Digital Investigation 10 (2013) 361–371

Contents lists available at ScienceDirect

Digital Investigation

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

F2S2: Fast forensic similarity search through indexing piecewise hash signatures

Christian Winter*, Markus Schneider, York Yannikos

Fraunhofer Institute for Secure Information Technology SIT, Rheinstr. 75, 64295 Darmstadt, Germany[1]

ARTICLE INFO

Article history:
Received 23 May 2013
Received in revised form 23 July 2013
Accepted 11 August 2013

Keywords:
Digital forensics
Similarity search
Indexing
Piecewise hashing
ssdeep
n-Gram

ABSTRACT

Fuzzy hashing provides the possibility to identify similar files based on their hash signatures, which is useful for forensic investigations. Current tools for fuzzy hashing, e. g. ssdeep, perform similarity search on fuzzy hashes by brute force. This is often too time-consuming for real cases. We solve this issue for ssdeep and even a larger class of fuzzy hashes, namely for piecewise hash signatures, by introducing a suitable indexing strategy. The strategy is based on *n*-grams contained in the piecewise hash signatures, and it allows for answering similarity queries very efficiently. The implementation of our solution is called F2S2. This tool reduces the time needed for typical investigations from many days to minutes.

1. Introduction

Forensic investigators suffer from ever increasing amounts of data, especially unstructured data. Therefore, it is essential to automate as much work as possible, and automatic tools should be highly effective and efficient.

A typical task for a forensic examiner is to identify digital documents related to criminal activities. An effective method for supporting this work is matching files against black- and whitelists. Hence practitioners often utilize forensic software tools for matching files based on cryptographic hashes of types like MD5 and SHA-1. The NIST Information Technology Laboratory (2003–2013) provides a huge reference collection of such hashes through the National Software Reference Library (NSRL).

On the one hand, cryptographic hashes provide an efficient means to match files against black- or whitelists. On the other hand, any small change in a file results in a completely different hash. Hence files cannot be recognized anymore after subtle changes – like inserting a single blank into a text document. Consequently, illegal documents will be missed by the blacklist after they have been modified (wittingly or unwittingly). For the same reason, benign files cannot be identified as such if the particular version is not in the whitelist – even if similar versions are contained.

In order to avoid these disadvantages of cryptographic hashes, researchers have proposed various kinds of similarity preserving hash functions for digital forensics. The hashes produced with such hash functions are called *fuzzy hashes*; one particular class of fuzzy hashes are *piecewise hash signatures*; cf. Section 2. A prominent software that creates piecewise hash signatures is ssdeep from Kornblum (2006a,b). A different example is bbHash from Breitinger and Baier (2012b). More tools and methods for fuzzy hashing are mentioned in Section 8.

The usage of fuzzy hashes raises a new challenge: While a reference list of cryptographic hashes has to be queried only for exact matches, a reference list of fuzzy hashes has to be queried for (potentially many) entries similar to the query item. Finding exact matches can be solved efficiently

* Corresponding author. Tel.: +49 6151 869 259; fax: +49 6151 869 224.
E-mail addresses: christian.winter@sit.fraunhofer.de (C. Winter), markus.schneider@sit.fraunhofer.de (M. Schneider), york.yannikos@sit.fraunhofer.de (Y. Yannikos).

[1] http://www.sit.fraunhofer.de.

1742-2876/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.diin.2013.08.003

with ordinary databases. Finding similar items cannot be solved that easily. Other indexing strategies are necessary to gain efficiency for similarity queries. By now no suitable approach has been proposed for efficient similarity search over piecewise hash signatures. Therefore, current fuzzy hashing tools run a brute force search over the whole reference list in order to answer similarity queries. This may take days or weeks for processing a typical harddisk. Thus the brute force method is completely impractical for real cases.

This paper presents an indexing strategy for piecewise hash signatures (Section 4). We use the indexing method for similarity matching of ssdeep hashes, and our experimental results in Section 7 present an impressive speedup compared to the search method of ssdeep. However, our method is not restricted to ssdeep and can be applied to other kinds of piecewise hash signatures, too.

2. Piecewise hash signatures

The general strategy of piecewise hashing is to divide a file into pieces and to produce a *piece hash* for each piece. The *piecewise hash signature* (PHS) of the file is the concatenation of all piece hashes. This strategy is driven by the goal that a small change in the file shall affect only a small portion of the PHS.

