Designing the Elements of a Fuzzy Hashing Scheme

Jonathan Oliver
TrendMicro Research
Australia
jon_oliver@trendmicro.com

Josiah Hagen
TrendMicro Research
USA
josiah_hagen@trendmicro.com

***Abstract*—The goal of "fuzzy hashing" is to identify near duplicates and similar documents using hashes or digests. We consider a range of design elements that can be used with fuzzy hashing schemes. We examine the criteria for evaluating fuzzy hashes, and develop an approach for optimising a scheme to respect those criteria. We apply this approach to select appropriate design elements and parameter values. This results in surprising choices, the preference of skip-ngrams for adversarial problem domains; and that 2 bit vectors are preferred to 1 bit vectors.**

***Index Terms*—Fuzzy Hashing, TLSH, Adversarial Machine Learning, Ngrams, Skip-Ngrams**

I. Introduction:

There has also been a series of work on "fuzzy hashes" which allow us to identify that two documents are similar based on their digest values. TLSH is one of the state of the art fuzzy hashes [1]–[3]. Fuzzy hashing can be used for a range of applications associated with malware, such as detecting malware [4], [5] scalable clustering [6], [7] and identifying the closest legitimate file to an unknown file so that meta data and certificates can be compared [8].

Approaches to "fuzzy hashing" include Locality Sensitive Hashing (such as TLSH [1] and Similarity Digests (such as Ssdeep [9] and Sdhash [10]). Example applications include tasks such as malware and spam detection. Fuzzy hashing schemes are generally intended to work for a range of file types. NIST [11] offered criteria for the evaluation of fuzzy hashing approaches:

- Accuracy
- Compression (the digest should be short, preferably fixed length)
- Performance (digest generation time, and comparison time), and
- Security of results.

We expanded on the security elements of those criteria, and recommended that the methods need to be attacked during the design process and the resilience to attack should be measured and evaluated [12]. There has been research on attacking fuzzy hashing schemes including using random modifications [13] and identifying specific weaknesses [13], [14]. The deep learning community applied gradient descent algorithms to create adversarial images [15]. In this paper, we shall adopt these criteria for a case study in the development of a fuzzy hashing scheme which is based on the LSH paradigm [1], [16]. We consider a range of extensions, including the form of the features (ngram, skip-ngram [17] or string), the granularity of the features (measured in bits), ancillary parameters (such as length) and parameters which optimize the distance calculation. We attacked our schemes by adapt the gradient descent approaches [15] to search for a sequence of transformations that break the schemes. We present results on the accuracy, performance and resilience of the schemes.

II. Design Process

In this section, we describe a baseline scheme. Section II-B offers criteria for making these design decisions by optimising parameters for accuracy and rejecting schemes which are easy to attack or have poor performance characteristics. Section II-C describes alternative schemes to ngrams and identifies that skip-ngrams [17] should also be considered during any experiments. Section II-C offers another alternative to the baseline scheme, namely the use of a binary code to encode the digest. Section II-E describes other elements and parameters which could be sensibly bolted onto to a similarity digest scheme. We have a wide range of digests schemes which need to be experimentally evaluated according to the criteria from Section II-B

A. The Baseline Digest Scheme

A baseline approach to constructing a digest scheme could be a standard local sensitive hashing scheme [16] with B buckets, using byte ngrams of length N for input features and mapping the ngrams to the B buckets using a suitable hash function. Evaluating a digest would involve the following steps:

1) process the byte string to evaluate the ngrams, and calculate the B bucket counts for the hashed ngram values;
2) calculate the median bucket count.
3) output the digest consisting of the bit string (with length B) which take value 0 if the bucket count is above the median value and 1 if the bucket count is less than or equal to the median value.

The distance function would then be the hamming distance between two digests. The design elements of our digest scheme that we need to optimise is B and N. We assume that the choice of the function mapping ngrams to buckets has negligible effect on the outcome and use the Pearson hash [18]. We use $N = 5$ in the baseline design following the Nilsimsa [19] and TLSH schemes [1].

