approximate matching. The detection rates with respect to true positives and false positives for cryptographic hashing (SHA-1) and approximate matching (`ssdeep`) were examined. Note that the assessment only considered positive errors.

Some large files, such as `hyberfil.sys` (Windows XP), device files and named pipes (Ubuntu), did not generate `ssdeep` hashes. Since the algorithm was considered to be a blackbox, these problems were treated as bugs and were not investigated further.

Hash values could not be generated for a very small number of files. When a file hash value could not be generated, the file was treated as if it did not exist. In the worst case (Ubuntu_U), only 0.12% of the files had to be dropped.

3.2 System Description and Snapshots

The Windows XP samples came from two snapshots of an extensively-used system, which were 14 and 27 months old, respectively, when the snapshots were taken (the age of a system is defined as the time elapsed between system installation and snapshot generation). The Windows XP system was used on a daily basis for office work; it contained software installations, system and software update artifacts, user created files and other software usage traces. The Ubuntu system was about six months old when the snapshot was taken. This snapshot included system updates, and system development and web browsing artifacts. The Windows 7 system was used for half a day; during that time, the latest updates were applied, some files were created and some web pages were visited. The trusted reference snapshots were created from default

Table 1. Operating system snapshots.

Operating System	File Count	Disk Usage
WinXP_D (Windows XP, Default installation)	8,946	1.9 GB
WinXP_{U1} (Windows XP SP3, 14 months old)	195,186	128.4 GB
WinXP_{U2} (Windows XP SP3, 27 months old)	466,266	109.5 GB
Win7_D (Windows 7, Default installation)	45,470	8.1 GB
Win7_U (Windows 7 SP1, Used installation)	66,312	9.4 GB
Ubuntu_D (v12.04, Default installation)	185,468	3.3 GB
Ubuntu_U (v12.04, Used installation)	411,209	25.2 GB

installations of the operating systems (i.e., without any updates, service packs or additional programs).

Table 1 provides information about the operating system snapshots. Note that WinXP_{U1} used more disk space than WinXP_{U2} because an 82 GB image file was deleted between the times that the two snapshots were taken.

3.3 Reference Databases

The following reference databases were used to determine the identification rates for each operating system image:

- **NIST Database:** This NIST-published database (RDS 2.27 of December 2009) contains more than eight million entries. The hashes are provided in the form of a text file with one SHA-1 hash and one `ssdeep` hash per line. The SHA-1 hash is used to link each entry to the regular NIST RDS. The experiments used the SHA-1 and `ssdeep` hashes from the NIST database.

- **Custom Reference Databases:** Because the NIST database above is an outdated subset of the current RDS 2.42 (September 2013) and no newer `ssdeep` dataset was available, we created our own reference sets based on the default installations of the operating systems. The procedure used to create the reference sets was essentially the same as that used to create the NIST database.

3.4 Quality of Matches

The matches reported by `ssdeep` correspond to the number of positive collisions. The collisions require truth data in order to be classified as true positives and false positives. Because there were too many files to perform a manual inspection, the true positives were considered to be the files that matched the SHA-1 hashes.

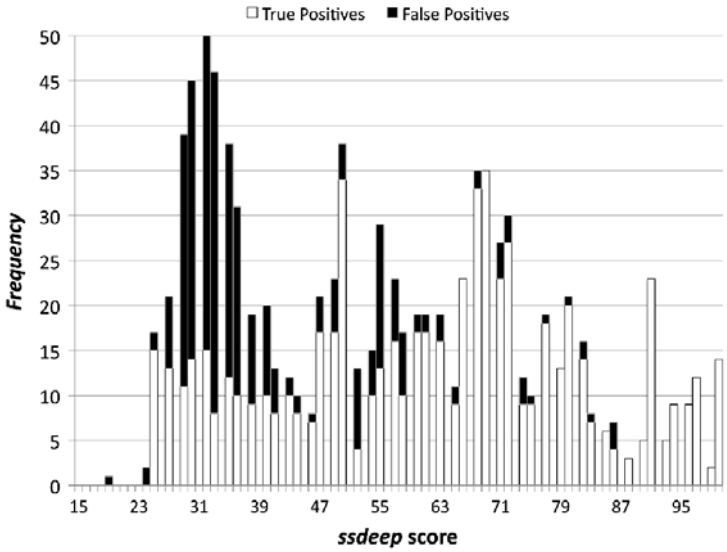

Figure 1. Distribution of `ssdeep` scores according to Roussev [14].

