FRASH: A framework to test algorithms of similarity hashing

DFRWS'13 – F. Breitinger, G. Stivaktakis & H. Baier

Frank Breitinger

CASED

da/sec
BIOMETRICS AND INTERNET-SECURITY RESEARCH GROUP

- **Bachelor Degree at University of applied sciences Mannheim in March 2009.**

- **Master Degree at University of applied sciences Darmstadt in February 2011.**
  - Area of specialization: IT-Security.
  - Bytewise approximate matching.

- **PhD Research Student at CASED since March 2011.**

- **Current working topics:**
  - Testing, comparing and improving existing approaches.
  - Finishing my thesis.

Motivation

- Handling terabytes of data is a challenge in today's IT forensic investigation.
  - Needle in the Haystack.

How to minimize the haystack or *enlarge the needle?*

Towards a solution

- **Automatically identify files.**
  - Highlight suspect files (e.g, company secrets or child pornography) or
  - Remove non-relevant objects (e.g, OS files) from further investigation

- **Identifying exact duplicates is often solved using cryptographic hash functions.**
  - National Software Reference Library (NSRL).

- **However, it is also helpful to have more flexible and robust algorithms that allow *similarity detection.***
  - E.g., different versions of files.

→ Approximate matching (a.k.a. similarity hashing).

Problem

- **Establishing a new algorithm requires a thorough assessment by the community on base of well-known criteria.**
  - E.g., NIST governed the processes to standardize AES and SHA-3.

- **Approximate matching will only be accepted by both the scientific community and practitioners if an assessment methodology and a test framework are available.**

What do we expect from tools?

CASED

da/sec
BIOMETRICS AND INTERNET-SECURITY
RESEARCH GROUP

An approach should solve at least one of these "tasks":

- **Document similarity detection.**
  - Identify related documents, e.g., different versions of a Word document.
- **Embedded object detection.**
  - Identify a given object inside a container, e.g., a JPG within a Word document.
- **Fragment detection.**
  - Identify an original input based on a fragment, e.g., analyzing a device on the byte level or cropped pictures.
- **Clustering files.**
  - Group files that share similar content, e.g., a Word document and an e-mail.

CASED

da/sec
BIOMETRICS AND INTERNET-SECURITY
RESEARCH GROUP

Distinction of approaches

- **Semantic approximate matching.**
  - Uses contextual attributes of the digital object, and operates at a level close to human perception.
- **Syntactic approximate matching.**
  - Uses internal structures present in digital objects, e.g., byte structure of network packets.
- **Bytewise approximate matching.**
  - Matching relies only on the sequences of bits which make up a digital object.
    - Our focus in the following.

Tools for bytewise approximate matching

CASED

da/sec
BIOMETRICS AND INTERNET-SECURITY RESEARCH GROUP

Most prominent Tools:

- **ssdeep (Jesse Kornblum, 2006)**
  - Divide input in chunks based on the rolling hash. Concatenate chunk hashes to get a final similarity digest (fingerprint).

- **sdhash (Vassil Roussev, 2010)**
  - Extract statistically improbable features, hash them and put them into a Bloom filter which is the similarity digest.

Further approaches:

- bbHash, mvHash-B, mrsh-v2.

What should we test?

Efficiency [1/2]

- **Runtime efficiency (ease of computation).**
  - Fundamental properties of algorithms.
  - Due to large amount of data it is obvious that algorithms have to be fast.
  - Time that the algorithm needs to process the input (reading file from device and generating the similarity digest).
  - FRASH includes SHA-1 as a benchmark.

- **Compression.**
  - Traditional hash functions output a fixed length fingerprint, which is in contrast to approximate matching, where we often have a variable length.
  - Short fingerprints are desirable.
  - Compression measures the ratio between input and output.

$$compression = \frac{output\ length}{input\ length} \cdot 100$$

Efficiency [2/2]

- **Fingerprint comparison.**
  - An approach is only useful if it has a fast comparison function.
  - Time may vary due to different fingerprint length and comparison algorithms (e.g., Hamming distance of sdhash vs Levenshtein of ssdeep).
  - Fingerprint comparison measures the time of an all-against-all comparison of fingerprints (excludes the fingerprint generation).