B. Optimization Approach

We consider a number of factors when making our design choices:

1) Accuracy:: It is primary consideration that our digest scheme can accurately reflects when byte strings are similar. This can be done using measures such as precision and recall, detection rates and false positive rates. There is a problem with using such measures; calculating them requires we specify some threshold. A suitable way to summarize precision and recall values is to use the Area Under Curve (AUROC) of a ROC curve. The advantages of using AUROC are (i) the AUROC reduces a range of accuracy measurements to a single number by integrating over the possible thresholds one could use when using the scheme; and (ii) using the AUROC requires that the digest scheme to work for a range of thresholds which covers a wide range of security scenarios. For example, we may use a high threshold with a high detection rate and high false alarm rate for identifying candidate samples to be put in a sandbox. On the other hand, we would typically require a very low false alarm rate when identifying files to quarantine. Neither Sdhash and Ssdeep have a range of values which are suitable for ROC analysis [1].

2) Performance (Time Complexity):: We require that the digest generation and distance calculations can be done in reasonable time and space complexity and prefer high performance schemes. Size of the Digest (Space Complexity): We require that the digests be of reasonable length, that is of the order of the length of schemes such as MD5 (128 bit), SHA1 (160 bit) and SHA256 (256 bit), perhaps somewhat longer. So we focus on schemes with digests in the range 128-256 bits and we prefer schemes where the number of bits is a multiple of 8.

3) Security:: We require that the digest scheme not be trivial for an attacker to evade. We measure the vulnerability to attack of a scheme by measuring the degradation in distance scores achieved by a search algorithm iteratively selecting transformations that maximise distance scores. We have a complex decision here. We will reject schemes that have unacceptable performance or security characteristics. We will then use accuracy as a primary selection criterion, noting that amongst schemes with almost equivalent accuracy, we will use performance and security considerations to select a preferred scheme. Our approach to design choices and parameter optimisation is to:

1) consider as many settings as possible;
2) remove candidates which were unacceptable either due to digest size or unacceptable performance or unacceptable security characteristics (trivial to evade);
3) select the candidate designs which had the highest AUROC;
4) amongst the candidate designs which have the highest accuracy, use performance and security considerations to select a preferred design.

C. Feature Selection

In addition to ngrams, we considered a range of features that are typical of many clustering and ML applications, though we did not consider features typical of image analysis and embeddings from Deep Learning that map problems into a problem space typical of image analysis. We considered 4 types of features:

- Ngrams are used for many clustering and ML applications. The use of long ngrams for security application has a serious short-coming (discussed below).
- Bag-of-words are suitable for applications where the files can be tokenized in some way. A common design requirement (and for the Security domain in particular) is that the scheme needs to be applicable to many file formats (text files, executable files on any architecture, source files, image files, etc.) This requirement removed bag-of-words from consideration.
- Variable length strings are used for by some schemes. For example, Ssdeep [9] uses a rolling hash to split a file into long segments. A checksum of the segment is taken and if these segments match, then this contributes towards matching.
- Skip-ngrams [17] are similar to as ngrams, except that when processing each window of size N, K bytes are dropped from the window. They are a far less frequently used in clustering and ML applications.

Skip-ngrams have many of the properties of ngrams with 2 differences:

1) Skip-ngrams may be require more computational resources; and
2) Skip-ngrams are useful for short strings.

Skip-ngrams potentially offer an attack surface which is considerably more difficult to attack than ngrams. We draw a distinction between long ngrams / strings and short ngrams / strings. Attacking an ngram solution (with large N) is straight forward and can be achieved by changing every Nth byte. For example, attacking Sdhash [10] (which uses-64 grams) can be achieved by changing every 64th byte. Similarly attacking the Ssdeep scheme is straight forward, by making sure that a byte in each long string extracted is modified. Short sed scripts that evaded Sdhash and Ssdeep for HTML and source code files are given in [13] (responsible disclosure done at the time). With shorter ngrams / strings (say $N = 3$), the amount of change to defeat a scheme is significant (changing 33% of the file when $N = 3$). Due to this reason, we consider shorter ngrams (with $N < 9$) for the remainder of the paper.

