Using Randomization to Attack Similarity Digests

Jonathan Oliver, Scott Forman
and Chun Cheng

Jonathan Oliver
Trend Micro, Melbourne, Australia
jon_oliver@trendmicro.com

Scott Forman
Trend Micro, Melbourne, Australia
lane_forman@trendmicro.com

Chun Cheng
Trend Micro, Melbourne, Australia
chun_cheng@trendmicro.com

Contents

4
Introduction

6
Description of Ssdeep, Sdhash and TLSH

8
Analyzing Spam Image Files

11
Analyzing Text Files and Web Pages

16
Analyzing Executable Files

21
Conclusion

Abstract

There has been considerable research and use of similarity digests and Locality Sensitive Hashing (LSH) schemes - those hashing schemes where small changes in a file result in small changes in the digest. These schemes are useful in security and forensic applications. We examine how well three similarity digest schemes (Ssdeep, Sdhash and TLSH) work when exposed to random change. Various file types are tested by randomly manipulating source code, Html, text and executable files. In addition, we test for similarities in modified image files that were generated by cybercriminals to defeat fuzzy hashing schemes (spam images). The experiments expose shortcomings in the Sdhash and Ssdeep schemes that can be exploited in straight forward ways. The results suggest that the TLSH scheme is more robust to the attacks and random changes considered.

Introduction

Similarity digest schemes exhibit the property that small changes to the file being hashed results in a small change to the hash. The similarity between two files can be determined by comparing the digests of the original files.

We considered the following schemes: Ssdeep [6], Sdhash [9], and TLSH [7]. We restricted the research to these schemes because they had mature implementations which were available as open source code. In addition, Ssdeep [6] is the de facto standard in the area of malware analysis. It is currently supported by NIST [12], and is the only similarity digest supported by Virus Total [16]. We did not report on the Nilsimsa [11] scheme here due to its high collision rate and false positive rate [7].

There have been several security analyses of similarity digests [2, 3, 8]. In [2], Breitinger analyzed Ssdeep and concluded that Ssdeep "is not suitable as a 'cryptographic similarity hashing function'. There are vulnerabilities that are easily exploitable". Roussev [8] concludes that Sdhash demonstrated the potential to address all five of the design requirements, where the design requirements were reasonable security requirements for similarity digests. Breitinger et al. [3] conclude that "Sdhash has the potential to be a robust similarity preserving digest algorithm".

An important property to consider for similarity digests [2, 3] is anti-blacklisting. Anti-blacklisting involves modifying a file to be semantically similar, but where a digest method assesses the files to be non-similar.

We have no expectation for similarity digests to match files which use an encrypted file format. For example, executable code which has been encrypted as a part of a packing process is not considered "semantically similar" to the original executable code for the purpose of this paper. Typical ways that files are modified include:

- Spam email: It is standard practice for spammers to use templates for their spam and to add randomized content to each individual message;

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also "TLSH - A Locality Sensitive Hash" CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

- Source code: It is not uncommon for the whitespace in source code to be changed by programmers, program beautifiers or editors;
- Malware: Malware uses techniques such as packing, polymorphism and metamorphism [5] to make the executable code more difficult to analyze. In this paper, we do not consider the packing issue, but we consider elements of polymorphism/metamorphism such as adding NOPs, permuting registers, adding useless instructions and loops, function re-ordering, program flow modification and inserting un-used data [5].

We focus on situations where the file is deliberately modified by an adversary using randomization as a key component. This paper offers the following new aspects to the research area:

- we provide simple rules for modifying content to make Ssdeep ineffective,
- we reject the proposal in [3] that Sdhash is a robust similarity digest, and provide simple rules for modifying content to make Sdhash ineffective, and
- provide evidence that locality sensitive hashing schemes (such as TLSH) scheme are more difficult to exploit.