Sensitivity & robustness [1/4]

- **Single-common-block correlation.**
  - Simulates a situation where two files have a single common object”. Considering two files f1 and f2 that are completely different, but share a common object O, “what is the smallest O for which the similarity tool reliably correlates the two targets?” (Roussev, 2011).

- **Test procedure:**
  - Create two random files f1 and f2 of size X ∈ {512 KB, 2048 KB, 8192 KB} and a common block O of size X/2.
  - O overwrites f1 and f2 at different and randomly chosen offsets.
  - If score > 0, reduce O by 16 KB and restart.
  - Test stops when match score = 0.

Sensitivity & robustness [2/4]

- **Fragment detection.**
  - Considering a file, what is the smallest piece/fragment, for which the similarity tool reliably correlates the fragment and the original file? Fragment detection identifies the minimum correlation between an input and a fragment.

- **Test procedure:**
  - Cut X% of the original input length and generates the match score.

    Default X = 5; max cuts: 100/X -1.
  - In case the algorithm still identifies similarity, FRASH does a further reduction in 1% steps until only 1% of the input is left.
  - Two different modes:
    - 1. Random cutting: randomly cut at the beginning or at the end.
    - 2. End side cutting: only cut blocks at the end.

Sensitivity & robustness [3/4]

- **Alignment robustness.**
  - Analyzes the impact of inserting byte sequences at the beginning of an input whereby we add fixed and percentage blocks.

- **Test procedure:**
  - Test consists of two parameters, the maximum size M and the size of a step s.
  - Insert sequentially a block of size s at the beginning and stops after n steps when $n \cdot s \geq M$.
  - Two different modes:
    - Fixed blocks: M = 64 KB; s = 4 KB.

      We decided for a step size of 4 KB as this is the typical sector size.
    - Percentage blocks: M = 100%; s = 10%.

      We decided for a step size of 10% in order to analyze the impact of large changes. Especially logfiles may grow very rapidly.

Sensitivity & robustness [4/4]

- **Random-noise-resistance.**
  - Randomly driven test trying to produce false negatives.
  - E.g. a few changes all over the input are sufficient to obtain a non-match for ssdeep.

- **Test procedure:**
  - What is the maximum number of changes if the match score s is equal or above X, i.e., $s \geq X$ where = {90, 80, . . . , 0}.
  - Randomly change bytes all over the input.
    - Edit operations: deletion, insertion, and substitution.

General information about FRASH

- **Implemented in Ruby 2.0 and currently supports sdhash and ssdeep.**
- **Unix environment is necessary to run the framework.**
  - Find command is used.
- **FRASH is a command-line tool.**

```
$ frash [-v] [-r] [-t] PATH
```

  - -v: verbose – prints more details
  - -t: set the test scope: efficiency, single_common_block, fragment, alignment, random-noise.
  - -r: reads path recursively

Integrating new algorithms

- **Requirements for algorithms:**
  - Accept a directory and a file as input.
  - Print fingerprint to standard output, e.g., Base64 encoded.
  - The implementation needs to support an all-against-all comparison.

- **Integration:**
  - Create a wrapper: Copy the wrapper template and modify it.
    - E.g., which flag is used for all-against-all comparison.

Experimental results

- **Tools:**
  - ssdeep 2.9 and sdhash 3.2.
- **Test-corpus:**
  - T5 (4457 files, total 1.78 GB).
  - Types: jpg, gif, doc, xls, ppt, html, pdf and txt.
- **Remark:**
  - Test results are very comprehensive therefore this presentation only contains a rough summary.

Efficiency test - runtime

	Average	Total	Fingerprint comparison	$\frac{algorithm}{SHA-1}$
sha1sum	0.0013s	5.632s	-	1.00
ssdeep -s	0.0089s	39.789s	18.217s	7.06
sdhash	0.0167s	74.278s	346.730s	13.19
sdhash -p4	0.0066s	29.382s	346.902s	5.22

	Avg. hash length	Avg. ratio	Digest file size
sha1sum	20 B	0.00466 %	311 KB
ssdeep -s	57 B	0.01329 %	483 KB
sdhash	10.6 KB	2.52033 %	61.2 MB

- **Conclusion**
  - sdhash is slower than ssdeep but outperforms it when it is parallelized.
  - ssdeep shows a better compression.

