INTERNATIONAL JOURNAL OF PURE AND APPLIED RESEARCH IN ENGINEERING AND TECHNOLOGY

A PATH FOR HORIZING YOUR INNOVATIVE WORK

SECURITY ENHANCEMENT OF WEB DATABASE USING THE TECHNIQUE OF NEGATIVE DATABASE

RANJIT KUMAR, SANDEEP RAJ

IJPRET-QR CODE *PAPER-QR CODE*

1. Assistant Professor, Galgotias College of Engineering and Technology, Greater Noida.

Accepted Date:

16/11/2012

Publish Date:

01/12/2012

Keywords

Database

Security

Corresponding Author

Mr. Ranjit Kumar

Assistant Professor, Galgotias College of Engineering and Technology, Greater Noida.

Abstract

Web database is combination of database and web technology. Web database is placed on the Internet, there are many security problems. Web data security is a major issue in any web based system. Web and distributed databases play the key role in most of these Web applications and thus it is critical to protect them from unauthorized access and malicious attacks. A negative database contains huge amount of data which contains of counterfeit data along with the actual data. Attackers may be able to get access to such databases, but, as they try to extract information, they will retrieve data sets that would include both the actual and the negative database. We present an approach to secure database using negative database.

INTRODUCTION

A striking feature of the natural immune system is its use of negative detection in which *"self"* is represented (approximately) by the set of circulating lymphocytes that fail to match self. This suggests the idea of a negative representation, in which a set of data elements is represented by its complement set. That is, all the elements not in the original set are represented (a potentially huge number), and the data itself are not explicitly stored. This representation has interesting information-hiding properties when privacy is a concern and it has implications for intrusion detection.[5] In a negative database, the negative image of a set of data records is represented rather than the records themselves. For now we assume a universe U of finite-length records (or strings), all of the same length l, and defined over a binary alphabet. We logically divide the space of possible strings into two disjoint sets: DB representing the set positive records (holding the information of interest), and $U - DB$ denoting the set of all strings not in DB. We assume that DB is uncompressed (each record is represented explicitly), but we allow $U - DB$ to be stored in a compressed form called *NDB*. We refer to *DB* as the *positive* database and *NDB* as the *negative* database.[4]

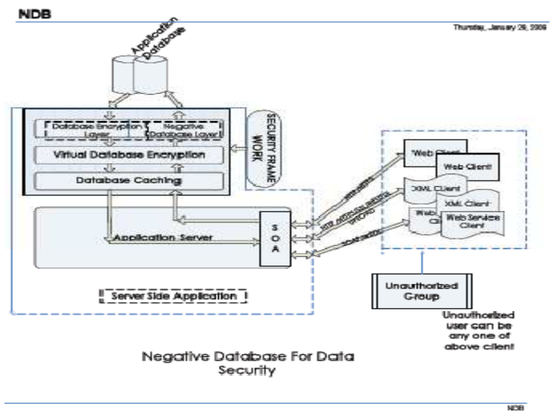
Negative information can be represented efficiently, even though the negative image will typically be much larger than the positive image. In particular, a database consisting of n, l-bit records can be represented negatively using only $O\ (ln)$ records.[5]

Current technologies of encryption (for the data itself) and query restriction (for controlling access to the data) help ensure confidentiality, but neither solution is appropriate for all applications. In the case of encryption, the ability to search data records is hindered, while in the case of query restriction, individual records are vulnerable to insider attacks. Negative databases potentially address both of this concerns.[4] Security in database has become an important problem because of the large amount of personal data, which is tracked by many business web applications. Security data mining addresses the need of multiple parties with private inputs to run a data mining algorithm and learn the results over the combined data without revealing any unnecessary information. Security is a major concern in the application of database techniques to datasets containing personal sensitive or confidential information