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also “TLSH - A Locality Sensitive Hash” CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

A Description of Ssdeep, Sdhash and TLSH

Ssdeep [6] uses 3 steps to construct the digest from file F:

1. use a rolling hash to split the document into distinct segments;
2. produce a 6 bit value for each segment by hashing the segment; and
3. concatenate the base64 encoded values from step (2) to form the signature. Ssdeep assigns a similarity score in the range of 0-100 by calculating the edit distance between the two digests using the dynamic programming algorithm.

Ssdeep is vulnerable to anti-blacklisting in two ways [2]:

- to disrupt the content identified by the rolling hash, and
- to modify content in all the segments. Because of these vulnerabilities, Breitinger [2] concludes that Ssdeep is insecure.

Sdhash [9] uses 3 steps to construct the digest:

1. identify 64 byte sequences which have a low probability;
2. hash the sequences identified in step (1) and put them in a Bloom filters; and
3. encode the series of Bloom filters to form the output signature.

Sdhash assigns a similarity score in the range 0-100 by calculating a normalized entropy measure between the two digests.

A security assessment of Sdhash is made in [3]. In [3], the authors state that the main contribution of the paper is that “Sdhash is a robust approach, but an active adversary can beat down the similarity score

---

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also “TLSH - A Locality Sensitive Hash” CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

to approximately 28 while preserving the perceptual behavior of a file". Breitinger et al. (Section 5.1 of [3]) note that 20% of the input bytes do not influence the similarity digest, giving scope for attack.

TLSH [7] is a locality sensitive hash closer in spirit to the Nilsimsa [11] hash than the Ssdeep and Sdhash digests. TLSH uses 4 steps to construct the digest:

1. process the input using a sliding window to populate an array of bucket counts;
2. calculate the quartile points;
3. construct the digest header values based on the quartile points, the length of the file and a checksum; and
4. construct the digest body by generating a sequence of bit pairs, which depend on each bucket's value in relation to the quartile points.

TLSH assigns a distance score between two digests by summing the distance between the digest headers and the digest bodies. The distance between the digest bodies is calculated as an approximate Hamming distance between the two digest bodies. The distance between two digest headers is determined by comparing file lengths and quartile ratios. The distance score between two digests is in the range 0-1000+. The recommended threshold [7] is 100, which should be tuned for each application.

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also "TLSH - A Locality Sensitive Hash" CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

Analyzing Spam Image Files

We collected a sample of 1000 images which had been deliberately manipulated by spammers to avoid detection. There were 30 distinct groups of related spam images. In 23 of these groups, the spammers had systematically manipulated the images so that the image files were distinct, leaving us with a data set of 911 images. Examples of the types of manipulations are shown in Figure 1 below. The manipulations included changing the height and width, changing the font size, doing rotations of the images, adding dots and dashes to the images, and changing the background colours.

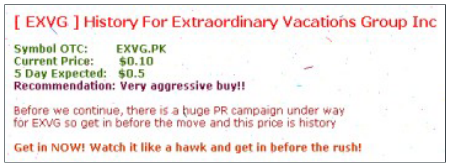
Manipulation	Example Image #1	Example Image #2
Image rotation		
Changing image dimensions; stretching image.	 Dimensions = 134 x 71	 Dimensions = 123 x 73
Changing image height and width; Changing font and changing font size.		

Figure 1. Example spam images

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also "TLSH - A Locality Sensitive Hash" CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

Due to the processes used to compress jpeg and gif images, it is not a useful experiment to apply the similarity digests to the raw gif and jpeg images. So CxImage [10] was used to extract the image and save the file as a bit mapped image. The digest methods were then used on each group to determine detection rates and across distinct groups to determine a false positive rate for each of the methods.

