Chapter 3:

Similarity Preserving Hash Functions

Frank Breitinger

Hochschule Darmstadt, CASED

SoSe 2012
Repetition

Similarity Preserving Hashing

Approaches and their Tools

Applications for Fuzzy Hashing

Peculiarities
Repetition

Similarity Preserving Hashing

Approaches and their Tools

Applications for Fuzzy Hashing

Peculiarities
Type: Similarity Preserving Hash Functions [1/2]

- Further property: similar inputs yield similar hash values.
  - Alignment Robustness.
  - Non-Propagation.
  - If possible: Also fulfill expectations for cryptographic hash functions (partly).

- Field of Application:
  - Detection similar files during a forensic investigation (e.g., different versions of files).
  - Biometrics: Template protection.
  - Malware: Detect obfuscated malware (e.g., metamorphic malware).
  - Junk mail detection.
Repetition

Type: Similarity Preserving Hash Functions [2/2]

- Possible representatives:
  - Segment hashes (Tool `dcf1dd`).
  - Context-triggered piecewise hashes (Tool `ssdeep`).
  - Similarity digests (Tool `sdhash`).

- Example:
  - ‘I don’t have any fear at home.’ vs ‘I don’t have any bear at home.’
  - Is there a match?
    - SHA-1: no match.
    - ssdeep: similarity of 90%.
Repetition

Similarity Preserving Hashing

Approaches and their Tools

Applications for Fuzzy Hashing

Peculiarities
Problem

Let $X \subseteq \Sigma^*$ be the set of inputs, let $d_x$ denote a distance function on $X$ and let $x_1, x_2 \in X$.

In order to identify the similarity between two inputs $x_1$ and $x_2$ we run into two problems:

1. Both inputs $(x_1, x_2)$ might be very ‘large’ and thus calculating $d_x(x_1, x_2)$ is very time consuming.

2. If $x_1$ is known and we want to compare it against $x_2$, we need to have $x_1$ available (disk space problem).
Solution

The aim is to use a compression function that obtain the similarity of the domain.

The Definition in the following is an own creation.
Similarity Preserving Hashing

Naming

Several terms for the compression function: similarity preserving hashing, similarity digest, fuzzy hash function or similarity preserving hash function.

We use the term approach for similarity preserving hashing which consists of a

similarity preserving hash function, which is a function / algorithm (which is denoted by $h$) to build a hash value / fingerprint and a
distance function, (denoted by $d_y$) that outputs a similarity score for two hash values / fingerprints. The term fingerprint or hash value is due to cryptographic / traditional hash functions.
Solution: Possible candidates

Let \( h \) be called a similarity preserving hash function with \( h : X \rightarrow Y \) and let \( d_y \) denote a distance function on \( Y \). Then a possible solution candidate is a setting of \( (Y, h, d_y) \) with the following properties:

1. \( |Y| < |X| \).
2. \( h \) is fast computable.
3. \( d_y \) is fast computable.

Questions:

- Compare these properties to the ones of cryptographic hash functions. What is the difference?
- What is fast?
Valid solution candidates

A solution candidate is valid, if there is a $\varepsilon_y$:

$$\forall x_1, x_2 : \text{ if } d_x(x_1, x_2) \leq \varepsilon_x,$$

$$\text{ then } d_y(h(x_1), h(x_2)) \leq \varepsilon_y$$

This issue is called \textit{correctness}.

In words:
Similar inputs yield similar outputs.
Quality of solution candidates

The quality of a solution candidate can be measured by its \textit{completeness} which is defined as follows:

\[
\forall x_1, x_2 : \quad \text{if } d_y(h(x_1), h(x_2)) \leq \varepsilon_y,
\]
\[
\text{then } d_x(x_1, x_2) \leq \varepsilon_x
\]

In words:

Similar outputs imply similar inputs.

