ISSN (ONLINE):2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.25, February 2017

Detection of Sensitive Data Leaks on Mail Server

Dr.Giri.M[1], Shylaja.G[2], Suganya.M[3], Swathilakshmi.R[4]
[1]Associate Professor, Department of Computer Science and Engineering
Vel Tech High Tech Dr.Rangarajan Dr.Sakunthala Engineering College, Avadi, Chennai, Tamil Nadu.
[2, 3, 4] UG Scholars, Department of Computer Science and Engineering
Vel Tech High Tech Dr.Rangarajan Dr.Sakunthala Engineering College, Tamil Nadu, Avadi.
prof.m.giri@gmail.com, gshylu05@gmail.com, 95sugikarthik@gmail.com, swathi8675@gmail.com

***Abstract*- As internet grows and network bandwidth continues to get increased, ad ministrators are facing with the task of keeping confidential information from leaving their networks. Today the network traffic is voluminous that manual inspection would be unreasonably expensive. In response, researchers have created data loss prevention systems that checks outgoing traffic for known confidential information. These systems stop naive adversaries from leaking data, but fundamentally unable to identify encrypted information leaks. We present an approach for quantifying the information leak capacity in network traffic. Instead of trying to detect the presence of sensitive data an impossible task in the general case—our goal is to measure and constrain its maximum volume. By filtering this data, we can differentiate and quantify real information flowing from the computer. In this paper, we present data detection algorithms for the transformed data, the major algorithms for web browsing. When applied to the real web browsing traffic, algorithms were able to discount 98.5% of measured bytes and effectively isolate information sensitive leaks.**

***Index Terms*— Data leak detection, Lucene search engine framework, Levenshtein algorithm, inadvertent data leak, filtering, and base-64.**

INTRODUCTION

In an existing system straight forward realizations of data-leak detection require the plaintext Sensitive data. However, this requirement is undesirable, as it may threaten the confidentiality of the Sensitive information. If a detection system is compromised, then it may expose the plaintext Sensitive data. In addition, the data owner may need to outsource the data-leak detection to providers, but may be unwilling to reveal the plaintext. There was no privacy preserving in existing system, so providers can access he data without data-owners permission. Existing commercial data leak detection/prevention solutions include Symantec DLP, Identity Finder, Global Velocity, and Go Cloud DLP. Global Velocity uses FPGA to accelerate the system. All solutions are likely based on n-gram set intersection. Identity Finder searches file systems for short patterns of numbers that may be sensitive (e.g., 16-digit numbers that might be credit card numbers). It does not provide any in-depth similarity tests. Symantec DLP is based on n-grams and Bloom filters. The advantage of Bloom filter is space saving. However, as explained in the introduction, Bloom filter membership testing is based on unordered n-grams, which generates coincidental matches and false alarms. Bloom filter configured with a small number of hash functions has collisions, which introduce additional unwanted false positives.

ISSN (ONLINE):2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.25, February 2017

LUCENE SEARCH ENGINE FRAMEWORK

The Lucene search engine is a toolkit which is robust, powerful, and flexible search and is ready to handle many search problems. And since it' is now available under the more flexible LGPL open source license, is avilable free of cost.

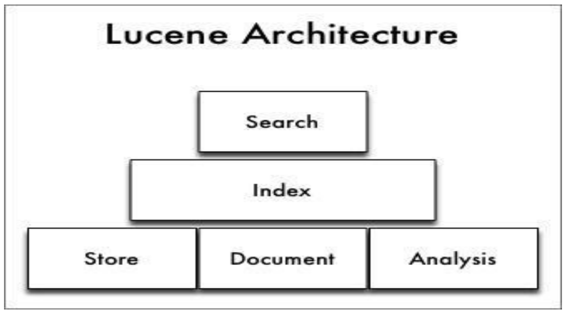


Fig.1: Lucene framework architecture

Doug Cutting, an experienced creator of text search and retrieval tools, created Lucene. Cutting is the preliminary author of the V-Twin search enginewhich is a part of Apple's Copland operating system effort and is presently a senior architect at Excite. He designed Lucene to make it easy to sum up indexing and search capability to a wide range of applications, including:

