



The Trend Locality Sensitive Hash: TLSH

Jon Oliver, Chun Cheng, Yanggui Chen

21st May 2013

Contact: tlsh@trendmicro.com

Getting TLSH

- Contact us at
tlsh@trendmicro.com
- Source Code:
<https://github.com/trendmicro/tlsh.git>

What is Locality Sensitive Hashing

- Traditional hashes (such as SHA1 and MD5) have the property that a small change to the file being hashed results in a completely different hash
- Locality Sensitive Hashes (LSH) have the property that a small change to the file being hashed results in a small change to the hash
 - You can measure the similarity between 2 files by comparing their LSH values

Example Locality Sensitive Hashing



Text 1 – Chapter 1 of Pride and Prejudice

It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife.

..

When she was discontented, she fancied herself nervous.
The business of her life was to get her daughters married; its solace was visiting and news.

Text 2 - Chapter 1 of Pride and Prejudice with last line removed

It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife.

..

When she was discontented, she fancied herself nervous.

Example Locality Sensitive Hashing

TLSH

| | |
|--------|--|
| Text 1 | E491A51FA380022245B021E9770F3A6FF706C1780365C631581EF6263731EAA87F96EE |
| Text 2 | 5B91940FA380026245B021A9771F7A6FF706C1780765C671981EF6263731EAA87F96DE |

MD5_HASH

| | |
|--------|----------------------------------|
| Text 1 | 3b9dd1f86ce0c3b467055b48f9a5221c |
| Text 2 | 7dc8267c6bea14d36df64934aad4604f |

SHA1_HASH

| | |
|--------|--|
| Text 1 | 8b8c6ce1253515a1fbceaec0f5cfc58780e6fd5e |
| Text 2 | e494d7fa7b4080520c59a6702764983ff9b6d399 |

The MD5 and SHA1 hashes are completely different

For these 2 pieces of text, the TLSH values are quite similar Hashes

Example Locality Sensitive Hashing

TLSH

| | |
|--------|--|
| Text 1 | E491A51FA380022245B021E9770F3A6FF706C1780365C631581EF6263731EAA87F96EE |
| Text 2 | 5B91940FA380026245B021A9771F7A6FF706C1780765C671981EF6263731EAA87F96DE |

The distance between Text 1 and Text 2

$\text{distance}(\text{Text1}, \text{Text2}) = 11$

Distance scores can go up to 1000 and above

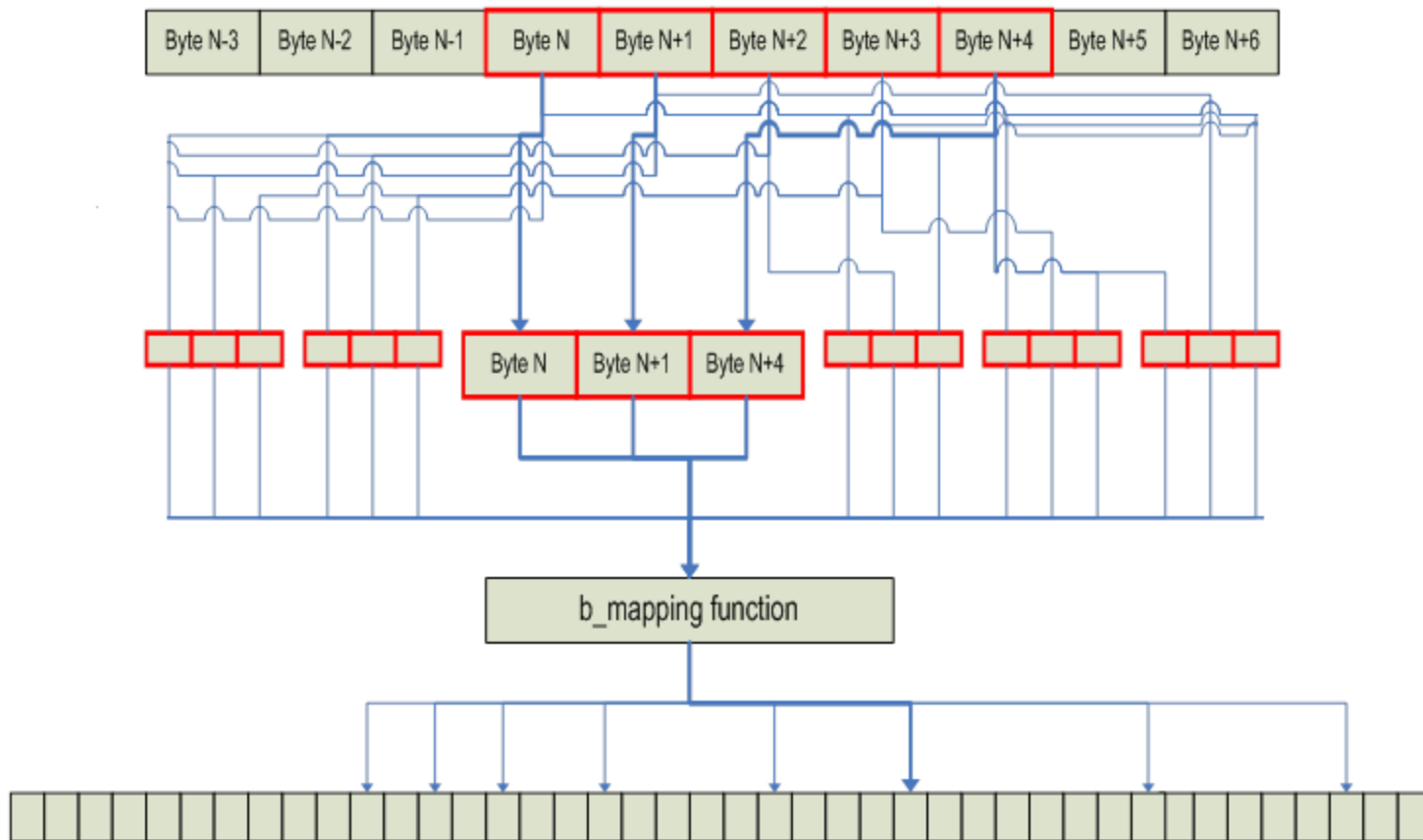
A low score (of 50 or less) means that the files are quite similar