To perform approximate matching, a threshold t was set to yield a match score M such that $M \geq t$ indicates a positive match (i.e., similar files). However, identifying an appropriate threshold t was challenging because a low t value increases the false positive rate while a high t value decreases the true positive rate.

Roussev [14] has studied the ratio between true and false positives based on the comparison of 4,457 files (= 9,930,196 pairs). His results are presented in Figure 1. Accordingly, we set the thresholds to $t = 60$ and $t = 40$. Although the $t = 60$ threshold yielded a few false positives, we rated the false positive rate of the $t = 40$ threshold as acceptable.

4. Experimental Results

This section presents the experimental results. First, the difference between SHA-1 matches and `ssdeep` matches is examined. Next, identification rates are compared for the various images and databases. Following this, the identification rates are presented in correlation with file names and paths. Finally, the relationship between identification rate and file type is clarified.

4.1 Seemingly Identical Files

Minor differences exist between a SHA-1 match and an `ssdeep` score of 100. A SHA-1 match implies that two files are identical with extremely

Table 2. TC1: Comparing default installations against the NIST database.

D	|D|	$\mathbf{I}_{\text{ssdeep}}$ t = 40	$\mathbf{I}_{\text{ssdeep}}$ t = 60	$\mathbf{I}_{\text{ssdeep}}$ t = 100	$\mathbf{I}_{\text{SHA-1}}$
WinXP_D	8,946	68.69 %	63.85 %	35.84 %	35.24 %
Win7_D	45,470	16.59 %	9.32 %	1.73 %	1.70 %
Ubuntu_D	185,468	16.14 %	9.54 %	1.90 %	1.95 %

high certainty. In contrast, an `ssdeep` score of 100 does not necessarily mean that the two files are identical. Additionally, identical `ssdeep` hashes may not receive a score of 100 despite the fact that the small file score boundary was removed.

The score matching problem is due to the similarity digest comparison method used by `ssdeep`. The method requires that the two digests being compared have a common substring of length seven, otherwise the score is simply set to zero [3]. Note also that `ssdeep` assigns a score of zero to two empty files whereas SHA-1 matches the two files.

An `ssdeep` score of 100 can be obtained even when the SHA-1 hashes are different; this is due to fingerprint collisions and the comparison method. Note that each `ssdeep` digest consists of two fingerprints and it is sufficient to have one matching pair to obtain a score of 100. Thus, there are two types of collisions for `ssdeep`. In some cases, the `ssdeep` hashes match completely; in other cases, only one pair of `ssdeep` fingerprints matches. We consider a similarity score 100 for two non-identical files to be a false positive.

4.2 Detection Thresholds

We also analyzed the detection rates based on the thresholds $t = 40$ and $t = 60$. When presenting the results, note that D denotes the analyzed system and $|D|$ denotes the number of files in the system. Also, I_A is the identification rate using algorithm $A \in \{\texttt{ssdeep}, \text{SHA-1}\}$. For example, $I_{\text{ssdeep}} = 10\,\%$ means that 10 % of the files in system D were found in the database using `ssdeep`. Thus, the higher the value of I, the greater the number of files that are identified automatically.

- **Test Case 1 (TC1):** This test case compared the files in the default operating system installations with the files represented in the NIST database. The results are shown in Table 2. The identification rates for the Windows XP system are significantly higher compared with those for the other operating systems; this because the underlying database was created in December 2009

Table 3. TC2: Comparing used installations against the NIST database.

D	**|D|**	$\mathbf{I}_{\text{ssdeep}}$ **t = 40**	$\mathbf{I}_{\text{ssdeep}}$ **t = 60**	$\mathbf{I}_{\text{ssdeep}}$ **t = 100**	$\mathbf{I}_{\text{SHA-1}}$
WinXP_{U1}	195,186	17.79 %	14.70 %	7.69 %	8.03 %
WinXP_{U2}	466,266	23.02 %	17.39 %	7.05 %	7.30 %
Win7_{U}	66,312	17.88 %	10.21 %	1.45 %	1.44 %
Ubuntu_{U}	411,209	23.76 %	17.11 %	1.79 %	1.82 %

when Windows XP was very popular. Still, the trend is obvious, regardless of the operating system, the identification rate is much higher for `ssdeep` (e.g., nearly ten times better for Win7 with the threshold $t = 40$).