- Searchable email: An email application will let users search archived messages and add new messages to the index as they enter.
- Online documentation search: A documentation reader (i.e., CD-based, Web-based, or embedded within the application) will let users search online documentation or archived publications.
- Searchable Webpages: A Web browser or proxy server will build a personal search engine to index every Webpage a user has seen, allowing users to easily revisit the pages.
- Website search: A CGI program will let users search you're Website.
- Content search: An application will let the user search saved documents for specific content; this will be integrated into the Open Document dialog.

Version control and content management: A document management system will index documents, or document versions, so they can be easily taken.

News and wire service feeds: A news server or relay will index articles as they arrive. Many search engines could perform many of those functions, but only few open source search tools offer with Lucene's ease of use, rapid implementation, and flexibility.

Lucene when developing Eyebrowse, an open source Java-based tool used for cataloguing and browsing mailing lists. A major requirement for Eyebrowse was the flexible message search and retrieval ability. It urged an indexing and search component that would efficiently update an index base as new messages entered, allow multiple users to search and update the index base simultaneously and scale to archives which contains millions of messages.

Every open source search engine including Swish-E, Glimpse, iSearch and libibex was poorly suited to Eyebrowse's requirements in either way. This made integration problematic and time-consuming. With Lucene, adding index and search to Eye browse in little more than half a day, from an initial download to fully working code. That was less than one-tenth of the and yielded a more tightly integrated and feature-rich result than other search tool considered.

How search engines work

Creating and maintaining an inverted index is the major problem when building an efficient search engine. To index a document, we must first scan it to provide a list of postings. Postings describe number of occurrences of a word in a document (i.e., the word, a document ID, and possibly the location(s) or frequency of word) within the document.

If the postings are tuples of the form <word, document-id>, a set of documents will yield a list of postings arranged by document ID. But to efficiently find documents that contain particular words, instead sort the postings by word (or by both word and document, which make multiword searches fast). In that sense, building a search index is primarily a sorting problem. The search index is the list of postings sorted by a word.

An innovative implementation

Most of the search engines uses B-trees to maintain index; they are relatively constant with respect to insertion and have proper I/O characteristics (lookups and the insertions are O(log n) operations). Lucene takes a slight different approach that is rather than maintaining a single index, it creates multiple index segments and merges them immediately. For every document indexed, Lucene creates new index segments each time and quickly merges small segments with large ones .This keeps the total number of segments small so searches will remain fast. To optimize the index searching faster, Lucene can merge all the segments into one segment,and that is useful for infrequently updated indexes. To prevent conflicts / locking overhead between index readers and writers, Lucene will never modify segments in place, it only create the new ones.

A Lucene index segment has of several files:

- A dictionary index with one entry for each 100 entries in the dictionary
- A dictionary with one entry for each unique word
- A postings file with an entry for each posting

Since Lucene will never update segments in place, they get stored in flat files instead of complicated B-trees. For fast retrieval, the dictionary index has offsets into the dictionary file, and dictionary hold offsets into the postings file

- Create an index

for creating an index this is a simple program CreateIndex.java creates an empty index by creating an IndexWriter object and instructing it to build an empty index. In this example, the name of directory that will store the index is specified in the command line.

```
public class CreateIndex

{

// usage is CreateIndex index-directory

public static void main(String[] args) throws IO Exception

{

String indexPath = args[0];

IndexWriter writer;

//      An index is created by opening the IndexWriter

//      create an argument set to true.

writer = new IndexWriter(indexPath, null, true);

writer.close();

}

}
```

- Index text documents:

IndexFile.java shows how to add documents , the files named on the command line to an index. For every file, Index Files creates a Document object, then it calls IndexWriter.addDocument to add that to the index. From Lucene point of view, a Document is a collection of fields that are name value pairs. A Field will obtain its value from a String for short fields or an InputStream, for long fields.

```
public class IndexFiles

{
```

```
// usage: IndexFiles index-path file

public static void main(String[] args) throws Exception

{

String indexPath = args[0];

IndexWriter writer;

writer = new IndexWriter(indexPath, new SimpleAnalyzer());

for (int i=1; i<args.length; i++)

{

System.out.println("Indexing file " + args[i]);

InputStream is = new FileInputStream(args[i]);

//	We create Document with two Fields, one contains

//	the file path, and one the file's contents.

Document doc = new Document(); doc.add(Field.UnIndexed("path", args[i]));

doc.add(Field.Text("body", (Reader) new InputStreamReader(is)));

writer.addDocument(doc);

is.close();

};

writer.close();

}

}
```

LEVENSHTEIN DISTANCE

Levenshtein distance (LD) is a measure of similarity between the two strings, which is referred to as source string (s) and the target string (t). The distance is the number of deletions, insertions, or substitutions required to transform from s into t.(i.e.,)

- If s and t is a"test", then LD(s,t) = 0, because no transformations are needed. The strings are already identical.

- If s and t is a "test", then LD(s,t) = 1, because one substitution (change "s" to "n") is sufficient to transform s into t.

The greater is the levenshteindistance,the more different the strings are.

Levenshtein distance is named after the Russian scientist Vladimir Levenshtein, who invented the algorithm in 1965. If you can't spell or pronounce Levenshtein, the metric is also sometimes called as edit distance.

The Levenshtein distance algorithm has been used in:

- Spell checking
- Speech recognition
- DNA analysis
- Plagiarism detection

The following simple Java applet allows you to experiment with different strings and to compute their Levenshtein distance:

Algorithm:

```
Step.1.   Set n to be the length of s. Setm to bethe length of t. If n = 0, Return m
          and exit. If m = 0, return n and exit. Construct a matrix containing 0..m rows and
          0..n columns.
Step.2.   Initialize the First row to 0..n. Initialize the first column to 0..m.
Step.3.   Examine each character of s (i from 1 to n).
Step.4.   Examine each character of t (j from 1 to m).
Step.5.   If s[i] equals t[j], the cost is 0. If s[i] doesn't equal t[j], the cost is
          1.
Step.6.   Set cell d[i,j] of the matrix minimum of:
              a) The cell immediately above plus 1: d[i-1,j] +1.
              b) The cell immediately to the left plus 1: d[i,j-1] + 1. c. The cell diagonally
                 above and to the left plus the cost: d[i-1,j-1] + cost.

Step.7.   After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell
          d[n,m].
```

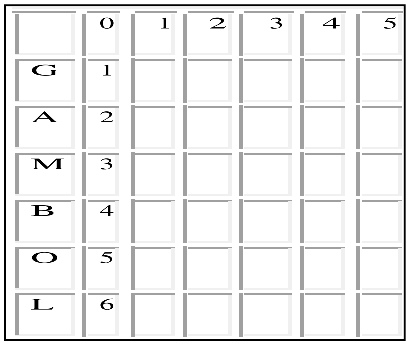
ISSN (ONLINE):2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.25, February 2017

Example

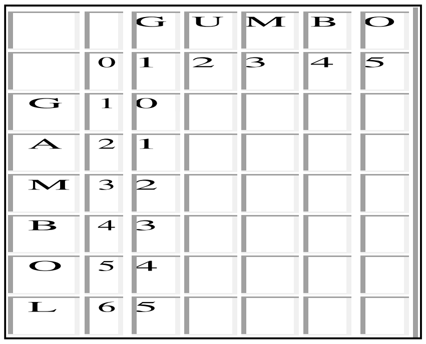
This section shows that how the Levenshtein distance is computed when the source string is "GUMBO" and the target string is "GAMBOL".