Tables 1 shows the detection rates for each digest scheme. The Sdhash and Ssdeep methods were considered to match images if they scored any value above 0. The threshold for the TLSH digest was selected to be 100. With these thresholds, Ssdeep and Sdhash had no false positive matches, and TLSH had a false positive rate of 0.007% (29 out of 414505 image combinations). The results in Table 1, show that

- Ssdeep was ineffective at identifying images as being related, although it did have a very low false positive rate.
- The TLSH and Sdhash methods were reasonably effective at identifying that images are related, for many of the other classes of image manipulation.
- The digest methods were ineffective at certain types of adversarial image manipulations. The groups that digest methods were ineffective against included the groups where multiple types of changes were made (Pharmacy erectile dysfunction, Stockspam CYTV, Stockspam EXVG).
- TLSH was able to identify images that were rotated, while Sdhash was not able to do so (see the "Discounted Pharma" images in Figure 1).
- TLSH was able to identify images that were stretched, while Sdhash was not able to do so (see the "Pharmacy Picture" images in Figure 1).

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also "TLSH - A Locality Sensitive Hash" CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

Image Group	N	TLSH	Sdhash	Ssdeep
Discounted Pharma	20	80.0%	3.7%	0.0%
International Greek	3	33.3%	33.3%	0.0%
Pharmacy erectile dysfunction	147	22.1%	22.6%	9.6%
Pharmacy legal RX	22	0.0%	0.0%	0.0%
Pharmacy online 1	22	90.5%	100.0%	10.8%
Pharmacy online 2	63	12.1%	11.2%	1.0%
Pharmacy online 3	10	64.4%	62.2%	4.4%
Pharmacy online 4	6	100.0%	100.0%	6.7%
Pharmacy picture	8	57.1%	3.6%	7.1%
Pharmacy pop a pill	5	80.0%	100.0%	60.0%
Pharmacy power pack	41	47.8%	47.8%	20.7%
Pharmacy research	3	0.0%	33.3%	33.3%
Pharmacy Viagra Pro	11	32.7%	38.2%	29.1%
Pharmacy Viagra Pro2	7	42.9%	42.9%	42.9%
Software OEM	6	66.7%	66.7%	66.7%
Software SOBAKA	11	100.0%	100.0%	100.0%
StockSpam CYTV	105	1.7%	1.4%	0.0%
StockSpam EXVG	389	1.2%	2.8%	0.6%

Table 1. Detection rates for each group of images

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also “TLSH - A Locality Sensitive Hash” CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

Analyzing Text Files and Web Pages

In the case of image files, we had real world data where images had been altered to try to stop a filter from determining that they were similar. For text and Html files, we randomly made changes to them to simulate the adversarial environment.

Performing Random Changes

Procedure "`greedy_adversarial_search`" takes two inputs a file F(0) and a digest scheme DS. At iteration n, it considers "random changes" to F(n-1), and creates F(n) by applying the change that results in the lowest score according to digest scheme DS. This creates a sequence of files F(0) … F(n) where for each i > j, score(F(i), F(0)) < score(F(j), F(0)) according to scheme DS. It will perform these changes until F(n) is considered a non-match or up to 500 iterations. In the case of the TLSH scheme, the scores of the sequence are increasing rather than decreasing. We define a single "random change" as one of the following actions:

1. nsert a new word (selected randomly);
2. delete an existing word (selected randomly);
3. swap two words (each word selected randomly from within the document);
4. substitute a word for another word (each word selected randomly) ;
5. replace 10 occurrences of a character with another character; VI. delete 10 occurrences of a character;
6. swap two lines (selected randomly)
7. append a low entropy token of length 10 at the end of the document (a single
8. character is selected randomly) (for example append "1111111111"); and

---

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also "TLSH - A Locality Sensitive Hash" CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

9. append a high entropy token of length 10 at the end of the document (for example append the token "Qo*\ezN)8$").

We used the procedure on 500 text and html files to identify vulnerabilities in the digest methods. Across the sample of files we measure the following:

- File Size: the size of original files in bytes,
- Number of Files Broken: the number of files where greedy procedure returned successfully (i.e, the greedy procedure was successful at defeating the digest within 500 iterations),
- Iteration Required to Break Digest: When the greedy procedure ends in success, we record the iteration number,
- Relative File Change: This was measured by comparing the original file with the manipulated file at the final iteration of the greedy procedure. The comparison is made by converting the original and final manipulated file into two sorted lists of tokens (by replacing all sequences of whitespace by a newline character) and using the Linux "diff" command to determine the ratio of tokens that have changed to the original number of tokens in the file.
- List of Random Changes: The sequence of changes performed by the greedy procedure.