- **Test Case 2 (TC2):** This test case compared the files in the used operating system installations with the files represented in the NIST database. The results in Table 3 are comparable with those obtained in Test Case 1 (Figure 2). Once again, the Windows XP systems have the best identification rates due to the underlying database. However, the identification rates are smaller than in the previous test case because the systems contained large numbers of files. For example, WinXP_{U2} had in excess of 50 times more files than WinXP_{D}. Still, the identification rates are approximately two to ten times higher for `ssdeep` with threshold $t = 60$ compared with SHA-1.

Table 4. TC3: Comparing used installations against default installations.

D	**|D|**	$\mathbf{I}_{\text{ssdeep}}$ **t = 40**	$\mathbf{I}_{\text{ssdeep}}$ **t = 60**	$\mathbf{I}_{\text{ssdeep}}$ **t = 100**	$\mathbf{I}_{\text{SHA-1}}$
WinXP_{U1}	195,186	5.12 %	4.71 %	4.13 %	4.14 %
WinXP_{U2}	466,266	2.27 %	2.01 %	1.72 %	1.74 %
Win7_{U}	66,312	93.85 %	90.19 %	67.41 %	67.72 %
Ubuntu_{U}	411,209	55.67 %	53.38 %	47.74 %	47.83 %

- **Test Case 3 (TC3):** This test case compared the files in the used installations against those in the default installations. The main results are shown in Table 4. The low rates for the Windows XP systems are a consequence of the "small" default installation. Recall that the default installation had only about 9,000 files. How-

Table 5. TC4: Comparing WinXP_{U2} against WinXP_{U1} as the reference database.

D	**|D|**	$\mathbf{I}_{\text{ssdeep}}$ **t = 40**	$\mathbf{I}_{\text{ssdeep}}$ **t = 60**	$\mathbf{I}_{\text{ssdeep}}$ **t = 100**	$\mathbf{I}_{\text{SHA-1}}$
WinXP_{U2}	466,266	31.98 %	28.87 %	24.04 %	24.37 %

ever, some files in the default installation are similar or identical to files in the used installation. Examples are the `desktop.ini` files, DLLs and file shortcuts (`.lnk`). Typical locations for such files in the default installation are `WINDOWS/system32/config/systemprofile`, `WINDOWS/pchealth/helpctr/System` and `WINDOWS/system32/dllcache`. Hence, it was possible to identify more files in the used system than were present in the default system.

The high identification rates for the Windows 7 system reflect the fact that the system was not used very much. Also, the updates introduced many files that are similar to the files found in the default system.

- **Test Case 4 (TC4):** This test case compared the files in the two Windows XP snapshots where WinXP_{U1} emulates the database. The main results are shown in Table 5. Despite the difference of 13 months between the two snapshots, identification rates are 24% for SHA-1 and 32% for `ssdeep`, a difference of 8%. The percentage may seem small, but 8% of the 466,266 files in the system corresponds to approximately 37,000 files.

The experiments confirmed that the detection rates for `ssdeep` are higher than for SHA-1 in all four test cases. This is especially relevant for blacklisting (files that are similar to suspicious files could constitute evidence). Also, as shown in the comprehensive study by Roussev [14], the thresholds used in the experiments are reasonable and have acceptable false positive rates.

4.3 File Names and Paths

This section analyzes the reliability of positive matches obtained using the two thresholds. Recall that the number of approximate matches is the total number of `ssdeep` matches minus the number of exact SHA-1 matches. Table 6 shows the results. The first column specifies the comparisons, e.g., WinXP U1 versus WinXP U2. The second and third columns list the numbers of files in the systems ($|D|$) and the numbers of matches with scores greater than or equal to t (`ssdeep` Hits), respec-

Table 6. Identification rates for file paths and names in different images.

D	|D|	ssdeep Hits	Without SHA-1	Path Matches	Name Matches
WinXP($U1$ v. $U2$); t=40	466,266	31.98 %	7.97 %	10.57 %	37.09 %
WinXP($U1$ v. $U2$); t=60	466,266	28.87 %	4.86 %	15.58 %	50.68 %
Win7(B v.U); t=40	66,312	93.85 %	26.44 %	1.51 %	5.79 %
Win7(B v. U); t=60	66,312	90.19 %	22.78 %	1.51 %	5.91 %
Ubuntu(B v. U); t=40	411,209	55.67 %	7.99 %	16.99 %	71.37 %
Ubuntu(B v. U); t=60	411,209	53.38 %	5.71 %	22.91 %	85.89 %

tively. The fourth column shows the benefit of `ssdeep`, i.e., matches that are not identified by SHA-1 (relative to $|D|$) – this is the "critical amount" of files. The last two columns list the file path and name matches relative to the numbers of files excluding the SHA-1 matches.

