DFRWS
DIGITAL FORENSIC RESEARCH CONFERENCE

On The Database Lookup Problem Of Approximate Matching

By

Frank Breitinger, Harald Baier and Douglas White

Presented At

The Digital Forensic Research Conference

DFRWS 2014 EU Amsterdam, NL (May 7th - 9th)

On the database lookup problem of approximate matching

Frank Breitinger, Harald Baier, Douglas White

Hochschule Darmstadt, CASED

DFRWS-EU, 2014-05-08

Harald Baier

1. Doctoral degree from TU Darmstadt in the area of elliptic curve cryptography.
2. Principal Investigator within Center for Advanced Security Research Darmstadt (CASED)
3. Establishment of forensic courses within Hochschule Darmstadt.
4. Current working fields:
   - Fuzzy Hashing (IT forensics, biometrics).
   - Anomaly detection in high-traffic environments.
   - Security protocols for eMRTD.

Motivation

Motivation

Foundations

Motivation

Foundations

Problem description and solution overview

Use Case: Prosecution

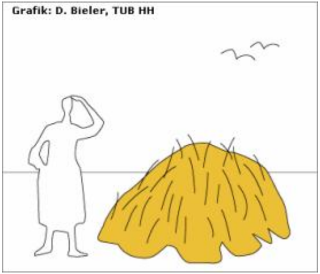
1. Police and prosecutors confronted with different storage media:
   - Hard disk drives, solid-state drives, USB sticks.
   - Mobile phones, SIM cards.
   - Digital cameras, digital camcorders, SD cards.
   - CDs, DVDs.
   - RAM (dumps).
   - ...
2. Amount of distrained data often exceeds 1 terabyte.

Different views of 1 terabyte

1 terabyte of digital text is (approximately) equal to:

1. 1 trillion characters: 1 character = 1 byte.
2. 220 million pages: 1 page = 5000 characters.
3. 21 years of printing time: 20 sheets per minute.
4. 1 million kg of paper: onesided printed.
5. Paper stack of 22 km height: bulk of 0.1 mm.

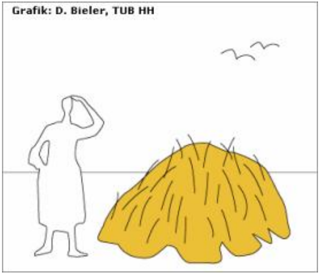
Finding relevant files resembles ...



Source: tu-harburg.de

Source: beepworld.de

Finding relevant files resembles ...



Source: tu-harburg.de

Source: beepworld.de

Key question:
How to minimize the haystack or enlarge the needle?

Foundations

Hash functions in digital forensics

1. Automatically identify known files:
   - Filter in: Highlight suspect files (e.g., company secrets)
   - Filter out: Remove non-relevant objects (e.g., OS files)
2. Proceeding:
   2.1 Hash the file,
   2.2 Compare the resulting hash against a database,
   2.3 and put it on one of the categories:
      - Known-to-be-good (non-relevant).
      - Known-to-be-bad (suspect).
      - Unknown files.
3. Goal:
   - Known files can be identified very efficiently.
   - Reduces amount of data investigator has to look at by hand.

Cryptographic hash functions in digital forensics

1. Identifying exact duplicates is solved using cryptographic hash functions:
   - Filter out: National Software Reference Library (NSRL).
   - Filter in: Perkeo database in Germany.
2. A sample drawback: avalanche effect.

```
$ echo ’Dear Angela, I give you 1 million EUR. Wolfgang’ | sha1sum
9bf13969f2c283cfe0ace585667fa22c7ab4f84a -

$ echo ’Dear Angela, I give you 1 billion EUR. Wolfgang’ | sha1sum
60d0b09f8d18e75b3cd7ffb0406de84bbc459510 -
```

Approximate matching

1. However, investigators need robust algorithms that allow similarity detection.
2. Sample use cases:
   - Different versions of files.
   - Embedded objects.
   - Fragments of files.
   - Network packets.

$\Longrightarrow$ Approximate matching (similarity hashing, fuzzy hashing).

Our notation

- x is the number of files in the database.
- Bloom filter is a bit array to represent data.
- m denotes the Bloom filter size in bits.
- $feature$ describes a byte sequence which is hashed and inserted into the Bloom filter.
- k number of sub-hashes where each one sets a bit in the Bloom filter.
- n is the number of features inserted into a Bloom filter.
- s denotes the file set size in MiB.
- S_B denotes the set of blacklisted files.
- S_D denotes the set of files on a seized device.