Table 2 gives the results of applying the greedy procedure to the 500 text and Html files. The table splits the results into 5 file size ranges, and for each range gives the average results for the criteria measured.

File Size	Average Relative Change	Digest Attacked	% Broken	Average Iterations to Break Digest
0-10000	34.3%	TLSH	20.6%	83.7
0-10000	34.3%	Ssdeep	100.0%	6.9
0-10000	34.3%	Sdhash	100.0%	14.5
10000-20000	21.8%	TLSH	12.7%	84.5
10000-20000	21.8%	Ssdeep	100.0%	7.1
10000-20000	21.8%	Sdhash	100.0%	26.3
20000-40000	14.4%	TLSH	2.8%	78.7
20000-40000	14.4%	Ssdeep	100.0%	7.9
20000-40000	14.4%	Sdhash	97.2%	44.9
40000-80000	10.4%	TLSH	0.0%	
40000-80000	10.4%	Ssdeep	100.0%	10.3

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also "TLSH - A Locality Sensitive Hash" CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

File Size	Average Relative Change	Digest Attacked	% Broken	Average Iterations to Break Digest
40000-80000	10.4%	Sdhash	32.9%	68.0
80000-	7.9%	TLSH	0.0%	
80000-	7.9%	Ssdeep	96.9%	1.4
80000-	7.9%	Sdhash	0.0%	

Table 2. Results after applying the greedy procedure to 500 text and Html files

The greedy procedure was highly successful at breaking the Ssdeep and Sdhash digests when the file size was below 40,000 bytes. The Ssdeep digest method was particularly vulnerable - on average being broken with less than 10 iterations. The difference in the robustness of the digest approaches to adversarial manipulation is highlighted with file sizes in the range 20,000-40,000; in this range manipulating an average of 14% of the original file will break Ssdeep and Sdhash most of the time, while the TLSH scheme is still able to identify the files as being related files.

The Ssdeep method was consistently broken by procedure greedy_adversarial_search. The random changes selected most frequently by the procedure were the swap-line, change-char and delete-char modifications. The characters selected the most often to be changed or deleted were ‘S’, ‘N’, newline, space.

This is particularly disturbing for the Ssdeep method since the changes which are very effective at breaking the digest method are those that humans are unlikely to notice, such as changing the spacing and the line length.

The Sdhash method was also consistently broken by procedure greedy_adversarial_search, though on average Sdhash required 25 more iterations to break than Ssdeep. The random changes selected most frequently by the procedure were the change-char, delete-char and swap-line modifications. The characters selected the most often to be changed or deleted were: ‘c’, ‘d’, ‘u’, ‘r’, ‘e’, ‘m’, newline, ‘f’, comma, ‘S’.

This is also disturbing for the Sdhash method - some changes which are very effective at breaking the digest method include those that do not change the meaning of the document - namely changing the length of lines in a document.

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also “TLSH - A Locality Sensitive Hash” CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

Anti-blacklisting for C/C++ Source Code

Task (1) is to modify source code in such a way that:

- The modified source code still compiled and produced an executable program identical to the original source code, and
- Each modified file of source code resulted in a similarity digest which was judged to not match the digest of the original source file.

This could be achieved with the sed [1] script: “s/;[\t]*$/& / s/{[\t]*$/& /”.

This sed script adds a space after each semicolon (;) and open brace ({) at the end of lines. We note that there is a multitude of ways that further changes can be made before we start to consider the types of program transformations which do not alter the meaning of the program, but change its representation. We tried it on a range of source code projects, and found the script to be 100% effective at breaking both Ssdeep and Sdhash.

Anti-blacklisting for Html files

Task (2) is to modify Html files in such a way that:

- The modified Html file had the same appearance and browser functionality to the original Html file, and
- The modified Html file had a similarity digest which was judged to not match the digest of the original Html file.