Consider, for example, the last row of Figure 6, which examines the default Ubuntu installation against the used Ubuntu installation that contains 411,209 files. When $t = 60$, `ssdeep` detects 53.38% as similar files. Reducing this figure by the number of SHA-1 matches yields 5.71% (i.e., 23,480 files). 22.91% of these files have the same path and 85.89% have the same file name. Hence, these files are considered to be true positives. The remaining $100\% - 85.89\% = 14.11\%$ files are either false positives or files that have been moved or renamed. A total of 3,313 files remain and these files need to be analyzed manually.

4.4 File Types

While approximate matching at the syntactic level can be applied to any file, it is not useful for all file types. This depends on the file type and the type of modification – whether small modifications preserve most of the binary content of a file or lead to a completely different binary pattern. For example, text is favorable while compressed data causes problems. The reason is that small text modifications result in small changes to the binary data of a text file, but compression algorithms create very different byte sequences given similar inputs. Thus, approximate matching at the raw byte level cannot discern the real similarity when compression algorithms have been used.

The final experiment sought to identify the file types for which `ssdeep` is better than cryptographic hashing. The test considered only `ssdeep` scores between 60 and 99. Table 7 shows the ten most frequent file types with scores between 60 and 99 for each operating system based on Test Case 2. The numbers correspond to the file percentages compared with

Table 7. File types with high identification rates for non-identical files.

$\mathbf{WinXP}_{U2}$		$\mathbf{Win7}_{U}$		$\mathbf{Ubuntu}_{U}$	
Type	**Amount**	**Type**	**Amount**	**Type**	**Amount**
	11.60 %	.mum*	40.05 %	.html*	60.62 %
.html*	10.44 %	.inf*	10.09 %	.h*	19.40 %
.h*	6.40 %	.dll	8.18 %		10.63 %
.yaml*	6.09 %	.png	6.06 %	.pm*	1.42 %
.svn-base*	4.52 %	.mui	4.77 %	.gz	0.73 %
.dll	4.15 %	.gpd*	4.42 %	.png	0.63 %
.png	3.82 %	.fon	3.37 %	.py*	0.63 %
.py*	2.53 %	.nls	3.30 %	.al	0.55 %
.mf*	1.86 %	.ttf	2.48 %	.ent	0.44 %
.htm*	1.69 %	.ini*	1.32 %	.ps	0.36 %

all the files identified within the score range. The file types that include text are marked with an asterisk; they constitute the majority of the listed files. However, binary file types (e.g., `.dll` in Table 7) also have high identification rates. Other results (not shown in this paper) reveal that `.exe`, `.pyc` and `.so` are also among the top ten file types. Note that files without suffixes are typically text files or binary executables.

5. Related Work

Approximate matching has been shown to be useful for detecting similar inputs (e.g., different versions of a file), detecting embedded objects (e.g., a `.jpg` file in a Word document) and detecting fragments (e.g., network packets) [14]. Apart from `ssdeep`, we are aware of six other approximate matching algorithms. Two of them have similar qualities as `ssdeep`. The first is `sdhash` [13], which identifies statistically improbable features (e.g., a byte sequence of 64 characters), hashes them using SHA-1 and sets five bits in a Bloom filter. The second, `mrsh-v2` [5], is a combination of `ssdeep` and `sdhash`. This algorithm uses a rolling hash to divide the input into chunks and each chunk is hashed using the FNV algorithm and inserted in a Bloom filter.

The remaining algorithms are less practical. For example, `bbHash` [4] is very slow – it requires about two minutes to process a 10 MiB file. `mvHash-B` [2] needs a specific configuration for each file type, while `SimHash` [15] and `MinHash` [6] can only handle near duplicates.

White [18] has analyzed the benefits of using hash functions in digital forensics. Instead of hashing complete files, block hashing applies cryptographic algorithms to smaller-than-filesize portions of data, where a portion is a 4,096-byte block. According to White, file-based data re-

duction leaves an average of 30% of disk space for human investigation and incorporating block hashes reduces the amount of data for human review to 15% of disk space. However, White focused on basic installations, which is not a realistic scenario. Baier and Dichtelmuller [1] showed that the reduction rates for used workstations were much worse than those obtained by White. In particular, Baier and Dichtelmuller obtained rates of approximately 50% for a basic Windows XP installation and rates as low as 15% for used Windows 7 and Ubuntu systems. In both cases, Baier and Dichtelmuller used the latest reference data set that contained many more entries than the reduced NIST database employed in this research.