Steps 1 and 2



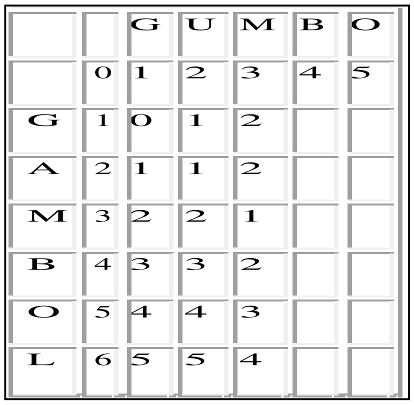
	0	1	2	3	4	5
G	1					
A	2					
M	3					
B	4					
O	5					
L	6					

Steps 3 to 6 When i=1



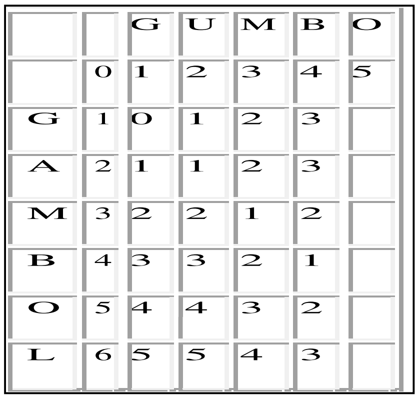
		G	U	M	B	O
	0	1	2	3	4	5
G	1	0				
A	2	1				
M	3	2				
B	4	3				
O	5	4				
L	6	5				

Steps 3 to 6 When i =3



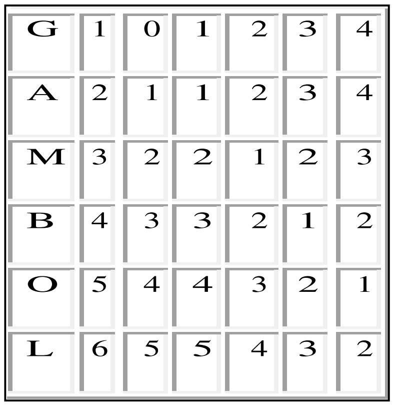
		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2		
A	2	1	1	2		
M	3	2	2	1		
B	4	3	3	2		
O	5	4	4	3		
L	6	5	5	4		

Steps 3 to 6 When i; =4



		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2	3	
A	2	1	1	2	3	
M	3	2	2	1	2	
B	4	3	3	2	1	
O	5	4	4	3	2	
L	6	5	5	4	3	

Steps 3 to 6 When i =5



G	1	0	1	2	3	4
A	2	1	1	2	3	4
M	3	2	2	1	2	3
B	4	3	3	2	1	2
O	5	4	4	3	2	1
L	6	5	5	4	3	2

Step 7:

The distance is in the lower right hand corner of the matrix, which is 2. This corresponds to our intuitive realization that the "GUMBO" can be transformed into "GAMBOL" by substituting "A" for "U" and adding "L" (one substitution and 1 insertion = 2 changes).

SOURCE CODE

Religious wars are often flare up whenever the engineers discuss differences between programming languages. A typical assertion is Allen Holub's claim in a JavaWorld article of July 1999. Neither the Microsoft Foundation Classes (MFC) or most of the other Microsoft technology which claims to be object-oriented."

We prefer to take a neutral stance in these religious wars.If a problem can be solved in one programming language, you can usually solve it in another as well. A good programmer is able to move from one language to another with relative ease, and completely learning a new language should not present any major difficulties. A programming language is a means to an end, not an end in itself.

As a principle of this neutrality, we present a source code which implements the Levenshtein distance algorithm in the following programming language:

Java

```
public class Distance

{

// Get minimum of three values

private int Minimum (int a, int b, int c) { int mi;

mi = a;

if (b < mi) { mi = b;

}

if (c < mi) { mi = c;
```

```
}
return mi;
}
// Compute Levenshtein distance public int LD (String s, String t) { int d[][]; // matrix
int n; // length of s int m; // length of t
int i; // iterates through s int j; // iterates through t char s_i; // ith character of s char t_j; // jth
character of t int cost; // cost
// Step 1
n = s.length (); m = t.length (); if (n == 0) {
return m;
}
if (m == 0) { return n;
}
d = new int[n+1][m+1]; // Step 2
for (i = 0; i <= n; i++) { d[i][0] = i;
}
for (j = 0; j <= m; j++) { d[0][j] = j;
}
// Step 3
for (i = 1; i <= n; i++) { s_i = s.charAt (i - 1);
// Step 4
for (j = 1; j <= m; j++) { t_j = t.charAt (j - 1);
// Step 5
if (s_i == t_j) { cost = 0;
}
else { cost = 1;
}
```

