Bringing Order to Approximate Matching: Classification and Attacks on Similarity Digest Algorithms

By:
Miguel Martín-Pérez, Ricardo J. Rodríguez and Frank Breitinger

From the proceedings of
The Digital Forensic Research Conference
DFRWS EU 2021
March 29 - April 1, 2021

Bringing Order to Approximate Matching: Classification and Attacks on Similarity Digest Algorithms

Miguel Martín-Pérez, Ricardo J. Rodríguez, Frank Breitinger

miguelmartinperez@unizar.es, rjrodriguez@unizar.es, frank.breitinger@uni.li

Universidad Zaragoza

March 31, 2021

DFRWS EU 2021

Cyberspace

Agenda

Agenda

Introduction

Similarity Digest Algorithms (SDA)

Similarity Digest Algorithms measure the bytewise similarity between two digital artifacts. First, they generate a digest of the artifacts, and then, they compare the similarity between digest, returning a similarity score

- Subtype of Approximate Matching Algorithms:
  - Identify similarity between digital artifacts
  - *Bytewise*: Comparison relies on the raw sequence of bytes
- **Similarity digest**: Aggregation of artifact's features
- **Similarity function**: Compares two similarity digest and returns a **similarity score**

Introduction
Methodology

- **Review relevant literature**
  - Algorithm description
  - Comparison of algorithms
  - Security evaluation
  - Suggest properties for approximate matching algorithms
- **Extract characteristics, features, and peculiarities**
- **Align characteristics**

Agenda

Categorization scheme for SDA

- **Artifact processing and digest generation**
  - **Feature generation** Extracts features from the input
  - **Feature processing** Maps and reduces the features yielding processed features
  - **Feature selection** Selections of features
  - **Digest generation** Transforms processed features to similarity digests
  - **Feature deduplication** Eliminates redundant processed features
- **Digest comparison** Compares two digest and returns a similarity score

Note: **Order of phases is not fixed**

Feature generation phase

Extract features from the input

- **Length**: Feature size
  - Static
  - Dynamic
- **Support function**: Allow identifying the feature boundaries
  - Trigger function: Compare output with a predefined value
  - Unique: Split input building a feature set where all features are unique
- **Intersection**: One byte belong to several features
  - Yes
  - No
- **Cardinality**: Amount of generated features
  - Variable: Depend on the input size
  - Fixed: Determined by algorithms, e.g., `ssdeep`

Feature preprocessing phase

Map and reduce the features yielding *processed features*

- **Mapping function** (a.k.a. *compression function*)
  - Hashing: Pseudo-random value e.g., MD5, SHA, FNV-1a
  - Encoding: Run Length Encoding
  - Identifier: Swap features by their identifiers
- **Bit reduction** Select few bytes from mapped features
  - Ratio between used bits and size of mapped output
  - None: Consider all bits

Feature selection phase

Select the features that will form the digest
(**optional phase**)

- **Selection function**
  - Minimum probability: Select features that are least likely to occur
  - Block matching: Select features that match with one of fixed blocks
  - Block similarity: Select features that are similar to one of fixed blocks
  - Minimum values: Select *n* features whose processed features have the *n* minimum values
- **Domain** Input of the selection function
  - Features
  - Processed features
- **Coverage**
  - Full: All input bytes belong to at least one feature in the digest
  - Partial: There are bytes not consider by any selected feature

Digest generation phase

Transform processed features to similarity digests

- **Digest size**
  - Fixed: always the same length, regardless of the input length
  - Input dependent: size depends on the input length
  - Input dependent with max: size depends on the input length but has a maximum length
- **Storing structure**
  - Processed feature concatenation: processed features are just concatenated
  - Set concatenation: sets with maximum capacity, e.g., Bloom filters
  - Set
  - Counter: Store how many times each feature appears
- **Order**
  - Absolute: All features keep the occurrence order
  - Set-absolute: Order between sets but not between features with a set
  - Processed feature–aware: Overlap between features store the occurrence order
- **Requirements**
  - Minimum features
  - Diversity: need variability in the input artifacts
  - Document frequency: Feature frequency in a training document set