6. Conclusions

This paper describes the first experimental evaluation of approximate matching with large test cases. The results show that approximate matching provides substantial benefits compared with cryptographic hashing. Approximate matching significantly increases the number of files identified as known files. Also, it reduces the number of files that have to be inspected manually by digital forensic investigators. Text files and binary files can be filtered effectively using approximate matching at the syntactic level. However, it is important that an up-to-date reference database is used to obtain the best identification rates.

Better results than `ssdeep` can be obtained using a more accurate approximate matching algorithm such as `sdhash`. However, an efficient similarity comparison method must be devised before `sdhash` can be evaluated using large test cases.

Acknowledgement

This research was partially supported by the European Union under the Integrated Project FIDELITY (Grant No. 284862) and by the Center for Advanced Security Research Darmstadt (CASED).

References

[1] H. Baier and C. Dichtelmuller, Datenreduktion mittels kryptographischer Hashfunktionen in der IT-Forensik: Nur ein Mythos? *DACH Security*, pp. 278–287, September 2012.

[2] F. Breitinger, K. Astebol, H. Baier and C. Busch, `mvhash-b` – A new approach for similarity preserving hashing, *Proceedings of the Seventh International Conference on IT Security Incident Management and IT Forensics*, pp. 33–44, 2013.

[3] F. Breitinger and H. Baier, Security aspects of piecewise hashing in computer forensics, *Proceedings of the Sixth International Conference on IT Security Incident Management and IT Forensics*, pp. 21–36, 2011.

[4] F. Breitinger and H. Baier, A fuzzy hashing approach based on random sequences and Hamming distance, *Proceedings of the Conference on Digital Forensics, Security and Law*, 2012.

[5] F. Breitinger and H. Baier, Similarity preserving hashing: Eligible properties and a new algorithm `mrsh-v2`, *Proceedings of the Fourth International ICST Conference on Digital Forensics and Cyber Crime*, 2012.

[6] A. Broder, On the resemblance and containment of documents, *Proceedings of the International Conference on the Compression and Complexity of Sequences*, pp. 21–29, 1997.

[7] L. Chen and G. Wang, An efficient piecewise hashing method for computer forensics, *Proceedings of the First International Workshop on Knowledge Discovery and Data Mining*, pp. 635–638, 2008.

[8] P. Deutsch and J. Gailly, ZLIB Compressed Data Format Specification Version 3.3, RFC 1950, 1996.

[9] J. Kornblum, Identifying almost identical files using context triggered piecewise hashing, *Digital Investigation*, vol. 3(S), pp. S91–S97, 2006.

[10] J. Kornblum, `ssdeep` (`ssdeep.sourceforge.net`), 2013.

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

[12] L. Noll, FNV hash (`www.isthe.com/chongo/tech/comp/fnv/index.html`), 2013.

[13] V. Roussev, Data fingerprinting with similarity digests, in *Advances in Digital Forensics VI*, K. Chow and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 207–226, 2010.

[14] V. Roussev, An evaluation of forensic similarity hashes, *Digital Investigation*, vol. 8(S), pp. S34–S41, 2011.

[15] C. Sadowski and G. Levin, SimHash: Hash-Based Similarity Detection, Technical Report UCSC-SOE-11-07, Department of Computer Science, University of California Santa Cruz, Santa Cruz, California (`simhash.googlecode.com/svn/trunk/paper/SimHashWithBib.pdf`), 2007.

[16] K. Seo, K. Lim, J. Choi, K. Chang and S. Lee, Detecting similar files based on hash and statistical analysis for digital forensic investigations, *Proceedings of the Second International Conference on Computer Science and its Applications*, 2009.

[17] A. Tridgell, `spamsum` (`mirror.linux.org.au/linux.conf.au/2004/papers/junkcode/spamsum/README`), 2002.

[18] D. White, Hashing of file blocks: When exact matches are not useful, presented at the *Annual Meeting of the American Academy of Forensic Sciences*, 2008.

[19] C. Winter, M. Schneider and Y. Yannikos, F2S2: Fast forensic similarity search through indexing piecewise hash signatures, *Digital Investigation*, vol. 10(4), pp. 361–371, 2013.