D. Bits Per Bucket (BPB)

With Locality Sensitive Hashing [16], it is standard to have a random projection function which maps each projection of the input sample (which we are assigning to a "bucket") to 0, 1. Similarly, in related fields such as Semantic Hashing, the autoencoder vector is typically mapped to a binary string [20]. We can consider functions which map a bucket onto 3 or more values. This introduces additional complexities in the distance

calculation since a hamming distance may longer be a suitable approach to compare values. We shall term the number of bits per bucket as BPB. Here we will consider schemes which use more values, and add additional parameters so that we can optimise our distance function. We considered two methods for allowing each bucket to have 3 values:

- allow BPB=2 bits in the digest for each bucket position. This would result in a digest which could be readily interpreted, but is inefficiently encode.
- efficiently encode the bits into a byte. While space efficient (the BPB would be fractional), this would lose the feature that the digest could be readily understood.

Therefore, we restricted the number of values per bucket to being either 2 (a binary code with BPB=1) or 4 (where each bucket is represented by BPB=2 bits).

E. Other Design Elements

We also considered how the length of the input byte string can be used. One approach is to add a function of the length to the digest. When we add this value to the digest we need to optimise a length parameter which accounts for any change to the distance function. We note that other elements have been previously proposed such as quartile ratios and checksums, but we do not have space to optimise these parameters in this paper.

III. Applying the Design Process

We now apply the design process as described in Section II. Section III-A describes a dataset suitable for security applications and labelled it in a manner which allows calculation of the AUROC for a given digest scheme. Section III-C looks at skip-ngrams, specifically looking at the computation complexity associated with various choices of K and N. We identify a list of candidate skip-ngram which can be used. In Section we perform tests on our set of digest schemes, and selected a candidate list with the highest AUROC scores.

A. The Data Set

We used a combined data set from [1] and [13] which consisted of 2 components: a distinct file set; and a similar file set. The distinct file set consisted of:

1) malware files (PEfiles) from different malware families
2) executables files (Elf files), both legitimate and randomly constructed HTML fragments,
3) random text selected from the Unix dictionary (with no overlapping words),
4) image files from spam images, and
5) text files about different topics.

They key to these files is that we have no reason to believe that they are similar. The similar file set consisted of groups of files where we could safely label as similar. For the malware files, we collected samples from the same malware family. For example, the similar set included 20 binary files from a set of malware families including the TROJ_DROPPER, TROJ_ZLOB, WORM_SOBER, etc malware families. We created similar sets for the text files by selecting words and replacing them with a word from another text file. Further variants were created by using the Linux command "fmt" (which is considered similar because it alters the formatting without altering the contents) and a random sort command (which altered the ordering of the lines without modifying their content).

B. Evaluating AUROC Values

Given a threshold T, we define a false positive as occurring when the distance between two distinct files ¡= T; and a true positive as occurring when the distance between two similar files ¡= T. With these definitions, we can plot a ROC curve and calculate AUROC.

C. Notes on Skip-ngrams

The smallest non-trivial skip-ngram is a $K = 2,\ N = 4$ skip-gram. When considering a window of length 4 bytes, there is 4C2 = 6 ways of selecting 2 bytes from the window, which we have labelled A-F below:

A:	X	X	-	-
B:	X	-	X	-
C:	X	-	-	X
D:	-	X	X	-
E:	-	X	-	X
F:	-	-	X	X

But choices D, E and F are redundant. They will be covered as a window moves along the byte string. We need to avoid double counting the same set of bytes extracted. We find that the number of ways of extracting non-redundant byte sets from a window of size N with K skips is:

$$Z = \binom{N-1}{N-K-1}$$

The speed in calculating the digest is dominated by the byte set extraction code [21] (version 3.11), so Z (the number of byte sets in a skip-ngram is a direct measure of the speed for calculating the digest. Table I gives a list of the number of non-redundant byte sets, Z, for a range of skip-ngrams with at least 3 bytes in it. The N column is the window size, K is the number of bytes skipped and BE is the number of bytes extracted. For reference, timing estimates of an optimized skip2-ngram5 ($Z = 6$) is of roughly the same speed as optimized MD5 and SHA1.