Motivation

Foundations

Problem description and solution overview

Experimental results and assessment

Conclusion and future work

NSRL-RDS

Cryptographic hash values can be sorted, e.g., RDS:

```
$ less NSRLFile.txt

"SHA-1","MD5","CRC32","FileName","FileSize","ProductCode","OpSystemCode","SpecialCode"
"000000206738748EDD92C4E3D2E823896700F849","392126E756571EBF112CB1C1CDEDF926","EBD105A0","I05002T2.PFB",9
"0000004DA6391F7F5D2F7FCCF36CEBDA60C6EA02","0E53C14A3E48D94FF596A2824307B492","AA6A7B16","00br2026.gif",2
"000000A9E47BD385A0A3685AA12C2DB6FD727A20","176308F27DD52890F013A3FD80F92E51","D749B562","femvo523.wav",4
"00000142988AFA836117B1B572FAE4713F200567","9B3702B0E788C6D62996392FE3C9786A","05E566DF","J0180794.JPG",3
"00000142988AFA836117B1B572FAE4713F200567","9B3702B0E788C6D62996392FE3C9786A","05E566DF","J0180794.JPG",3
"00000142988AFA836117B1B572FAE4713F200567","9B3702B0E788C6D62996392FE3C9786A","05E566DF","J0180794.JPG",3
"00000142988AFA836117B1B572FAE4713F200567","9B3702B0E788C6D62996392FE3C9786A","05E566DF","J0180794.JPG",3
"00000142988AFA836117B1B572FAE4713F200567","9B3702B0E788C6D62996392FE3C9786A","05E566DF","J0180794.JPG",3
"00000142988AFA836117B1B572FAE4713F200567","9B3702B0E788C6D62996392FE3C9786A","05E566DF","J0180794.JPG",3
```

$\Longrightarrow$ Efficient decision if a given hash value matches a hash of the RDS (in $O(\log(x))$ or $O(1)$ comparisons)

Indexing problem of similarity digests

1. Similarity digests cannot be indexed in general:
   - To decide if a given fuzzy hash is similar to one of the database requires $O(x)$ comparisons, i.e., against all.
   - Comparison complexity is $O(xy)$ if a set comprising y elements is compared to the database.
   - Too slow for practical usage.
2. Winter et al. presented a solution for `ssdeep` digests (a Base64 sequence) called F2S2.
3. No solution for *Bloom filter digests*:
   - `sdhash`.
   - `mrsh-v2`.
   - `mvhash-B`.

Solution overview

1. Overall idea: store all files in one single (huge) Bloom filter.
2. Bloom filter should fit to RAM for efficiency reasons.
3. Our setting aims at a ratio $1/100$, i.e., a 200 GiB set S_B requires ≈ 2 GiB Bloom filter.
4. Benefit:
   - Comparison complexity is $O(1)$.
5. Drawback:
   - File to set comparison yields a binary decision.
   - Result: *yes, file is in the set* vs. *no, it is not*.
   - Sufficient for Blacklisting?!

Solution alternatives

1. Bloom filter of S_B fits to RAM: Best case.
   - Bloom filter filled with the black listed files in advance.
   - Files of S_D compared against Bloom filter.
2. Bloom filter of S_B does not fit to RAM: Worst case.
   - Fill Bloom filter with files of S_D (if possible).
   - Black listed files from S_B are compared against Bloom filter of S_D (use precomputed hashes of S_B if possible).
   - Bloom filter of S_D cannot be computed in advance.

Some details

1. Match decision:
   - A fragment of a given file is assumed to be in the Bloom filter, if a sufficiently large number of subsequent features is found in the filter (longest run, lr).
   - Let r_{min} denote the minimum number of subsequent features for a match: $lr \geq r_{min}$.
   - Our prototype sets $r_{min} = 6$.
2. We aim at a fragment false positive rate of $p_f = 10^{-6}$.
3. Bloom filter size:

$$m = -\frac{k \cdot s \cdot 2^{14}}{\ln(1 - \sqrt[k r_{min}]{p_f})}$$

   - Approximately $1/100$ of the size of the input file set.

Our tool `mrsh-net`

1. Based on multi resolution hashing algorithm `mrsh-v2`.
2. Originally developed for network packet approximate matching.
3. Available via
   `http://www.dasec.h-da.de/staff/breitinger-frank/`
4. Result presentation:
   - Due to file to set comparison: no similarity score is computed.
   - Instead the following (sample) output is given:

     `file1.ppt: 163 of 2518 (longest run: 111)`
5. Parameters can be adjusted in the config file `config.h`.
6. The paper discuss all parameters and sample choices.