You will need to determine an appropriate threshold for your application

A distance of 0 means that the files are (very likely) to be exactly the same

Just like the MD5 and SHA1 schemes, collisions can occur and very different files will have the same hash value.

Algorithm to determine TLSH

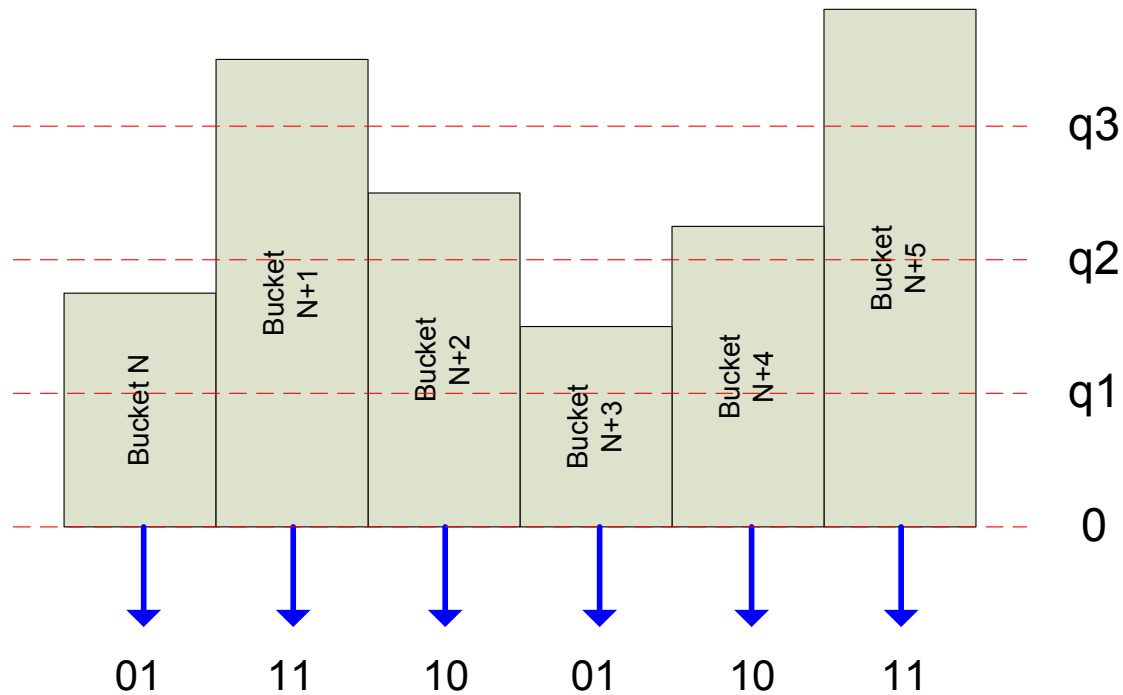


Algorithm to determine TLSH

- We use the Pearson hash [reference 1] as the mapping function between the trigrams from a window to the buckets.