	N	K	BE	Z
skip1-ngram4	4	1	3	3C2 = 3
skip2-ngram5	5	2	3	4C2 = 6
skip3-ngram6	6	3	3	5C2 = 10
skip4-ngram7	7	4	3	6C2 = 15
skip5-ngram8	8	5	3	7C2 = 21
skip1-ngram5	5	1	4	4C3 = 4
skip2-ngram6	6	2	4	5C3 = 10
skip3-ngram7	7	3	4	6C3 = 20
skip4-ngram8	8	4	4	7C3 = 35

TABLE I
THE NUMBER OF NON-REDUNDANT BYTE SETS.

Digest length (bits)	AUROC	Representative Threshold	Representative FP rate	Representative Detection rate
48	0.82330	2	0.01	0.0877
128	0.9194	2	0.01	0.1818
256	0.9527	3	0.01	0.3156
256 BPB=2	0.9651	33	0.01	0.6356

TABLE II
EXPERIMENT 1: AUROC FOR BASELINE SCHEMES WITH $N = 5$.

Length Parameter	AUROC	Length Parameter	AUROC
0	0.9651	9	0.9843
1	0.9754	10	0.9843
2	0.9792	11	0.9843
3	0.9812	12	0.9843
4	0.9824	13	0.9842
5	0.9832	14	0.9841
6	0.9837	15	0.9839
7	0.984	20	0.9833
8	0.9842	N/A	N/A

TABLE III
EXPERIMENT 2: AUROC FOR LENGTH PARAMETER VALUES.

Design	AUROC Rep.	Threshold Rep.	FP rate Rep.	Detection rate
ngram3	0.9746	47	0.01	0.8589
ngram4	0.977	55	0.01	0.8512
ngram5	0.9858	69	0.01	0.903
ngram6	0.9794	68	0.01	0.8134
ngram7	0.9884	93	0.01	0.9308
ngram8	0.9876	94	0.01	0.9178
skip1-ngram4	0.9836	47	0.01	0.8555
skip2-ngram5	0.9852	51	0.01	0.9019
skip3-ngram6	0.9851	51	0.01	0.871
skip4-ngram7	0.9864	56	0.01	0.9145
skip5-ngram8	0.9846	53	0.01	0.8704

TABLE IV
EXPERIMENT 3: AUROC FOR VARIOUS NGRAM AND SKIP-NGRAM SCHEMES.

D. A Series of Experiments

In this section, we will start with our baseline method and perform experiments to determine candidate schemes with high AUROC scores. We will do this work as a series of experiments. The space of possible design choices is a complex space, and we did optimization of AUROC scores in many different experiments to arrive at a candidate list of schemes. In this paper, we shall describe this as a series of experiments which highlight the approach used.

Experiment 1: The number of buckets. We start with a binary scheme (BPB=1) and evaluate the AUROC while using a distance function which is the hamming distance between the digests. In Table II, we present results varying the digest length. We show the AUROC with a representative threshold, FP rate and detection rate. The bottom row in Table I shows the result using a bit per bucket value of 2, and 128 buckets. For this scheme we need to define an extension of hamming distance because we need to be able to determine the distance between two bucket values in the range of 0-3. We use the absolute difference of the two values. For example, if a bucket in the first digest has value 3 (binary 11) and the corresponding bucket in the second digest has a value of 1 (binary 01), then the distance contributed from this bucket is abs(3-1) = 2.

Interestingly the schemes with 256 bits (64 hex digits) have quite different results and we prefer the BPB=2 scheme (which is a non-standard choice).

Experiment 2: The length parameter. We start with the best scheme from Experiment 1 and do experiments to determine suitable values for the length parameter. The length parameter works in the following manner. Digest 1 and 2 have value for the floor(log(byte length)). The value added to the distance between digest 1 and 2 will be a multiple of the difference of the floor(log(byte length)), where the multiple used is termed the "length parameter". Table III gives AUROC scores for a range of length parameter values with $N = 5$. We selected a value of length parameter = 12 since this optimized AUROC scores. We give an example representative classifier from this with a threshold of 46, which results in a detection rate of 0.8790 for our representative false positive rate of 0.01.

Experiment 3: Ngram versus skip-ngram. We implemented the ngram schemes for N from 3 to 8, and the skip-ngram schemes from Table I which extracted 3 bytes. Table IV gives AUROC scores for these ngram and skip-ngram schemes with an optimized length parameter and using BPB=2. All the ngram and kskip-ngram schemes with $N \geq 5$ have good accuracy.