Often completeness is a little bit too strict. Thus we introduce the term \textit{\(p\)-completeness}: Let \(p\) be a probability with \(0 \leq p \leq 1\).

\[
\forall x_1, x_2 : \quad \text{if } d_y(h(x_1), h(x_2)) \leq \varepsilon_y,
\]
\[
\text{then } P(d_x(x_1, x_2) \leq \varepsilon_x) \geq p
\]
Main Goal

1. Overcome ‘drawbacks’ of cryptographic hash functions in different application contexts.

2. E.g., for computer forensics the main drawbacks are:
   - Data acquisition: Integrity of copy is destroyed, if some bits change.
   - White-/Blacklisting:
     - Suspect files similar to known-to-be-bad-files are not detected.
     - Fragments are not detected (due to deletion, fragmentation).
Currently known approaches:

- Segment hashes (also called block hashes): Tool dcf1dd.
- Context-triggered piecewise hashes: Tool ssdeep.
- Similarity digests: Tool sdhash.
Approaches and their Tools

Repetition

Similarity Preserving Hashing

Approaches and their Tools

Applications for Fuzzy Hashing

Peculiarities
Segment Hashes including `dcfldd`
Segment Hashes

1. Underlying idea:
   - Split input data (volume, file) in blocks of fixed length.
   - Compute for each segment its cryptographic hash.

```
Input file m
```

```
m_1  m_2  m_3  ....  m_t
```

```
h(m_1)  h(m_2)  h(m_3)  ....  h(m_t)
```

2. Original aim: Improve integrity of storage media.
Segment Hashes: Example - dcfldd

```
$ dcfldd if=/dev/hda1 of=image-hda1.dd bs=512 hashwindow=4096 hash=sha256

0 - 4096: da0bd2b16c7cd5acb5695e9d81fb6d832cba85312d87e08d0c675e41b608de50
4096 - 8192: 281f4b8ac2dcda0f3fd9a0642a694f6df829d7567a531b1cfc8925f94eebe7a3
8192 - 12288: 1c05a3c7251b666c1ec4a2b689e25f95a92a311613ce685fd7cbf41552290e5
12288 - 16384: 6c9c17f271f18587bccc5f8f9c6154b4baf764664a3eb8ddf06881168c5c4698b
16384 - 20480: c61cd658e73450dfb0dfc9a1d83cbbdd162d9194d81f27f0516bb107280e841
[REMOVED]
```
Segment Hashes: Example from NIST

1. Sample tool of Nicholas Harbour (since 2002):
   - dcfldd: An extension of dd.
   - Department of Defense – Computer Forensics Laboratory.
   - Provides MD5, SHA-1, SHA-2 family.

2. Evaluation by NIST (Douglas White, 2008):
   - Hashing of File Blocks: When Exact Matches are not Useful.
   - NIST worked on Windows 2000 and XP OS files.
   - Main result:
     - File-based data reduction leaves an average 30% of disk space for human investigation.
     - Incorporating block hashes reduces this to an average of 15%.
     - Assist in recognising wiped media.
Approaches and their Tools

Segment Hashes: Evaluation

1. Anti-Blacklisting is very easy:
   ▶ Introduce an irrelevant byte in the first sector.
   ▶ All segment hashes differ from the stored segment hashes.
   ▶ Modified suspect file is not detected.

2. A good technique for whitelisting (see NIST results).

3. Size of segment hash database is large:
   ▶ 4096 byte block size, SHA-1.
   ▶ \[ \frac{\text{size of hash value in bytes}}{\text{size of raw data in bytes}} = \frac{20}{4096} = 0.00488 \]
   \[ \Rightarrow \] 1 terabyte of raw data yields a 5 gigabyte hash database.

4. Hash database depends on the hashwindow size.
Approaches and their Tools

Context Triggered Piecewise Hashes including ssdeep
Context Triggered Piecewise Hashes

1. Underlying idea:
   - Split input data (volume, file) in blocks of variable length.
   - Compute a hash value for each segment.
   - The sequence of these segment hashes is the context triggered piecewise hash of the input.