NEGATIVE DATABASE

This paper presents a security framework which can be embedded at the middle layer of any web-based or stand-alone application that requires high security levels. There is no restriction on the database management system that needs to be used. The top level architectural diagram of the implementation of the security framework concept is shown in Figure below. A user submits a query to the database using the web browser component. The server starts processing the query and the security framework comes into action. The middle tier includes the framework that consists of several technologies combined together to give a robust safeguard to the database layer that lies below the framework. More specifically, the framework consists of four main modules, namely, Database caching, Virtual database encryption and Database encryption algorithm, along with Negative Database conversion algorithm. The manipulated data is finally stored into the actual database after passing through all the components of the security framework. Thus the database contains the

Negative data, which is a manipulated form of the original one, as well as the positive data. All the queries that are used by a valid user work with the Java technology; hence the database is updated as objects and its attributes. Each table is represented as an object and the tuples are the values of the attributes of the object. Any query that is invoked towards the database will be associated with objects and its attributes; hence an invalid query will not be able to retrieve any useful information from the database.[6]



Proposed Algorithm

The algorithm is invoked when a SELECT, INSERT, UPDATE or DELETE action is performed in the database, maintaining the ACID property and providing high amount of security to the data in the database along with a faster access.

Algorithm

Virtual database encryption

```
1. Read and Get data from the user
2. Get the UNICODE value of the data.
2a. Calculate length of the data.
2b. Convert UNICODE: charAt(i); Store into variable "c"
2c. IF c=[A-Z] and c=[a-c] and c[0-9]
THEN (multiply by 10)
ELSE Take variable "j" and store the ASCII value in j
OR
2c. Concatenate all the UNICODE value of each character with any primitive number "*" and store it in “j”
3. Get the current system date and time.
3a. Create an object of Time Stamp
3b. Get "Year”, “Month”, "Day"
3c. Get "Hour”, "Minute”, "Second"
3d. Get AM as 0 and PM as 1.
Store into variable "z".
IF z=0 THEN "hour=hour".
ELSE "hour"="hour+12".
3e. Append all the values of “Year”, “Month”, “day”, “hour”, “minute”, “Second” and “z” and store in Time Stamp.
4. The Time Stamp is appended to the ASCII value of the data which is currently stored in variable “j” and pass the generated value to the Database Encryption algorithm, RSA_publicKey () layer.
```

3.1Database Encryption algorithm, RSA_publicKey ()

The database is always populated by data that is entered by the administrator and updated by a benign user, which could be a online banking or credit card company customer. A problematic situation may arise when a malicious user tries to update or modify the database. The example of such updates could be a SELECT query inside an INSERT query. The purpose of the encryption module along with other three modules is to provide utmost security to the data. In our framework we use the public key encryption algorithm RSA. Strong encryption makes the database more secure and reliable.

The encryption algorithm is preceded by two modules which are Database caching and Virtual Database Encryption algorithm. It takes the input from the previous module and applies RSA on the data. The encrypted data is passed through the Negative Database conversion algorithm as shown in Figure below to generate encrypted multi sets of data for a single true set of data, making the database hard to query. The data encrypted in this layer is passed to the next layer called the Negative Database Conversion layer.[6]

```
1. Get the input from the previous section & pass to the RSA_publicKey ()
2. Get the RSA encrypted value and pass it to hex String from Bytes to convert it into hexadecimal values
2a. Take a variable String hex="";
2b. Take integer variables msb, lsb=0, i
2c. FOR EACH i ranging from (i=0) to (i<length of the input)
Calculate msb=((int)b[i]& 0x000000FF)/16;
lsb=((int)b[i]& 0x000000FF)%16;
hex= hex + hexChars[msb] + hexChars[lsb];
3. Pass the value generated from hexStringFromBytes to the getEncryptionData() method which uses MD5
3a. Get an instance of MD5
3b. Update the data using MD5 object
3c. Take String variable hexEncoded and pass the hex value using digest.
3d. Take String variable enCoded and pass the sub string values by truncating (13, 15)
4. This values is passed to the next layer called "Negative Database Conversion Algorithm"
```

Negative Database Conversion algorithm

The main concept of the framework lies on the implementation of this module. The purpose of using negative database along with the positive database is to provide extra security to the database. Any data manipulation query from the administrator or the customer is referred to as a benign user query; on the other hand any query from anyone else is always referred to as a malicious query. All the queries will go through all the modules to generate a secure database value.

