Detection of Sensitive Data Leaks on Mail Server

Giri M[1], Shylaja G[2], Suganya M[3], Swathilakshmi R[4]

[1]Department of Computer Science and Engineering, Vel Tech High Tech Dr. Rangarajan Dr. Sakunthala Engineering College, Avadi, Chennai, TamilNadu.

***Corresponding author: E-Mail:prof.m.giri@gmail.com**

ABSTRACT

When the network grows and its bandwidth begins to get increased, administrators should do the work of keeping authorized information. Today the data traffic is very large so that data acquisition would be costlier for no reason. Researchers in order have created a data losses preventing system that checks the traffic on confidential data so that it will stop new adversaries from the leakage, but initially it is unable to detect or identify encrypted information leaks. Instead of trying the detection of sensitive data an impossible task in the general case—our aim is to measure and limit its maximum volume. By finding this data, we can differentiate and quantify real information traversing from the computer. In our paper, the data detection algorithm for the transformed data is the major algorithms for web browsing. When this is applied to the real web traffic, the techniques were able to reduce 96.5% of measured bytes.

KEY WORDS: Data leak detection, Lucene search engine framework, Levenshtein algorithm, inadvertent data leak, filtering, base-64.

1. INTRODUCTION

In an existing system it is an effortless of data leakage detection that requires the plaintext Sensitive data. This requirement is undesirable, as it may not provide the confidentiality of the Sensitive information. If a detection system is satisfied, then it may provide the plaintext Sensitive data. Then the data admin has to outsource the data leakage detection technique, but it may not be ready to reveal the plaintext. There was no privacy got preserved in the existing system, so providers will view the data without data owners permission. Global Velocity use FPGA to speeding up the system. All solutions are involved in n-grams set intersection. The identity finder searches file system available for the tiny patterns of digits that may be sensitive (e.g., 16-digits number that might be credit card number). It would not provide a deep similarity tests. The DLP is based on n-grams as well as bloom filters. The pros of bloom filter is that it saves space among data

Lucene Search Engine Framework: The Lucene Search Engine is a toolkit which is reliable, powerful, and provides flexibility in search and also it can handle many searching problems. And since it is at present available under the more flexible LGPL open source license, is available free of cost.

This is a part of Apple's operating system effort and is presently a senior architect at Excite. They designed Lucene to make it easy to sum up indexing and search capability to wide applications, including:

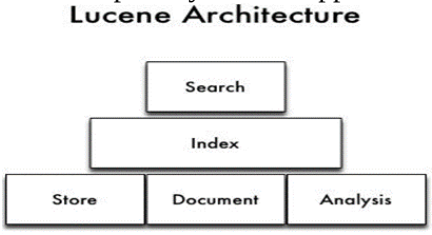

Figure.1. Lucene framework architecture

Searchable e-mail: An email application will let the users search archived messages and add new messages to an index as they enter.

Documentation search: A data reader (i.e., CD-based, Web-based, or embedded within the application) will allow the users to search online documentation or archived publications.

Searchable Web Pages: A proxy server will build a personal search engine for indexing the every Webpage, which allows users to easily visiting of the pages.

Website search: A CGI program will let the users while searching Website.

Content search: An application will let the users search saved documents for specific content; this will be integrated into the Open Document dialog.

How search engines will work: The Developing an inverted index is the major problem when building an efficient search engine. For indexing a document, we must first scan to provide a available number of postings. Posting will describe the 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) in the document.

If the postings are visible which will be like <word, document-id>, a set of documents will yield a of postings arranged by document ID. But for finding the documents efficiently that contain particular words, instead sort the postings by word. In that sense, introducing a search index is primarily a sorting problem. The search index is the postings sorted by a word.

An innovative implementation: Almost of the search engines uses B-trees to maintain the index, they are relatively constant with respect to the insertion and have proper I/O characteristics (the insertions are O (log n) operations).The Lucene took a slight different approach that is instead of handling a single index, it will create multiple index segments and merges them immediately. For every document being indexed, Lucene search engine creates a new index segments each time and quickly rejoins the small segments with the larger ones .This holds the whole number of small segments so searches will remain fast. To optimize the index searching faster, Lucene can merge all the segments into a single segment, and that is useful for frequently updated index. To prevent conflicts or locking overhead, Lucene will never modify segments instead it only creates the new one.

The Lucene index has of several files:

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

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

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

Index text documents: IndexFile.java shows how to add documents, the files named on to an index. For every file, Index Files creates a Document object, and then it calls Index Writer. Add Document to add that to an 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 Input Stream, for long fields. Fields allows you to partition a document into separate searchable and index able sections and to associate metadata (i.e., name, author, or modification date) with a document. For example, when storing a mail messages, you could put a message subject, author, date, and body in a separate fields, then can build semantically richer queries. In the code given below, we store two fields in every Document: path, to identify the accurate file path it can be retrieved later.

We will create a Document which involves two Fields; one contains the file path, and other the file contents.

Levenshtein Distance: Edit distance technique is the similarity measures between the two strings, referred as source (s) and the target (t). The edit distance is defined as the number of substitutions, insertions or deletions required for transforming from s into t.

The greater one is the edit distance, the more different the strings are. If it cannot be able to spell or pronounce edit, the term is sometimes called as edit distance. The edit distance algorithm can also be used in:

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

Sourcecode: Religious wars are often out-burst whenever the engineers discuss differences between programming languages. The Microsoft Foundation Classes, among most of the other Microsoft technology which claims to be object-oriented. We preferred 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 can able to move from one language to another 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.

Upper and lower bounds: The edit distance which has a simple upper and lower bounds. These includes:

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

Working: The edit distance algorithm used to calculates the least number of edit operations which 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 edit 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 operation.

The cost is normally set to 1 for each and every operation. The diagonal jump cost can be either one, if the two characters in the row and column do not match or 0, they can perform. Each cell always decreases the cost locally.

In this way the number in the lower right corner performs:

Table.1. Experimental table

		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.

```
le venshtein      lev enshtein
o=+o====.=====  or o=o+====.====
meilens tein      meilens tein
```

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

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

3. CONCLUSION AND FUTURE WORK

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

REFERENCES

Aho A.V and Corasick M.J, Efficient string matching, An aid to bibliographic search, Commun. ACM, 18 (6), 1975, 333–340.

Global Velocity Inc, Cloud Data Security from the Inside Out—Global Velocity, 2015.

Jang J, Brumley D and Venkataraman S, BitShred, Feature hashing malware for scalable triage and semantic analysis, in *Proc.* 18th ACMConf. Comput. Commun. Secur. *(CCS)*, 2011.

Li K, Zhong Z and Ramaswamy L, Privacy-aware collaborative spam filtering, IEEE Trans. Parallel Distrib. Syst, 20 (5), 2009, 725–739.

Lin P.C, Lin Y.D, Lai Y.C and Lee T.H, Using string matching for deep packet inspection, Computer, 41 (4), 2008, 23–28.

Xiaokui Shu, Jing Zhang, Danfeng (Daphne) Yao, *Senior Member*, IEEE and Wu-Chun Feng, Senior Member, IEEE, Fast Detection of Transformed Data Leaks, 11 (3), 2016.

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