Digital Investigation 11 (2014) S10–S17

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Automated evaluation of approximate matching algorithms on real data

Frank Breitinger [a,*], Vassil Roussev [b]

[a] da/sec - Biometrics and Internet Security Research Group, Hochschule Darmstadt, Haardtring 100, 64295 Darmstadt, Germany
[b] Department of Computer Science, University of New Orleans, New Orleans, LA 70148, USA

Keywords:
Digital forensics
Approximate matching
Hashing
Similarity hashing
sdhash
mrsh-v2
ssdeep
FRASH

A B S T R A C T

Bytewise approximate matching is a relatively new area within digital forensics, but its importance is growing quickly as practitioners are looking for fast methods to screen and analyze the increasing amounts of data in forensic investigations. The essential idea is to complement the use of cryptographic hash functions to detect data objects with *bytewise identical* representation with the capability to find objects with *bytewise similar* representations.

Unlike cryptographic hash functions, which have been studied and tested for a long time, approximate matching ones are still in their early development stages and evaluation methodology is still evolving. Broadly, prior approaches have used either a human in the loop to *manually* evaluate the goodness of similarity matches on real world data, or controlled (pseudo-random) data to perform *automated* evaluation.

This work's contribution is to introduce automated approximate matching evaluation on real data by relating approximate matching results to the longest common substring (LCS). Specifically, we introduce a computationally efficient LCS approximation and use it to obtain ground truth on the *t*5 set. Using the results, we evaluate three existing approximate matching schemes relative to LCS and analyze their performance.

Introduction

One of the biggest challenges facing a digital forensic investigation is coping with the huge number of files that need to be processed. This is a direct result of the exponential growth in our ability to store digital artifacts and the overall trend of digitizing all forms of information, such as text, documents, images, audio and video. Consequently, a critical requirement of modern forensic investigative tools is the ability to perform large-scale automated filtering and correlation of data.

One of the most common processing methods is *known file filtering*, which–in its basic form–consist of computing the crypto hashes for all files on a target device, and comparing them to a reference database. Depending on the underlying database, files are either filtered out (e.g., files of the operating system) or filtered in (e.g., known offensive content). For example, NIST maintains a large public database of known content–the NSRL (NIST Information Technology Laboratory, 2003–2013).

Reference databases based on crypto hashes provide precise and reliable results; however, they can only identify content based on *identity*. This makes them fragile and difficult to maintain as digital artifacts (such as code) get updated on a regular basis, making the reference data obsolete. Therefore, it is useful to have algorithms that provide *approximate* matches that can correlate closely related versions of data objects.

* Corresponding author.
E-mail addresses: frank.breitinger@cased.de (F. Breitinger), vassil@roussev.net (V. Roussev).

http://dx.doi.org/10.1016/j.diin.2014.03.002

Generally, an approximate matching algorithm extracts features of an input and produces a similarity digest; the digests can then be compared to determine a measure of similarity. Depending on the level of at which the algorithm operates, one distinguishes between bytewise, syntactic- or semantic approximate matching Breitinger et al. (2014).

Semantic matching operates at the highest level of abstraction and provides results that are closest to human perceptual notions of similarity. Syntactic matching relies on purely syntactic rules to break up the data representation into features, e.g., cutting a header of a fixed byte size. Bytewise matching relies only on the sequence of bytes that make up a digital artifact, without reference to any structures (or their interpretation) within the data stream. In what follows, we focus on bytewise approximate matching.

While crypto hashes are well-known and established in various fields of computer science, approximate matching is a rather new area and missing standardize processes for testing and evaluating these algorithms. Breitinger et al. (2013a) presented a test framework called FRASH which performs efficiency as well as sensitivity & robustness tests. Some time later FRASH was extended by a precision & recall test on synthetic data (Breitinger et al., 2013b).