The last module, also called the *Negative Database conversion algorithm*, is used to create a large set of values rather than just a single tuple. The generated sets of data are inserted in the database. Contrary to common database applications, in our negative database a malicious query will not be able to fetch the data from the database. The term *negative* is used because of the generation of false sets of data in reference to the actual data. The actual data passes through the Virtual Database Encryption algorithm and the Database Encryption algorithm, RSA_publicKey () layer to generate the data that passes through the negative database algorithm module. The number and the type of false data generation will depend on the algorithm used.[6]

NDB Conversion algorithm

```
1. Value: = Database Encryption algorithm, RSA_publicKey ()
2. Key: = Time Stamp from the Virtual database encryption layer
3. The value must be a string
4. Calculate the length of the values from the previous layer
5. The value is either the username or the password (the value can be any not only username and password)
OR
5. The value could be username as value 1 and password as value 2
6. DO FOR the length of the string in step 2
6a. Create User Account Object
6b. Set User Name
6c. similarly for all values that need to be changed into negative data
6d. Set User ModDt (timestamp)
7. Insert <value+"*"+timestamp> tuple in the database table after the input is manipulated using the algorithm in Figure above; it will be saved to the database.
```

For example, let the ACCNO be the attribute that we want to encrypt in the negative database. If the ACCNO value of customer

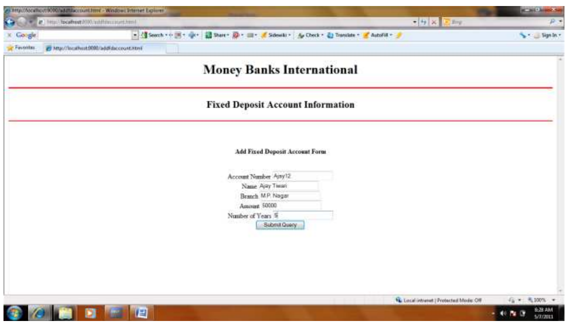
"Ajay Tiwari" after the Database Encryption algorithm, RSA_publicKey () layer is "Ajay12", Table below shows how the respective database table tuples.

ACCNO	Name
Apj659	Ajay
Jpl080	Ajay
Apa970	Ajay
Ypy1210	Ajay
1p1490	Ajay
2p2500	Ajay

The process of data retrieval from the database can be done in two ways, for a benign user and a malicious user. A benign user submits a legitimate query into the database and is able to get the correct output. On the other hand, a malicious user writes some queries that could be vulnerable and are able to retrieve some output, but the output will comprise of numerous non-interpretable data sets, as the ones shown in Table above.

Implementation

For testing negative database concept we implemented a real life secure banking application. The performance test is performed on Pentium IV 3 GHz machine with 2 MB RAM. The database server used is Oracle 10g with application server. Java technology such as JDBC, JSP, Servlets are used to implement the framework. A user submits a query to the database using the web browser component. The server starts processing the query and the security framework comes into action. The middle tier includes the framework that consists of several technologies combined together to give a robust safeguard to the database layer that lies below the framework. The framework consists of four main modules, namely, Database caching, Virtual database encryption and Database encryption algorithm, along with Negative Database conversion algorithm. Database caching is used to cache the data for fast access. Database encryption algorithm is used to secure the database. Negative database conversion algorithm is used to store actual data along with counterfeit data. Attackers may be able to get access to such databases, but, as they try to extract information, they will retrieve data sets that would include both the actual and the negative database. We have also implemented insertion and updation operation for negative database. The manipulated data is finally stored into the actual database after passing through all the components of the security framework. Thus the database contains the *negative* data, which is a manipulated form of the original one, as well as the positive data. All the queries that are used by a valid user work with the Java technology; hence the database is updated as objects and its attributes. Each table is represented as an object and the tuples are the values of the attributes of the object. Any query that is invoked towards the database will be associated with objects and its attributes; hence an invalid query will not be able to retrieve any useful information from the database. The output is shown below. Figure below shows the insertion operation in negative database.