Feature deduplication phase

Eliminate redundant processed features
(**optional phase**)

- **Type**
  - Consecutive: Consecutive features are reduced to a short sequence
  - In-scope: Identical features in the same scope are eliminated
- **Occurrence phase**
  - Digest generation
  - Digest comparison: e.g. `ssdeep`

Digest comparison phase

Compare two digest and return a similarity score, is not a percentage

- **Requirements**
  - Minimum commonality: always the same length, regardless of the input length
  - Minimum amount: size depends on the input length
  - Similar input size: size depends on the input length, but has a maximum length
- **Output score**
  - Binary value: inputs are similar or not (yes/no)
  - Interval: [0, 1] or [0, 100]
  - Half-bounded: lower boundary, but no upper boundary (for measuring dissimilarity)
- **Score trend**
  - Ascending: the higher the output score → the higher similarity of digests
  - Descending: the lower the output score → the higher similarity of digests
- **Space sensitivity** Sensitiveness to a different order of the same features
  - Total: Total order of the features is considered
  - Partial: Only considers the order of features partially
  - None: The order is not considered

Classification of similarity digest algorithms

Algorithm	Feature generation				Feature Processing		Feature Selection		
	Length	*Support Function*	*Intersection*	*Cardinality*	*Mapping Function*	*Bit Reduction*	*Selection Function*	*Domain*	*Coverage*
dcfldd	Static (512)	None	No	Variable ($L/512$)	Hashing	None (128)	None	(n/a)	Full
Nilsimsa	Static (3)	None	Yes	Variable ($6L$)	Hashing	None (8)	None	(n/a)	Full
ssdeep	Dynamic ($L/64$)	Trigger function	No	Fixed (64)	Hashing	Ratio (6/32)	None	(n/a)	Full
md5bloom	Static (512)	None	No	Variable ($L/512$)	Hashing	Ratio (40/128)	None	(n/a)	Full
MRS hash	Dynamic (234)	Trigger function	No	Variable ($L/234$)	Hashing	Ratio (44/128)	None	(n/a)	Full
SimHash	Static (1)	None	Yes	Variable ($8L$)	Identifier	None (8)	Block matching	Feature	Partial
sdhash	Static (64)	None	Yes	Variable (L)	Hashing	Ratio (55/160)	Minimum probability	Feature	Partial
MRSH-V2	Dynamic (320)	Trigger function	No	Variable ($L/320$)	Hashing	Ratio (55/64)	None	(n/a)	Full
mvHash-B	Static (20, 50)	None	Yes	Variable ($8L$)	Encoding	Ratio (1/32)	Block similarity	Feature	Full
TLSH	Static (3)	None	Yes	Variable ($6L$)	Hashing	None (8)	None	(n/a)	Full
saHash	Static (1)	None	Yes	Variable ($4L$)	None	None (8)	None	(n/a)	Full
LZJD	Dynamic ($1 + \log_{256} L$)	Unique	No	Variable ($L/(1 + \log_{256} L)$)	Hashing	None (128)	Minimum value	Processed feature	Partial
FbHash	Static (7)	None	Yes	Variable (L)	Hashing	None (64)	None	(n/a)	Full