Experiment 4: Attacking Ngram and skip-ngram. We have a range of ngram and skip-ngram candidates from Experiment 3, which have high accuracy. No single choice (with $N \geq 5$) stands out as preferable, so we now consider the security and performance considerations. We performed a simple attack scenario where we do simple modifications to a text file using a gradient descent algorithm. We apply 50 transformations, where at each step we select the transformation from SW modifications. The 10 possible transformations [13] were

1) insert word,
2) delete word,
3) substitute word,
4) delete 10 char,
5) insert 10 char,
6) change 10 char,
7) add a high entropy token,
8) add a low entropy token,
9) swap 2 lines, and
10) change the case of 10 char.

Table V shows the distance increase achieved by the 50 transformations. We see very different behavior between the ngram type features and the skip-ngram type features.

As N grows with the ngram type features, the methods are

Search Width	ngram4	ngram5	ngram6	ngram7	ngram8
Random SW=1	7.3	7.9	14.5	17.3	22.3
1	12.7	18.3	22.4	27.0	30.2
2	15.4	23.0	25.4	30.2	39.7
5	23.2	28.1	30.7	35.1	48.4
10	22.6	33.2	33.0	37.4	55.0
15	25.4	31.2	32.0	43.4	57.4
Search Width	skip1 -ngram4	skip2 -ngram5	skip3 -ngram6	skip4 -ngram7	skip5 -ngram8
Random SW=1	5.9	6.8	7.4	5.9	3.9
1	9.1	10.7	9.5	6.6	3.5
2	10.7	13.4	10.6	7.6	4.5
5	12.8	14.8	12.9	10.4	6.6
10	13.2	15.4	13.5	9.3	7.9
15	16.0	16.1	13.7	11.3	7.5

TABLE V
EXPERIMENT 4: RESULTS FOR ATTACKING NGRAM AND SKIP-NGRAM SCHEMES.

more vulnerable to both random attack, and intelligent attacks ($SW > 1$). The opposite effect occurs with skip-ngram type features; as N grows the methods are more resilient to attack. Here the data in this experiment, backs up our intuition and various security analyses of Sdhash and Ssdeep (methods that rely on long strings and long ngrams are easy to attack).

E. Selecting a Preferred Design

Most of the choices were fairly clear cut, except the choice of ngram versus skip-ngram. The data in Table IV provides accuracy data to compare ngram and skip-ngram. We found a set of methods where the difference in accuracy was negligible. The data in Table V suggests very strongly that skip-ngrams are significantly harder to attack. Using skip-ngrams comes with a performance cost which is shown in Table I. We therefore make the following recommendations. For security tasks,

1) If the performance of calculation of the digest is a concern, then the skip2-ngram5 is preferred. This is the TLSH method [Oliver2013]
2) If the performance of calculation of the digest is not a concern, then the skip2-ngram7 or skip2-ngram8 should be considered.

For non-security tasks

1) If the cost of computation is a factor, then ngram5-ngram8 should be considered.
2) If the cost of computation is not a factor, then all of the ngram and skip2-ngram methods with $N > 4$ should be considered.

IV. CONCLUSION

This paper has looked at criteria for developing fuzzy hashing schemes. We focus on the development of fuzzy hashing schemes that

- are accurate,
- have fast run-time performance,
- will operate in adversarial environments, and
- will work for a wide range of file types.

The two primary choices for feature types are ngrams and skip-ngrams.

We find that LSH schemes using ngrams and skip-ngrams have comparable accuracy. However, the skip-ngrams are far more resilient when attacked. The ngram schemes have the same run-time performance with each other. The run time performance of the skip-ngram schemes quickly degrades as the sliding window size, N, grows. We highlighted one of the design choices available, which was a skip-ngram scheme with $K = 2$ and $N = 5$.

The various approaches described here has been released as the open source TLSH [1], [21]. The default scheme in that software is the skip-ngram scheme with $K = 2$ and $N = 5$, which exhibits good accuracy, fast run-time performance, resilience to attack and works for a wide range of file types. We would also consider the evaluation of other more complicated skip-ngram schemes (higher K, higher N) for problem domains which require resilience to adversarial attack, and have lower run-time performance requirements.