2. Question: How do we achieve blocks of variable length?
Context Triggered Piecewise Hashes (CTPH)

The end points of the blocks are determined by a pseudo random function called rolling hash:

- Its value only depends on the current context.
- A window (e.g., of size 7 bytes) slides over the input.
- Context = Bytes of input data in the current window.
- If rolling hash matches a trigger value, an end point is set.

\[
\begin{align*}
\text{Four score} & \rightarrow 83,742,221 \\
\text{Four score} & \rightarrow 5 \\
\text{Four score} & \rightarrow 90,281
\end{align*}
\]
CTPH: A sample tool

1. ssdeep (based on spamsun).

2. Window size is 7.

3. CTPH is a sequence of printable characters:
   - LS6B are encoded base64.
   - Only the least significant 6 bits (LS6B) of a block hash are considered.

4. ssdeep decides about a match:
   - On base of the weighted edit distance (changing, inserting and deleting characters) of two CTPHs.
   - Edit distance is rescaled to a percentage match score.
Approaches and their Tools

Piecewise Hashes: The algorithm

last Trigger Seq.  Window

HELLO  _MY_ WORL D

RH(_MY_WOR) == Trigger Value

HASH (HELLO_MY_WOR) = 32234013
32234013 % 64 = 5

X  5 = E

Signature: ... X E ...

Frank Breitinger
Hash Functions in Forensics / SoSe 2012
Approaches and their Tools

Piecewise Hashes: The algorithm

**Algorithm 1 CTPH**(BS, PF, TV, h)**

**Input:** A string of bytes BS of length N.
A pseudo random function PF : \(\{0, 1\}^{8w} \rightarrow \{0, 1\}^{l}\).
A trigger value TV.
A hash function \(h : \{0, 1\}^* \rightarrow \{0, 1\}^{n}\).

**Output:** The corresponding context triggered piecewise hash of the byte string BS.

\[i \leftarrow 0; c \leftarrow 0;\]
\[CTPH \leftarrow \;'; \quad //\text{initialise CTPH as empty string}\]
\[B_{c-w+1} \leftarrow 0; B_{c-w+2} \leftarrow 0; \cdots; B_{c-1} \leftarrow 0; \quad //\text{initialise padding bytes}\]
\[\textbf{while } i < N \textbf{ do}\]
\[\quad \textbf{if } PF(B_{i-w+1}B_{i-w+2} \cdots B_i) = TV \textbf{ then}\]
\[\quad \quad CTPH \leftarrow CTPH \| h(B_cB_{c+1} \cdots B_i);\]
\[\quad \quad c \leftarrow i + 1; B_{c-w+1} \leftarrow 0; B_{c-w+2} \leftarrow 0; \cdots; B_{c-1} \leftarrow 0;\]
\[\quad i \leftarrow i + 1\]
\[CTPH \leftarrow CTPH \| h(B_c\|B_{c+1}\| \cdots \|B_{N-1});\]
\[\text{return } (CTPH);\]
CTPH: Sample Research Questions

1. Rolling hash:
   ▶ Shall be efficient and pseudo random.
   ▶ Current implementation is fast, but not pseudo random.
   ▶ Task: Find a slightly slower, but pseudo random rolling hash.

2. Fragment detection:
   ▶ Kornblum’s approach fails, if fragments are much smaller than the original file.
   ▶ Task: Find a different approach.

3. Edit distance:
   ▶ Kornblum’s approach addresses text files (due to spam detection).
   ▶ Task: Find a more general approach, which also addresses images, videos, ...
Approaches and their Tools

CTPH: Origin & Evaluation

1. Originally proposed for spam detection (*spamsum* by Andrew Tridgell, 2002)

2. Ported to forensics by Jesse Kornblum, 2006: *ssdeep*.

3. Evaluation by different researchers:
   - Some publications to improve *ssdeep*.
     - Performance.
     - Detection rate.
     - Rolling hash.
   - Main conclusion: Fail a security analysis → not usable in forensics.
Approaches and their Tools

Similarity Digests including sdhash
Similarity Digests

1. Underlying idea:
   - Use statistical improbable features from input data (volume, file) of 64 bytes.
   - Generate a cryptographic hash (SHA-1) for each feature.
   - Divide each hash value in 5 sub-hashes in order to set five bits within a Bloom filter.
Similarity Digests: Example from Roussev