Algorithm	Digest generation				Feature Deduplication		Digest comparison			
	Digest Size	*Storing Structure*	*Order*	*Requirements*	*Type*	*Occurrence*	*Requirements*	*Output Score*	*Score Trend*	*Space Sensitivity*
dcfldd	Input dependent	Processed feature concatenation	Absolute	None	None	(n/a)	None	Interval	Ascending	Total
Nilsimsa	Fixed	Counter	Processed feature-aware	None	Consecutive	Comparison	Minimum commonality	Interval	Ascending	None
ssdeep	Input dependent with max	Processed feature concatenation	Absolute	Minimum features	Consecutive	Comparison	Minimum commonality, Similar input size	Interval	Ascending	Total
md5bloom	Input dependent	Set concatenation	Set-absolute	None	In-Scope	Generation	None	Interval	Ascending	Partial
MRS hash	Input dependent	Set concatenation	Set-absolute	None	In-Scope	Generation	None	Interval	Ascending	Partial
SimHash	Fixed	Counter	None	None	None	(n/a)	Similar input size	Half-bounded	Descending	None
sdhash	Input dependent	Set concatenation	Set-absolute	Diversity	In-Scope	Generation	Minimum amount	Interval	Ascending	Partial
MRSH-V2	Input dependent	Set concatenation	Set-absolute	None	In-Scope	Generation	Minimum amount	Interval	Ascending	Partial
mvHash-B	Input dependent	Set concatenation	Set-absolute	Diversity	In-Scope	Generation	Similar input size	Interval	Ascending	Partial
TLSH	Fixed	Counter	Processed feature-aware	None	None	(n/a)	None	Half-bounded	Descending	None
saHash	Fixed	Counter	Processed feature-aware	None	None	(n/a)	None	Binary value	(n/a)	Total
LZJD	Fixed	Set	None	None	None	(n/a)	None	Interval	Ascending	None
FbHash	Fixed	Counter	Processed feature-aware	Document frequency	None	(n/a)	None	Interval	Ascending	None

Classification of state-of-the-art SDA

Exiting classifications which are mostly based on categories creators assigned to their algorithms

- **Block-based hashing**: uses cryptographic hashes, generating and storing features for every block of a fixed size
- **Context trigger piecewise hashing**: splits the input into contexts, defined as a sliding window on the input bytes when the trigger function is activated
- **Statistically-improbable features**: uses a selection function based on statistically improbable features
- **Block-based rebuilding**: chooses blocks (randomly selected or pre-fixed) and generate the digests selecting the most similar blocks to the input
- **Locality-sensitive hashing**: maps objects into buckets, grouping similar objects in the same bucket with high probability

Classification of state-of-the-art SDA

Proposed a **new** and simpler classification based on the complete behavior

- **Feature Sequence Hashing**: split the input into features and maps them, measuring the similarity by feature sequences
- **Byte Sequence Existence**: identify the existence (or similarity) of byte sequences (*blocks*) in the input. The similarity score is calculated by comparing the number of common blocks between similarity digests.
- **Locality-Sensitive Hashing**: map objects into buckets, grouping similar objects in the same bucket with high probability

Classification of state-of-the-art SDA

Algorithm	Previous classification	New classification
dcfldd	Block-Based Hashing	Feature Sequence Hashing
ssdeep	Context Trigger Piecewise Hashing	Feature Sequence Hashing
md5bloom	Context Trigger Piecewise Hashing	Feature Sequence Hashing
MRS hash	Context Trigger Piecewise Hashing	Feature Sequence Hashing
sdhash	Statistically-Improbable Features	Feature Sequence Hashing
MRSH-V2	Context Trigger Piecewise Hashing	Feature Sequence Hashing
SimHash	Block-Based Rebuilding	Byte Sequence Existence
mvHash-B	Block-Based Rebuilding	Byte Sequence Existence
LZJD	(none)	Byte Sequence Existence
Nilsimsa	Locality-Sensitive Hashing	Locality-Sensitive Hashing
TLSH	Locality-Sensitive Hashing	Locality-Sensitive Hashing
saHash	(none)	Locality-Sensitive Hashing
FbHash	(none)	Locality-Sensitive Hashing

Agenda

Attacks against Similarity Digest Algorithms

Possible attacks against SDA and the characteristics of the algorithms facilitating these attacks

- **Attacks against the similarity score**:
  Modifying an input to manipulate its similarity score
  - Reduction of Similarity
  - Emulation of Similarity
- **Attacks against impeding the last phases of an SDA**:
  Crafting an input not comparable or always dissimilar
  - Impeding the Digest Generation Phase
  - Impeding the Digest Comparison Phase