S&R – Single-common-block correlation

- **File size of 2048 KB.**

	score	≥ 40	≥ 30	≥ 25	≥ 20	≥ 5
ssdeep	Avg. block size (KB)	605	384	368	-	-
	Avg. block size (%)	29.53	18.75	17.97	-	-
	Matches	5	5	4	-	-
sdhash	Avg. block size (KB)	912	720	604	480	170
	Avg. block size (%)	44.53	35.16	29.49	23.44	8.28
	Matches	3	5	4	4	5

- **Conclusion:**
  - sdhash is able to detect smaller, common blocks.

S&R – Fragment detection

- **Random cutting.**

	fragment size	50%	30%	25%	20%	5%
ssdeep	Avg. score	65.86	50.90	47.62	44.98	26.00
	Matches (%)	94.64	38.64	20.75	8.86	0.04
	Std. deviation	10.09	10.29	11.34	13.08	1.00
sdhash	Avg. score	69.49	70.63	71.18	71.91	76.16
	Matches (%)	100	99.46	98.86	97.33	75.59
	Std. deviation	22.45	23.17	23.27	23.22	22.72

- **Conclusion:**
  - ssdeep detect file fragments between 50% and 25%; high precision until 45% pieces then 'matches' reduces rapidly.
  - sdhash also identifies 5%-fragments in over 75% of all cases.

S&R – Alignment robustness

- **Fixed blocks...and percentage.**

	Added block	1 KB	4 KB	16 KB	32 KB	64 KB	...	400%
ssdeep	Avg. score	96.56	91.25	82.66	79.33	76.47		29.00
	Matches (%)	100	99.69	87.91	74.29	59.28		0.06
	Std. deviation	3.79	10.51	16.27	17.84	18.40		2.94
sdhash	Avg. score	84.11	51.47	64.37	52.68	78.12		67.52
	Matches (%)	100	100	100	100	100		100
	Std. deviation	10.57	21.04	17.01	21.05	15.90		21.98

- **Conclusion:**
  - sdhash detects all (100% matches) but score is alternating.
  - ssdeep runs into trouble the larger the inserted blocks.

S&R – Random-noise resistance

CASED

da/sec
BIOMETRICS AND INTERNET-SECURITY RESEARCH GROUP

	score	≥ 80	≥ 60	≥ 50	≥ 30	≥ 20	≥ 10
ssdeep	Avg. changes	14.65	43.89	85.17	160.00	-	-
	Avg. changes (%)	0.009 %	0.026%	0.050 %	0.094 %	-	-
	Matches	71	54	29	1	-	-
sdhash	Avg. changes	211.67	514.62	729.36	1116.24	1483.54	1860.83
	Avg. changes (%)	0.1216 %	0.304 %	0.431 %	0.660 %	0.877 %	1.100 %
	Matches	78	80	78	85	82	84

- **Conclusion:**
  - ssdeep is vulnerable against noise, e.g., only 29 matches for 85 changes.
  - sdhash is very robust, e.g., detects files with >10 while 1% of the bytes changed.

Take home messages

- **To establish approximate matching, we need to test algorithms.**
  - This shows strengths and weaknesses of approaches.
- **An automatic testing is now possible.**
  - No dedicated tests are needed anymore (e.g., Vassil 2011).
- **FRASH provides a first set of tests.**
  - Classes: efficiency AND sensitivity & robustness.
- **Open issues / future work:**
  - Integrate further algorithms.
  - Do we need further tests / test-classes?
  - How to obtain precision & recall rates? (See panel discussion at 1:45pm)

Thank you! - Questions?

- **Contact:**
  - da/sec – biometrics and internet-security research group darmstadt
  - Email: frank.breitinger@cased.de
  - Web: https://www.dasec.h-da.de/staff/breitinger-frank/
    - FRASH download.

"Your x-ray showed a broken rib, but we fixed it with Photoshop."