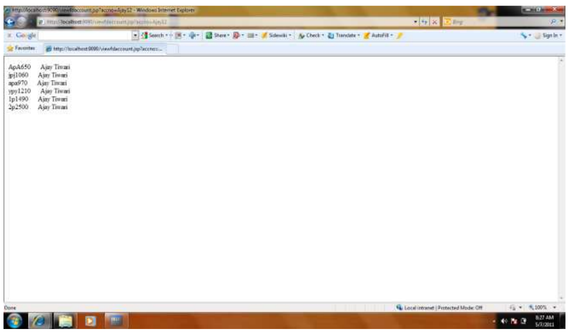

Money Banks International

Fixed Deposit Account Information

Add Fixed Deposit Account Form

Account Number	Ajay12
Name	Ajay Tiwari
Branch	M.P. Nagar
Amount	50000
Number of Years	5

Submit Query


```
ApA650    Ajay Tiwari
jpj1060   Ajay Tiwari
apa970    Ajay Tiwari
ypy1210   Ajay Tiwari
1p1490    Ajay Tiwari
2p2500    Ajay Tiwari
```

Conclusion and Future Work

A negative database contains huge amount of data which contains of counterfeit data along with the actual data. Attackers may be able to get access to such databases, but, as they try to extract information, they will retrieve data sets that would include both the actual and the negative database. Web database is combination of database and web technology. Web data security is a major issue in any web based applications. Real world databases have information that needs to be securely stored. This solution of designing a security framework aims at providing high security to a web-based environment, where attacks to the database are frequent which results in theft of data and data corruption. We present a security framework which can be embedded at the middle layer of any web based or stand alone applications that requires high security levels. We propose a framework that allows the negative representation of the original set of data, resulting in returning invalid results for malicious queries. At the same time, it enables the retrieval of the original data in case of legitimate queries. There are still many open issues related to the notion of negative databases for real world applications. The framework consists of four main modules, namely, Database caching, Virtual database encryption and Database encryption algorithm, along with Negative Database conversion algorithm. We provide the web security at the database layer.

The manipulated data is finally stored into the actual database after passing through all the components of the security framework. Thus the database contains the *negative* data, which is a manipulated form of the original one, as well as the positive data. The framework provides secure retrieval and storage of data, for a valid user access denial for a malicious user. The framework is implemented for the banking enterprise. We have provided updation and insertion operation in negative database. The negative database has the manipulated value as well as the timestamp of each insertion and updation operation.

In future work we are planning to apply semantic web technology to improve the searching operation and performance of the web based real applications.

REFERENCES

1. Ebersole S, Hibernate core for JAVA. 2008. http://www.hibernate.org/344.html

2. Esponda F, Negative Representations of Information, Ph.D. Dissertation, the University of New Mexico, 2005.

3. Esponda F, Everything that is not important: Negative databases, Computational Intelligence Magazine, IEEE, 2008; 3(2): 60 – 63.

4. Esponda F, Ackley ES, Helman P, Jia H and Forrest S, Protecting data privacy through hard to- reverse negative databases. International Journal of Information Security, 2007.

5. Negative Databases. Retrieved May 29, 2008, from http://esa.ackleyshack.com/ndb/

6. Patel A, Sharma N, Negative Database for Data Security, Master's Project Report, Computer Engineering Department, San Jose State University, 2008.

7. Elisa Bertino and Ravi Sandhu, Database Security- Concepts, Approaches, and challenges, IEEE Transactions on Dependable and secure computing, 2005.

8. Shariq Rizvi, Alberto Mendelzon, S. Sudarshan and Prasan Roy, Extending Query Rewriting Techniques for Fine grained access control, ACM SIGMOD, 2004.

9. Macro Vieira and Henrique Madeira, Towards a Security Benchmark for Database Management Systems, IEEE 2005.

10. Afonso Araujo Neto, Mecro Vieira and Henrique Madeira, an Appraisal to Access the Security of Database Configuration, IEEE 2009.

11. Sohail Imran and Irfan Hyder, Security Issues in Database, IEEE 2009.

12. Frank SR, Application Layer Intrusion Detection for SQL Injection, ACM SE, 2006.