Algorithm to determine TLSH

- TLSH uses a 4-way to reflect the differences between different histograms
- The q2 point is at the median bucket count
- The q1 are the lower and higher quartiles respectively



Algorithm to determine the hash

- Introduce three head bytes (6 hexadecimal characters) to preserve this information.
- The hexadecimal representation of the hash is
 - $H[0]H[1] \rightarrow$ checksum
 - $H[2]H[3] \rightarrow$ L value
 - $H[4] \rightarrow$ Q1 ratio
 - $H[5] \rightarrow$ Q2 ratio
 - $H[6] .. H[69]$ the binary representations of the 128 buckets (using the method from the previous slide) turned into hex characters

Algorithm to determine the hash

- The L value
- The input for the L value is the length of the original document (len)

```
If len <= 656
    i ← log(len)/log(1.5)
else
    if len <= 3199
        i ← log(len)/log(1.3) – 8.72777
    else
        i ← log(len)/log(1.1) – 62.5472
i ← i MOD 256
return i
```

Algorithm to determine TLSH

- The Q ratio values

$q1 \leftarrow$ the 32nd smallest number in bucket[0..127]

$q2 \leftarrow$ the 64th smallest number in bucket[0..127]

$q3 \leftarrow$ the 96th smallest number in bucket[0..127]

$q1_ratio \leftarrow (q1 * 100 / q3) \text{ MOD } 16$

$q2_ratio \leftarrow (q2 * 100 / q3) \text{ MOD } 16$

Calculating the distance between 2 hashes

Define a $\text{mod_diff}(X, Y, R)$ function between two values, X and Y , according to a range R .

X and Y are values in the range $[0, .. R-1]$

Calculate the distance between X and Y in 2 ways

i. the difference between X and Y

ii. the difference between X and Y if you go up to $R-1$ and then back to 0

The mod_diff value is the minimum of (i) and (ii)

examples:

$\text{mod_diff}(3, 4, 16) = 1$

$\text{mod_diff}(3, 10, 16) = 7$

$\text{mod_diff}(3, 15, 16) = 4$

Calculating the distance between 2 hashes

```
// Input: t1 and t2
Const RANGE_LVALUE = 256
Const RANGE_QRATIO = 16

diff ← 0
ldiff ← mod_diff(t1.lvalue, t2.lvalue, RANGE_LVALUE);
If ldiff ≤ 1
    diff ← diff + ldiff
else
    diff ← diff + ldiff * 12;

q1diff ← mod_diff(t1.q1ratio, t2.q1ratio, RANGE_QRATIO);
If q1diff ≤ 1
    diff ← diff + q1diff
else
    diff ← diff + (q1diff-1) * 12;

q2diff ← mod_diff(t1.q2ratio, t2.q2ratio, RANGE_QRATIO);
If q2diff ≤ 1
    diff ← diff + q2diff
else
    diff ← diff + (q2diff-1) * 12;
```

Calculating the distance between 2 hashes (cont.)

```
If t1.checksum <> t2.checksum
    diff ← diff + 1

for i ← 0 to 31 {
    decode t1.H[i+6] in 4 binary values b10 b11 b12 b13
    decode t2.H[i+6] in 4 binary values b20 b21 b22 b23
    if (b10,b11) != b(20,21) {
        if (b10,b11) == (1,1) AND (b20,b21) == (0,0) diff = diff + 6
        else if (b10,b11) == (0,0) AND (b20,b21) == (1,1) diff = diff + 6
        else if (b10,b11) == (1,1) AND (b20,b21) == (0,1) diff = diff + 2
        else if (b10,b11) == (0,0) AND (b20,b21) == (1,0) diff = diff + 2
        else diff = diff + 1
    }
    // do the identical process for (b12,b13) and (b22,b23)
    ...
}

return diff
```

References

The Pearson Hash

[1] "Fast Hashing of Variable-Length Text Strings" by Peter K. Pearson
Communications of the ACM, Volume 33 Issue 6, June 1990 Pages 677-680.
<http://cs.mwsu.edu/~griffin/courses/2133/downloads/Spring11/p677-pearson.pdf>

SdHash

[2] "Data fingerprinting with similarity digests"
Vassil Roussev
Sixth IFIP WG 11.9 International Conference on Digital Forensics, Hong Kong, China, January 4-6, 2010
<http://roussev.net/pdf/2010-IFIP--sdhash-design.pdf>

Nilsimsa

[3] Source code for Nilsimsa <http://ixazon.dynip.com/~cmeclax/nilsimsa.html>

[4] "An open digest-based technique for spam detection"
E. Damiani¹, S. De Capitani di Vimercati¹, S. Paraboschi², P. Samarati
Proceedings of the 2004 international workshop on security in parallel and distributed systems. 2004.
<http://spdp.di.unimi.it/papers/pdcs04.pdf>

References (cont.)

SSDEEP

[5] "Identifying almost identical files using context triggered piecewise hashing"

Jesse Kornblum

Journal Digital Investigation: The International Journal of Digital Forensics & Incident Response archive
Volume 3, September, 2006 Pages 91-97

<http://dfcrws.org/2006/proceedings/12-Kornblum.pdf>

[6] Source code for SSDEEP: <http://ssdeep.sourceforge.net/>

Comparison Paper

[7] "An evaluation of forensic similarity hashes"

Vassil Roussev

Journal Digital Investigation: The International Journal of Digital Forensics & Incident Response archive
Volume 8, August, 2011

Pages S34-S41