This could be achieved with the sed [1] script:

```
s/<([a-zA-Z0-9]*)>/<\1>/g
s/>[\t]/&/gs/,[\t]/&/g
s/[;}>][\t]*$/&/
s/[a-zA-Z0-9]$/&/
```

The intent of the script is to

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also “TLSH - A Locality Sensitive Hash” CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

exploit the following features of Html:

- It is permissible to put additional whitespace to further separate attributes inside Html tags [13], (lines 1-4).
- It is permissible to put an additional whitespace after end tags and commas in the text in the Html page will result in an identical output page being displayed, (lines 5-6).
- It is permissible to put an additional whitespace at the end of lines where the last token is an end tag or a word, which will result in an identical output displayed (lines 7-8).

We applied the technique to 500 HTML files and got the following results:

Digest Method	Number of manipulated Html files identified as matching original file
TLSH	291
Sdhash	16
Ssdeep	11

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also “TLSH - A Locality Sensitive Hash” CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

Analyzing Executable Files

We expect that similarity digests will behave differently when applied to executable files than when applied to image files, text files and html files. The reason for this is that text files and image files have no requirement to share common components. However, we fully expect executable files to share standard components. For example we expect C and C++ programs to share components such as the stdio library and the preparation of the argc and argv parameters to the main() function. Thus we need to establish a baseline threshold for each similarity digest scheme. In Section 5.1, we determine suitable thresholds for the digest schemes for Linux executable programs. We use these thresholds in Section 5.2 in our efforts to break the digest schemes.

Suitable Thresholds for Linux Executable Files

We analyzed the binary files from /usr/bin of a standard Linux distribution. There are 2526 files in /usr/bin, and we removed all those files which were either symbolic links or less than 512 bytes (since the Sdhash scheme requires a minimum of 512 bytes to create a digest). This left 1975 executable files. We applied the similarity digest schemes doing 1975 * 1974 / 2 = 1,949,325 file comparisons. We begin this analysis using the tentative thresholds of <= 100 for TLSH, and >= 1 for Sdhash and Ssdeep. Using these thresholds resulted in the following number of file matches:

Digest	Number of matches
TLSH ≤ 100	35,733
Sdhash ≥ 1	25,408
Ssdeep ≥ 1	836

Manual inspection of the files showed that:

- A threshold of 100 was not useful for TLSH – it was making many unjustified matches near the threshold of 100 – for example matching “time” and “xtrapchar”.

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also “TLSH - A Locality Sensitive Hash” CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

- A threshold of 1 was not useful for Sdhash – it was making many unjustified matches near the threshold of 1 – for example matching “ap2” and “xkill”.
- A threshold of 1 was appropriate for Ssdeep.

To improve the thresholds for Sdhash and TLSH, we consider thresholds where there is similar discriminatory power. We found the thresholds which where closest to assigning 1 in 1000 and 1 in 100 of the possible 1,949,325 file combinations as matching:

	Threshold	Number of matches
1 in 1000	Sdhash ≥ 13	2,215
1 in 1000	TLSH ≤ 52	2,130
1 in 100	Sdhash ≥ 2	19,029
1 in 100	TLSH ≤ 85	19,307

We found that for the thresholds of 13 for Sdhash and of 52 for TLSH, file pairs near the thresholds are very likely to be related executable files. For the thresholds of 2 for Sdhash and of 85 for TLSH, file pairs near the thresholds are almost always unrelated executable files.

Based on this, we will take a conservative approach and use a threshold of 2 for Sdhash and 85 for TLSH as the basis of anti-blacklisting testing. By this, we mean that if an executable program can be modified (while keeping its functionality the same) in a way which causes the TLSH distance between the original and modified program to be >= 86, then we have broken the digest scheme.

Anti-blacklisting for Executable Programs

Task (3) is to modify an executable program in such a way that:

- The modified source code still compiled and produced identical program behavior (determined by finding no difference on various output runs), and
- The modified executable program had a similarity digest which was judged to not match the digest of the original program.