The most simple version of piecewise hashing is block hashing, where all pieces are consecutive blocks of a fixed size. As this strategy is not robust against insertion and deletion of bytes, its main purpose is the recognition of identical file fragments.

To be robust against insertion and deletion of bytes, the boundaries of pieces have to be determined based on the content of the file. Kornblum (2006a) has coined the term context triggered piecewise hashing (CTPH) for this strategy, and his implementation ssdeep (Kornblum, 2006b) is one particular instance.

A variation of piecewise hashing uses pieces of fixed size, but triggers the starting points of pieces based on the content of the file. Hence pieces may overlap, and there may be gaps between pieces as well. However, the concatenation of piece hashes produced for such pieces as in bbHash (Breitinger and Baier, 2012b) still yields a PHS.

While our solution is intended for piecewise hashing in general (cf. Section 5.2), we use ssdeep as basis for the evaluation of our indexing strategy because this program is a very prominent, prevalent, and mature piecewise hashing software. Our indexing strategy removes one limitation of ssdeep, but there are also other weaknesses, e. g. mentioned by Roussev (2011). In practice, each fuzzy hashing approach has certain advantages and disadvantages. It is the task of the research community to tackle the disadvantages and to develop improvements. In fact, some improvements of ssdeep have already been achieved (cf. Section 8), and we assume that in addition to our indexing strategy more improvements will follow.

2.1. ssdeep Hashes

The idea behind the CTPH mechanism of ssdeep is to determine the boundaries of pieces by a rolling hash function over a sliding window.[2] Whenever the rolling hash value has a certain property, a new boundary is triggered. The rolling hash function in ssdeep is a variation of the Adler-32 algorithm (Deutsch and Gailly, 1996) on a sliding window with a size of 7 bytes.

For creating a piece hash, a variation of the Fowler–Noll–Vo (FNV) algorithm (Noll, 1994–2012) is applied by ssdeep. Each piece hash is the base64 character corresponding to the six least significant bits of the FNV value. A PHS is just the concatenation of such piece hashes. Finally, an *ssdeep hash* consists of two PHS for reasons explained later in this section.

One aspect of ssdeep is that it adjusts the size of pieces to the file size so that the resulting number of pieces is in the same range for all files. Thus ssdeep uses larger pieces when hashing larger files. This is achieved by adjusting the triggering mechanism for piece boundaries: A piece size parameter p is derived from the file size, and a piece boundary is triggered whenever the rolling hash modulo p is equal to $p-1$. Hence a larger p produces less triggers. If the chosen p produces too few triggers, ssdeep reduces p; if it produces too many triggers, ssdeep discards the exceeding number of triggers. The parameter p is only allowed to take certain values, namely those from the geometric sequence 3,6,12,24,…, which has growth factor 2.

The parameter p (and thus the PHS) is not robust against small changes in a file. The insertion or deletion of one byte might change p by one *step* in the geometric sequence, i. e. p might be multiplied or divided by 2. As PHS for different piece size parameters cannot be compared, ssdeep provides two PHS in one ssdeep hash: a PHS for the parameter p and one for $2p$. This enables the comparison of ssdeep hashes even if their piece size parameters differ by one step. Note that the parameter p of the first PHS is declared as the piece size parameter of the ssdeep hash.

2.2. Generalized definition

This paper assumes that each piece hash is encoded with one byte. We denote a PHS as sequence of these bytes by $b = b_1 b_2 \ldots b_l$ where $l \in \mathbb{N}$ is the length of the signature. Each b_i $(1 \leq i \leq l)$ is an element of the byte alphabet $\mathcal{A} = \{0, \ldots, 255\}$. Thus a PHS is a member of $\mathcal{A}^*$. This definition of PHS holds for ssdeep, which restricts itself to a subset of $\mathcal{A}$ with only 64 elements, and for its variants mentioned in Section 8. Each piece hash of bbHash is encoded with 4 bits, and thus our definition does not apply. Nevertheless, our proposed solution can be adjusted to handle this case, too.

The principles used in our strategy for speeding up the similarity search do not make additional assumptions on the PHS; cf. Section 5.2. In particular, the strategy does not depend on the chosen method for producing piece hashes. However, when implementing our strategy as a proof of concept for ssdeep hashes, some details of ssdeep must be considered as we will see later.

[2] A rolling hash value should not be confused with a piece hash. It is only used to prepare the calculation of piece hashes.