The main contribution of this work is the development of *automated* precision and recall tests on *real world data*. Compared to synthetic data, real world data yields more realistic results and allows a better characterization of the behavior of approximate matching algorithms. For this test, we first created a ground of truth wherefore we identified the similarity of objects based on an own metric called *approximate longest common substring* (aLCS). We validate our aLCS results by comparing them against the traditional *longest common substring* (LCS). In a second step, we analyze the false positive and false negative rates of the approximate matching algorithms with respect to the ground truth.

The rest of the paper is organized as follows: Sec. 2 introduces the necessary background and related work. Our evaluation methodology as well as some implementation details are provided in Sec. 4. The core of this paper is Sec. 5 where we present our experimental results. Sec. 6 concludes the paper.

Background & related work

Hash functions (e.g., SHA-1 Gallagher and Director (1995)) have a long tradition and are applied in various fields of computer science like cryptography (Menezes et al., 2001), databases (Sumathi & Esakkirajan, 2007, Sec. 9.6) or digital forensics (Altheide and Carvey, 2011, p. 56ff). This is in contrast to bytewise approximate matching which probably had its breakthrough in 2006 with an algorithm called context triggered piecewise hashing (CTPH). In the following we give a brief overview of bytewise approximate matching and explain how these algorithms are tested.

Bytewise approximate matching algorithms

The introduction of approximate matching for forensic purposes dates back to 2006 when Kornblum (2006) presented an approach called *context triggered piecewise hashing* and Roussev et al. (2006) introduced *similarity hashing*. Subsequently a small community came up which follows the challenges of approximate matching (a.k.a. 'similarity hashing'). To date, several different approaches have been published, all with different strength and weaknesses.

Besides the two most prominent implementations `ssdeep` and `sdhash`, a couple of further algorithms raised. However, most of them are very limited. For instance, `MinHash` (Broder, 1997) and `SimHash` (Sadowski and Levin, 2007) allow to detect small changes (up to several bytes) only, `bbHash` is too slow (2 min for 10 MiB), `mvHash-B` is not file type independent. Hence, in what follows we briefly describe the three most promising approaches with respect to digital forensics (a detailed description is beyond the scope of this paper as we treat them black boxes for testing purposes).

ssdeep

The `ssdeep` tool ([1]) was introduced as a proof of concept implementation of *context triggered piecewise hashing* (CTPH) and has gained widespread acceptance. It was presented by Kornblum (2006) and is based on the spam detection algorithm from Tridgell (2002–2009). The basic idea is behind it is simple: split an input into chunks, hash each chunk independently and concatenate the chunk hashes to a final similarity digest (a.k.a. fingerprint).

In order to split an input into chunks, the algorithm identifies trigger points using a rolling hash (a variation of the Adler-32 function) which considers the current context of seven bytes. Each chunk is then given to the non-cryptographic hash function FNV Noll (1994–2012). Instead of using the complete FNV hash, CTPH only takes the least significant 6 bits which is equal to one Base64 character. Thus, two files are similar if the have common chunks.

Follow up efforts (Chen and Wang, 2008; Seo et al., 2009; Baier and Breitinger, 2011) have targeted incremental improvement of the algorithm, however, none of these implementations have been made available for public testing and evaluation.

sdhash

The `sdhash` tool[2] was introduced four years later Roussev (2010) in an effort to address some of the shortcomings of `ssdeep`. Instead of dividing an input into chunks, the `sdhash` algorithm picks statistically improbable features to represent each object. A feature in this context is a byte sequence of 64 bytes, which is hashed using SHA-1 and inserted into a Bloom filter (Bloom, 1970). The similarity digest of the data object is a sequence of 256-byte Bloom filters, each of which represents approximately 10 KB of the original data. The tool also supports *block* mode (Roussev, 2012) in which the input is split into fixed-size chunks (by default 16 KiB) and the best features are selected from each block.

[1] http://ssdeep.sourceforge.net (last accessed 5 Dec. 2013).
[2] http://sdhash.org (last accessed 5 Dec. 2013).