To achieve this, we performed modifications to the source code and applied the digest methods to the executable program created by compiling the source code. Each change considered was designed to leave unchanged the semantic meaning of the program, while creating small changes in the object code. The semantic meaning of the code was verified using unit-test programs. The changes introduced to the

---

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also “TLSH - A Locality Sensitive Hash” CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

source code, were typical of the changes performed by polymorphic malware and metamorphic malware [6]. The changes implemented are given in Table 3.

Modification	Description
And-Reordering	Changing the order of clauses in an "if" statement if the condition is a conjunction
Or-Reordering	Changing the order of terms in an "if" statement if the condition is a disjunction
Control-Flow-If-Then-Else	Change control flow of an if-then-else statement
Control-Flow-If-Then	Change control flow of an if-then statement
New Integer Variables	Introducing new integer variables
New String Variables	Introducing new string variables
Re-ordering Functions	Changing the order of functions within the source code
Adding NOPs	Adding variables definitions and adding NOPs related to those variables.
Adding Random Binary Data	Adding character strings with randomized content.
Splitting Strings	Split the control string within printf statements

Table 3. 10 Modifications for source code

We applied these changes to 3 programs:

- C4.5 [4],
- SVMlight [14],
- greedy_adversarial_search (the program from Section 4.1)

We applied the modifications listed to each source file in turn. Some of the modifications were not applicable to some source files, and some of the modifications could cause syntactic or semantic errors. Where this occurred the modification was discarded.

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also "TLSH - A Locality Sensitive Hash" CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

	Number of source files modified	TLSH	Sdhash	Ssdeep
And-Reordering	5	13	28	32
Or-Reordering	5	26	25	0
If-Then-Else	9	13	46	27
If-Then	12	9	81	69
New Integer Variables	6	12	35	30
Reorder Funs	1	9	79	71
Add NOPs	4	13	16	29
Add Random Data	3	11	70	60
Split Strings	1	13	24	33
New String Variables	14	62	1	0

Table 4. Scores after a single modification on the C4.5 source code

Table 4 gives the scores of the various digests schemes when we apply a single manipulation from Table 3 to the source code of C4.5 [4]. The column "Number of source files modified" is the number of source code files that the manipulation is applicable to and produces no errors. For the C4.5 source code, applying a single type of manipulation broke both the Sdhash and the Ssdeep digest schemes.

We applied the same approach to SVMlight:

- 5 of the manipulations reduced the Ssdeep score to 0.
- The "New String Variables" manipulation reduced the Sdhash score to 0 and increased the TLSH score to 50.

Applying the "New String Variables" manipulation followed by the "And-Reordering" manipulation increased the TLSH score to 34 and reduced the Sdhash and Ssdeep scores to 0.

We applied the same approach to greedy_adversarial_search:

- Again 5 of the manipulations reduced the Ssdeep score to 0 (it was a different set of 5 manipulations).
- The "Add NOPs" manipulation reduced the Sdhash score to 2.

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also "TLSH - A Locality Sensitive Hash" CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

- The “New String Variables” manipulation increased the TLSH score to 23.
- The “Add NOPs” manipulation reduced the Sdhash score to 2.

Applying the “New String Variables” manipulation followed by the “New Integer Variables” manipulation increased the TLSH score to 38 and reduced the Sdhash and Ssdeep scores to 0.

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also “TLSH - A Locality Sensitive Hash” CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

Conclusion

Research into similarity digests and locality sensitive hashes for security applications should be done in an adversarial environment, where the people developing the digest schemes are actively trying to break their own work and the work of other such schemes.

Our work demonstrated that different types of manipulations can have very distinct effects on the scores of similarity digests. Researchers should also explore the manipulations which are mostly likely to adversely affect the scheme.

Different thresholds need to be considered for different file types. The experiments described in this paper show that executable files appear to be a more difficult discrimination task for similarity digests than Html, text files and images, requiring careful selection of suitable thresholds.