REFERENCES

[1] J. Oliver, C. Cheng, and Y. Chen, "Tlsh–a locality sensitive hash," in *2013 Fourth Cybercrime and Trustworthy Computing Workshop*. IEEE, 2013, pp. 7–13.

[2] J. Oliver and J. Pryde, "Smart Whitelisting Using Locality Sensitive Hashing," https://blog.trendmicro.com/trendlabs-security-intelligence/smart-whitelisting-using-locality-sensitive-hashing/, 2017, [Online; accessed 29-Feb-2020].

[3] J. Coffman, A. Chakravarty, J. A. Russo, and A. S. Gearhart, "Quantifying the effectiveness of software diversity using near-duplicate detection algorithms," in *Proceedings of the 5th ACM Workshop on Moving Target Defense*, 2018, pp. 1–10.

[4] J. Oliver, "How Machine Learning Techniques Helped us Find Massive Certificate Abuse by BrowseFox," https://blog.trendmicro.com/trendlabs-security-intelligence/how-machine-learning-techniques-helped-us-find-massive-certificate-abuse-by-browsefox/, 2018, [Online; accessed 11-Mar-2021].

[5] K. E. Cybersecurity, "Machine learning for malware detection ," https://pdfs.semanticscholar.org/f63e/7ebb30e87d0bf60bed 138db9d23d059 8d5e5.pdf.

[6] J. Oliver, M. Ali, and J. Hagen, "Hac-t and fast search for similarity in security," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, pp. 1–7.

[7] M. Ali, J. Hagen, and J. Oliver, "Scalable malware clustering using multi-stage tree parallelization," in *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2020, pp. 1–6.

[8] J. Oliver and M. Osena, "Cluster of Coins: How Machine Learning Detects Cryptocurrency-mining Malware," https://blog.trendmicro.com/trendlabs-security-intelligence/cluster-of-coins-how-machine-learning-detects-cryptocurrency-mining-malware/, 2018, [Online; accessed 11-Mar-2021].
[9] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital investigation*, vol. 3, pp. 91–97, 2006.
[10] V. Roussev, "Data fingerprinting with similarity digests," in *IFIP International Conference on Digital Forensics*. Springer, 2010, pp. 207–226.
[11] F. Breitinger, B. Guttman, M. McCarrin, and V. Roussev, "Approximate matching: definition and terminology," 2014.
[12] J. Oliver, "On criteria for evaluating similarity digest schemes," https://github.com/trendmicro/tlsh/blob/master/2015_Euro_DFRWS_Criteria_Sim_Digests.pdf, 2015, presented at DFRWS Europe 2015.
[13] J. Oliver, S. Forman, and C. Cheng, "Using randomization to attack similarity digests," in *International Conference on Applications and Techniques in Information Security*. Springer, 2014, pp. 199–210.
[14] H. Baier and F. Breitinger, "Security aspects of piecewise hashing in computer forensics," in *2011 Sixth International Conference on IT Security Incident Management and IT Forensics*. IEEE, 2011, pp. 21–36.
[15] A. Kurakin, I. Goodfellow, S. Bengio, Y. Dong, F. Liao, M. Liang, T. Pang, J. Zhu, X. Hu, C. Xie *et al.*, "Adversarial attacks and defences competition," in *The NIPS'17 Competition: Building Intelligent Systems*. Springer, 2018, pp. 195–231.
[16] "Locality sensitive hashing," https://en.wikipedia.org/wiki/Locality-sensitive_hashing, [Online; accessed 24-Mar-2020].
[17] "Skip-gram," https://en.wikipedia.org/wiki/N-gram#Skip-gram, [Online; accessed 24-Mar-2020].
[18] P. K. Pearson, "Fast hashing of variable-length text strings," *Communications of the ACM*, vol. 33, no. 6, pp. 677–680, 1990.
[19] "Nilsimsa hash," https://en.wikipedia.org/wiki/Nilsimsa_Hash, [Online; accessed 11-Mar-2021].
[20] "Autoencoder," https://en.wikipedia.org/wiki/Autoencoder, [Online; accessed 24-Mar-2020].
[21] Github, "Tlsh software," https://github.com/trendmicro/tlsh/.