1. Sample tool of Vassil Roussev (since 2010):
   - sdhash.
   - Very popular approach; NIST provides databases.

   - Robust approach with several design errors.
   - Performance is slower than ssdeep.
     - A parallelized version is coming.
Approaches and their Tools

Similarity Digests: A sample tool

Preparation:

1. Select statistically improbable features on the basis of their entropy.
   ▶ A ‘feature’ is a (substring-) sequence of 64 consecutive bytes.
Similarity Digests: A sample tool

Feature selection:

2. Shannon entropy score $H$ is calculated where $P(X_i)$ is the empirical probability (i.e., the relative frequency) of encountering ASCII code $i$ within a feature.

$$H = - \sum_{i=0}^{255} P(X_i) \cdot \log_2 (P(X_i))$$

3. Roussev uses a precedence rank $R_{prec}$. The least likely features measured by its entropy score gets the lowest rank.
Identification of *popular* features:

4. A sliding window of a size 64 is going through all $R_{prec}$ values. At each position, sdHash increments the $R_{pop}$ score for the leftmost feature with the lowest $R_{prec}$.
Similarity Digests: A sample tool

Feature selection:

(input byte stream)

Select statistically improbable features
Similarity Digests: A sample tool

Fingerprint generation:

5. Hash underlying byte sequences for all $R_{pop}$ scores higher than a certain threshold using SHA-1.
   - Split hash (160 bit) into 5 sub-hashes of 32 bits and use the 11 least significant bits of each sub hash.
   - E.g., 24645aa1 3b9b9d0a b6e2da45 89fcd42d 215de81c
     Least significant 11 bits of each sub-hash:
     24645aa1 = aa1 = 1010 1010 0001
     11 bits: 010 1010 0001

6. Depending on these $5 \times 11$ bits set 5 bits within a Bloom filter (size 256 bytes $= 2048$ bits $= 2^{11}$ bits).
   - A Bloom filter is a bit vector / array.
Bloom filter

- Empty Bloom filter is a bit array of $m$ bits, all set to 0.
- $k$ different hash functions are defined (e.g., sub-hashes within `sdhash`).
- Each hash function maps or hashes some set element to one of the $m$ array positions with a uniform random distribution.

To query for an element (test whether it is in the set), feed it to each of the $k$ hash functions to get $k$ array positions.
Applications for Fuzzy Hashing

Repetition

Similarity Preserving Hashing

Approaches and their Tools

Applications for Fuzzy Hashing

Peculiarities
Applications

1. Forensics (on the file level): Detect similar files.
   - Blacklisting:
     - Detect manipulated suspicious files.
     - Find fragments of suspicious data.
     - Identify similar versions of files.
   - Whitelisting: Find changed unsuspicious files?
Applications for Fuzzy Hashing

Applications

   ▶ Due to privacy only save hash value.
   ▶ Why do we need similarity preserving hashing and not cryptographic hashing (e.g., SHA-1)?

3. Malware Detection:
   ▶ Detect obfuscated malware (e.g., metamorphic malware).

4. Junk mail detection.
Peculiarities

Repetition

Similarity Preserving Hashing

Approaches and their Tools

Applications for Fuzzy Hashing

Peculiarities
Peculiarities

Database comparison

Let \( n \) be the number of hash values entries within a database.

- Comparison time for cryptographic hash functions: \( O(\log_2 n) \).
- Comparison time for similarity preserving hashing: \( O(n) \).
  - Ordering is not possible.
  - Fragment detection.
Peculiarities

Drawbacks

1. *Compression* → size of the hash value.
   - Fixed size of few bits vs. variable of several thousand bytes
2. *Ease of computation* → creating a hash value.
   - Currently cryptographic hash functions are faster.
Whitelisting

Does it make sense to use non-cryptographic hash functions for whitelisting?
Questions?

"That’s our CIO. He’s encrypted for security purposes."

Copyright 2002 by Randy Glasbergen.
www.glasbergen.com

http://www.glasbergen.com/