Our work also demonstrates that similarity digests should not use a restricted range, such as 0 to 100. This gives adversaries a target to strive for; once a Sdhash or Ssdeep digest has been reduced to zero, then these schemes cannot adjust their threshold any further. An open ended distance criteria makes the job of an adversary more difficult.

Based on the analysis in this paper, we make the following conclusions:

- Ssdeep: We concur with the previous assessments [2, 8] that Ssdeep is not suitable as a 'secure similarity digest'.
- Sdhash: We disagree with the security assessment in [3] that "Sdhash is a robust approach". Sdhash has significant vulnerabilities that can be exploited.
- TLSH: Based on the experiments done here, TLSH appears significantly more robust to random changes and adversarial manipulations than Ssdeep and Sdhash.

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also "TLSH - A Locality Sensitive Hash" CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

REFERENCES

1. Barnett, B.: Sed - An Introduction and Tutorial, http://www.grymoire.com/Unix/Sed.html.
2. Breitinger, F.: Sicherheitsaspekte von fuzzy-hashing. Master's thesis, Hochschule Darmstadt, 2011
3. Breitinger, F., Baier, H., Beckingham, J.: Security and Implementation Analysis of the Similarity Digest sdhash, 1st International Baltic Conference on Network Security & Forensics (NeSeFo), Tartu (Estland) (2012).
4. C4.5 source code http://www.rulequest.com/Personal/.
5. Hosmer, C.: Metamorphic and Polymorphic Malware, Black Hat USA, 2008, http://blackhat.com/presentations/bh-usa-08/Hosmer/BH_US_08_Hosmer_Polymorphic_Malware.pdf.
6. Kornblum, J.: Identifying Almost Identical Files Using Context Triggered Piecewise Hashing. In: Proceedings of the 6th Annual DFRWS, pp. S91.S97. Elsevier, (2006).
7. Oliver, J., Cheng, C., Chen, Y.: TLSH - A Locality Sensitive Hash. 4th Cybercrime and Trustworthy Computing Workshop, Sydney, November 2013 https://www.academia.edu/7833902/TLSH_-A_Locality_Sensitive_Hash.
8. Roussev, V.: An Evaluation of Forensics Similarity Hashes. In: Proceedings of the 11th Annual DFRWS, pp. S34.S41. Elsevier, (2011).
9. Roussev, V.: Data Fingerprinting with Similarity Digests. In: Chow, K.; Shenoi, S. (eds.) Research Advances in Digital Forensics VI, pp. 207--226. Springer (2010)
10. CxImage, http://www.codeproject.com/Articles/1300/CxImage.
11. Nilsimsa source code, https://web.archive.org/web/20150512025912/http://ixazon.dynip.com/~cmeclax/nilsimsa.html.
12. NIST, http://www.nsrl.nist.gov/ssdeep.htm.
13. Stackoverflow Blog, White space inside XML/HTML tags, http://stackoverflow.com/questions/3314535/white-space-inside-xml-html-tags.
14. SVMlight source code http://svmlight.joachims.org/.
15. TLSH source code https://github.com/trendmicro/tlsh.
16. Virus Total, http://www.virustotal.org/.

Available at
https://github.com/trendmicro/tlsh/blob/master/Attacking_LSH_and_Sim_Dig.pdf

See Also "TLSH - A Locality Sensitive Hash" CTC 2013
https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf

TREND MICRO™

Trend Micro Incorporated, a global cloud security leader, creates a world safe for exchanging digital information with its Internet content security and threat management solutions for businesses and consumers. A pioneer in server security with over 20 years experience, we deliver top-ranked client, server, and cloud-based security that fits our customers' and partners' needs; stops new threats faster; and protects data in physical, virtualized, and cloud environments. Powered by the Trend Micro™ Smart Protection Network™ infrastructure, our industry-leading cloud-computing security technology, products and services stop threats where they emerge, on the Internet, and are supported by 1,000+ threat intelligence experts around the globe. For additional information, visit **www.trendmicro.com**.

Securing Your Journey
to the Cloud