ISSN (ONLINE):2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.25, February 2017

```
// Step 6
d[i][j] = Minimum (d[i-1][j]+1, d[i][j-1]+1, d[i-1][j-1] + cost);
}
}
// Step 7
return d[n][m]
}
}
```

A. Upper and lower bounds:

The Levenshtein distance has a several simple upper and lower bounds. These includes:

- It is always at least the difference of the sizes of two strings.
- It is at most the length of the longer string.
- It is zero if and only if the strings are equal.
- If the strings are of same size, the Hamming distance is an upper bound on the Levenshtein distance.

APPLICATIONS

In the approximate string matching, the objective is to find matches for short strings in many longer texts, in situations where a small number of differences is to be expected. The short strings could come from a dictionary (i.e.,) one of the strings is typically short, while the other is arbitrarily long. This has a wide range of applications, for eg: spell checkers, correction systems and software to assist natural language translation based on translation memory.

The Levenshtein distance can also be computed between two longer strings, but the cost is to compute it, which is roughly proportional to the product of the two string lengths, makes this impractical. Thus, when used to aid in fuzzy string searching in applications such as record linkage, the compared strings are usually short to improve speed of comparisons.

A. Relationship with other edit distance metrics:

There are some popular measures of edit distance, which are calculated using a different set of operations. For example, The Damerau–Levenshtein distance allows insertion, deletion, substitution, and the transposition of two adjacent characters;nthe longest common subsequence metric will allow only insertion and deletion, not substitution; the Hamming distance allows only substitution, hence, it can only applied to strings of same length. Edit distance is defined as a

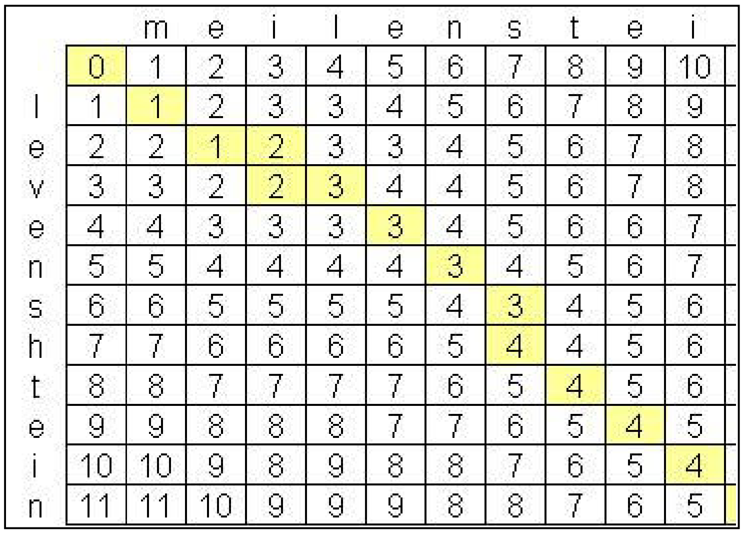
ISSN (ONLINE):2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.25, February 2017

parameterizable metric calculated with a specific set of allowed edit operations, and each operation is assigned as a cost which is possibly infinite. This is further simplified by DNA sequence alignment algorithms which makes an operation's cost depend on where it is applied.