Adversary model

Assuming an intelligent adversary who is knowledgeable about the processes and techniques used by the SDA

Reduction of Similarity

Minimize the similarity score between two inputs (e.g., deny-listing)

- **Mapping function**: hashing
  - Change 1 byte in each (or at least the majority) of the features
- **Feature length**: static; **Support function**: none
  - Add one byte at the beginning of the input will modify all subsequent features (all offsets shift)
- **Requirement** (digest comparison): minimum commonality
  - `ssdeep` needs 7 consecutive common features, modifying 1 out of 7 features for dropping the similarity score to zero
- **Storing structure**: set concatenation
  - Shifting half of the features of one filter are displaced to the next filter, their similarity drops significantly. For `sdhash`, this attack drops the similarity of approximately 28
- **Selection function**: block matching or minimum value
  - Search these particular set of blocks or features and modifying them accordingly (e.g., `LZJD`)
- **Selection function**: block similarity; **Intersection**: yes
  - Changing the amount of consecutive features (incrementing or decrementing it by one), this will cause that the subsequent derived features change

Emulation of Similarity

Crafting a new digital artifact that yields a high similarity score to a known artifact (e.g., allow-listing)

- **Cardinality**: fixed; **Support function**: trigger function
  - `ssdeep`: Craft an input that generates several small features with the beginning of the input filling up the similarity digest
- **Coverage**: partial
  - `sdhash`: Up to 20 % of the content of an input can be modified without influencing the generated digest
  - `LZJD`: Input alterable after the last selected feature while the new processed features have a high value

Impeding the Digest Generation Phase

Modify an input in a way that the algorithm is unable to generate a similarity digest due to the lack of necessary conditions

- **Support function**: trigger function; **Requirements** (digest generation): minimum features
  - `ssdeep`: Manipulate trigger function to yielding insufficient features for generating the digest
- **Requirements** (digest generation): diversity
  - `sdhash`: Discard features with low entropy

Impeding the Digest Comparison Phase

Hamper the digest comparison process

- **Type**: consecutive; **Requirements** (digest comparison): minimum commonality
  - `ssdeep`: Craft an input such that the digest, computed after deduplication, is smaller than the number of common elements expected
- **Requirements** (digest comparison): minimum amount
  - `sdhash`: Yield a similarity digest with less elements than required for comparison

Agenda

Building a Robust Similarity Digest Algorithm

Desirable characteristics to build a robust SDA
(resilient against attacks)

- **Feature generation**
  - Length: Static; Support function: None
  - Intersection: Yes
  - Cardinality: Variable
- **Feature processing**
  - Mapping function: Yes, but not hashing or not identifier
  - Bit reduction: None or Low reduction (ratio)
- **Feature selection**
  - Selection function: Yes, to reduce the digest size
  - Domain: ?
  - Coverage: **Full**

Building a Robust Similarity Digest Algorithm

- Digest generation
  - Digest size: Input dependent
  - Storing structure: Processed feature concatenation
  - Order: Absolute
  - Requirements: **None**
- Feature deduplication: **None**
- Digest comparison
  - Requirements: **None**
  - Output score: Interval or Half-bounded
  - Scoring Trend: ?
  - Space sensitivity: **Total**

Note

Focus on perfect error rates, not consider runtime efficiency

Agenda

Conclusions

- **Classification Scheme for Similarity Digest Algorithms**
  - Categorization scheme
  - New classification
- **Attacks against Similarity Digest Algorithms** with regard to the classification scheme
- Building a Robust Similarity Digest Algorithm: highlight the **desired properties that an SDA shall have to be robust against these attacks**

Bringing Order to Approximate Matching: Classification and Attacks on Similarity Digest Algorithms

Miguel Martín-Pérez, Ricardo J. Rodríguez, Frank Breitinger

miguelmartinperez@unizar.es, rjrodriguez@unizar.es, frank.breitinger@uni.li

Universidad Zaragoza

March 31, 2021

DFRWS EU 2021

Cyberspace