Experimental results and assessment

Test corpus

1. Real world files from the t5-corpus.
2. Available via `http://roussev.net/t5/`
3. Contains 4,457 files with a total size of 1.78 GiB.
4. The average file size is ≈ 400 KiB.
5. File type distribution:

jpg	gif	doc	xls	ppt	html	pdf	txt
362	67	533	250	368	1093	1073	711

Efficiency: Database size

	sdhash	mrsh-v2	mrsh-net worst	mrsh-net worst	mrsh-net best	F2S2	SHA-1
Database size	61.18 MiB	27.33 MiB	1.78 GiB	1.78 GiB	32 MiB	3.69 MiB	0.24 MiB

1. In case of `sdhash`, `mrsh-v2`, `F2S2` and `SHA-1` the database comprises the (similarity) hashes.
2. *Worst case* describes the scenario where the Bloom filter of S_B does not fit to RAM and hence is not used.
3. `mrsh-net` makes use of the default Bloom filter size of 32 MiB (sufficient for set size of S_B of ≈ 3 GiB).

Efficiency: Run time

	sdhash	mrsh-v2	mrsh-net worst	mrsh-net worst	mrsh-net best	F2S2	SHA-1
Hashing	178 s	53 s	123 s	77 s	77 s (123 s)	221 s	24 s
Comparing	1281 s	1259 s	< 1 s*	< 1 s*	< 1 s	< 1 s	< 1 s
Total	1459 s	1312 s	246 s	154 s	77 s (123 s)	221 s	24 s

1. 'Hashing' denotes the time to hash S_D, i.e., to hash all files of the t5-corpus.
2. `mrsh-net` 'worst'-columns: 2nd column is optimised for this dataset (more efficient feature hash function for 'small' datasets).
3. 'Comparing' is the time to compare all files of the t5-corpus against the hash database of S_B (if available).
4. 'Total' is the overall time (total $=$ comparing $+$ hashing).

Efficiency: Real world scenario

	sdhash	mrsh-v2	mrsh-net worst	mrsh-net best
Database size	49.79 GiB	22.22 GiB	1500 GiB	16 GiB
Hashing	329 min	98 min	227 min	227 min
Comparing	3.84 years	3.77 years	32.63 h	< 1 min

1. Assumptions:
   - Size of S_B: $1,500$ GiB.
   - Size of S_D: 200 GiB.
2. We assume a linear growth in both space and run time.
3. The efficiency advantage of `mrsh-net` is obvious.

Detection performance

1. Our prototype decides between match and non-match based on the longest run and thus on longest common substring (LCS).
2. Key issue: there is no labelled reference data set available.
3. We therefore use an approximation of the LCS (aLCS) as explained yesterday in the talk by Vassil Roussev.
   - Ground truth decision based on aLCS score:

```
file1 | file2 | aLCS  | entropy
a.dat | b.dat |   993 | 5.56
c.dat | d.dat | 11945 | 0.5
```

   - Note: aLCS is a lower bound on actual LCS.

Definition of classification result

1. Definition of true positive (TP), false positive (FP), true negative (TN) and false negative (FN) as follows:
   - TP: $mrsn(f, BF) \geq r_{min}$ and $aLCS(f, GT) \geq r_{min} \cdot bs$.
   - FP: $mrsn(f, BF) \geq r_{min}$ and $aLCS(f, GT) < r_{min} \cdot bs$.
   - TN: $mrsn(f, BF) < r_{min}$ and $aLCS(f, GT) < r_{min} \cdot bs$.
   - FN: $mrsn(f, BF) < r_{min}$ and $aLCS(f, GT) \geq r_{min} \cdot bs$.
2. Remark: $r_{min} \cdot bs = 6 \cdot 64 = 384$ bytes.

Detection performance: Results

1. Confusion matrix:

Actual situation \ Classified as	Positive	Negative
Positive	2537	436
Negative	18	1466

2. Results:

Precision: $\frac{TP}{TP+FP} = \frac{2537}{2555} = 99.3\%$

Recall: $\frac{TP}{TP+FN} = \frac{2537}{2973} = 85.3\%$

Accuracy: $\frac{TP+TN}{TP+FP+TN+FN} = \frac{4003}{4457} = 89.8\%$

Decrease false negatives

1. Having a closer look at the very high number of false negatives, we observe that most aLCS matches are based on low entropy sequences.

entropy	> 0	> 1	> 2	> 3
TN/(TN+FN)	78.5 %	82.3 %	86.4 %	91.2 %
FN/(TN+FN)	21.5 %	17.7 %	13.6 %	8.8 %

2. Future work will take entropy of LCS into account.

Conclusion and future work

Take home messages

1. It is important to have indexing strategies for similarity digests.
2. Otherwise they will not operate with practical speed.
3. We have presented and evaluated a new approach to efficiently decide about the similar membership of a file to a given dataset.
4. The lookup complexity decreased from $O(x)$ comparisons to $O(1)$ for one file.

Future work

1. Decrease the number of false negatives.
2. Perform a detection performance study in terms of ROC or DET curves.
3. Extend the algorithm to find the actual similar file.

Questions?

Source: `www.dilbert.com/strips/2011-02-03`