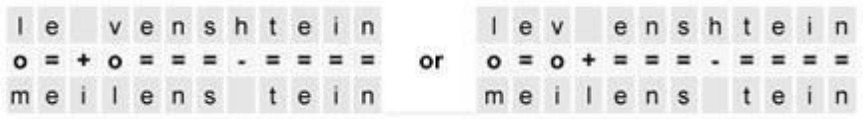
WORKING

The Levenshtein algorithm calculates the least number of edit operations that are necessary to modify one string to obtain another string. The most common way of calculating this is performed by the dynamic programming approach. The levenshtein distance between the m-character prefix of one with the n-prefix of the other word. The matrix can be filled from the upper left to the lower right corner. Each jump can be horizontally or vertically corresponds to an insert or delete. The cost is normally set to 1 for each and every operations. The diagonal jump can be cost either one, if the two characters in the row and column does not match or 0, they can perform. Each cell always decreases the cost locally. In this way the number in the lower right corner is the Levenshtein distance between both words.



		m	e	i	l	e	n	s	t	e	i
	0	1	2	3	4	5	6	7	8	9	10
l	1	1	2	3	3	4	5	6	7	8	9
e	2	2	1	2	3	3	4	5	6	7	8
v	3	3	2	2	3	4	4	5	6	7	8
e	4	4	3	3	3	3	4	5	6	6	7
n	5	5	4	4	4	4	3	4	5	6	7
s	6	6	5	5	5	5	4	3	4	5	6
h	7	7	6	6	6	6	5	4	4	5	6
t	8	8	7	7	7	7	6	5	4	5	6
e	9	9	8	8	8	7	7	6	5	4	5
i	10	10	9	8	9	8	8	7	6	5	4
n	11	11	10	9	9	9	8	8	7	6	5

There are two possible paths through the matrix which actually produces the least cost solution. Namely



l	e		v	e	n	s	h	t	e	i	n
o	=	+	o	=	=	=	-	=	=	=	=
m	e	i	l	e	n	s		t	e	i	n

or

l	e	v		e	n	s	h	t	e	i	n
o	=	o	+	=	=	=	-	=	=	=	=
m	e	i	l	e	n	s		t	e	i	n

"=" Match; "o" Substitution; "+" Insertion; "-"Deletion

Though there are sophisticated improvements on the complexity, there is no alternative to calculating the matrix to at least a large extent.

ISSN (ONLINE):2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.25, February 2017

CONCLUSION AND FUTURE WORK

Hence we introduced and proposed fast detection of data leak framework to avoid sensitive data which is exposed and also provide privacy for the sensitive data..We used data leak detection technique on the transformed data to provide security. We also presented a deep packet inspection technique for detecting leaks in the content of files or network traffic and searches for any occurrences of sensitive data patterns. For future work, we plan to explore data-movement tracking approaches for data leak prevention on a host.

REFERENCES

[1] Xiaokui Shu, Jing Zhang, Danfeng (Daphne) Yao, Senior Member, IEEE, and Wu-Chun Feng, Senior Member, IEEE" Fast Detection of Transformed Data Leaks", VOL. 11, NO. 3, MARCH 2016.

[2] A.V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," Commun. ACM, vol. 18, no. 6, pp. 333–340, Jun. 1975.

[3] Global Velocity Inc. (2015). Cloud Data Security from the Inside Out— Global Velocity. [Online]. Available: http://www.globalvelocity.com/, accessed Feb. 2015.

[4] J.Jang, D. Brumley, and S. Venkataraman, "BitShred: Feature hashing malware for scalable triage and semantic analysis," in Proc. 18th ACM Conf. Comput. Commun. Secur. (CCS), 2011.

[5] L.Yang, R. Karim, V. Ganapathy, and R. Smith, "Improving NFA-based signature matching using ordered binary decision diagrams," in Proc. 13th Int. Symp. Recent Adv. Intrusion Detect., Sep. 2010.

[6] K.Li, Z. Zhong, and L. Ramaswamy, "Privacy-aware collaborative spam filtering," IEEE Trans. Parallel Distrib. Syst., vol. 20, no. 5, pp. 725–739, May 2009.

[7] P.-C. Lin, Y.-D. Lin, Y.-C. Lai, and T.-H. Lee, "Using string matching for deep packet inspection," Computer, vol. 41, no. 4, pp. 23–28, Apr. 2008.