ISSN:0975-9646

Vemula Vamshi Krishna et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (5) , 2011, 2225- 2228

A Framework on Database Content Security using Negative Databases

Vemula Vamshi Krishna
Dept. of CSE
SCCE
Karimnagar, AP, India

Rama Krishna V
Dept. of CSE
SCITS
Karimnagar, AP, India

Abstract-In today's Information warfare providing Data Security to Web based databases is a critical issue. Existing techniques to protect Database content are not sufficient to extend for web based databases. Hence there is a huge requirement for developing algorithms, which deals with data protection against intruders. Developing Negative Databases will solve such problem. A Negative Database is a Database which holds original data as well as forged data (i.e. counterfeit data). Intruders may be able to get access to such databases, then they will access data blocks which holds forged data along with original data. In this paper we try to present a framework to protect and improve the retrieval of data in the databases by negative data representations.

Keywords: Negative Databases, Data Security, Web databases

I. INTRODUCTION

In Web based Database Systems security is a big threat. People always try to access the information available in web through different techniques. In these situations the organizations such as Government and Security Agencies should provide utmost security to their web databases. There are number of techniques to secure the data, but the question is "How many are perfect enough in protecting the databases?". It is very difficult to provide 100% security to database from malicious accesses. Hence we are presenting a Framework using Negative Database representations. By Negative Database the original data in database will be represented in a complemented form. Here Negative Database is defined as a database which holds counterfeit data along with original data. When malicious user tries to access the information he will get data containing original data and counterfeit data which are in very difficult to analyze form. That means the malicious user unable to extract original data from that negative data. With this we can achieve utmost security by avoiding malicious accesses to the data in databases. As per our knowledge gained with literature survey it is observed that there are very few algorithms which are reversible. Hence in this paper we come up with a reversible algorithm to convert original data to its negative form. Some of the theoretical algorithms on reversible negative databases are proposed in [3] but they are not implemented to use with real time applications.

The main intension of our algorithms in this paper is to efficiently retrieve original data from negative data by avoiding malicious access of unauthenticated users and allowing access of authenticated users.

II. ARCHITECTURE

In this paper we present 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 1. 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 Hibernate technology [1]; 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. In what follows, we discuss in detail the framework's modules.

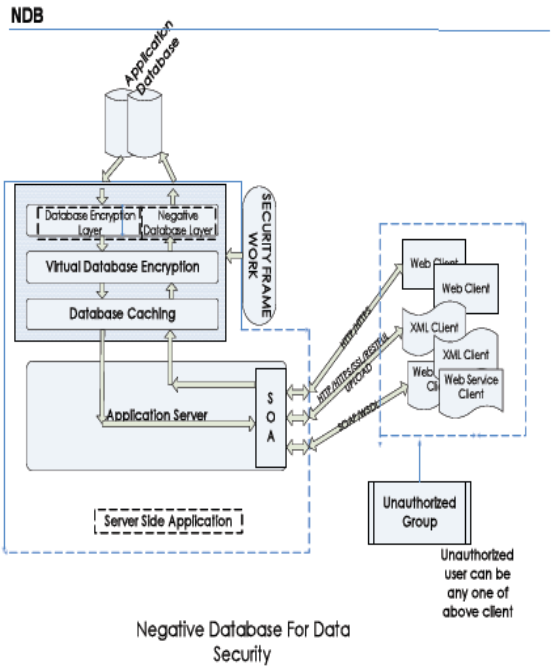

Figure 1. System Architecture

A. Database Caching

Database caching refers to the concept of easy access of frequently used data. It is the way in which frequently used data is stored in an easy or quick access area such as the RAM and defining specific ways of retrieving those data. The purpose of data caching has several advantages compared to the use of a simple database namely, (a) faster data access by using index to the disk and reduces disk access, and (b) higher CPU utilization as the computation time to access the data is reduced. This concept provides higher performance when there is huge amount of data access and a lot of modification to the database.

B. Virtual Database Encryption

Virtual database encryption refers to the use of a set of randomly generated keys attached to the objects in any hibernate scenario. It is one by generating a set of random keys using the system time and attaching it to the data in the form of objects and its attributes. This is detailed in Figure 2. The new values are then sent to the database as an attribute of object, the objects refer to the tables in the database and the value of the attributes relates to the tuples in the table.

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.

```
(Call before Session is updated)
1. Read and Get data from the user
2. Get the ASCII value of the data.
   2.a.Calculate length of the data.
   2.b.Convert ASCII: charAt(i);
        Store into variable "c"
   2.c.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
   2.c.Concatenate all the ASCII value
       of each character with any
       primitive number "*" and store it
       in "j"
3. Get the current system date and time.
   3.a.Create an object of TimeStamp
   3.b.Get "Year", "Month", "Day"
   3.c.Get "Hour", "Minute", "Second"
   3.d.Get AM as 0 and PM as 1.
        Store into variable "z".
        IF z=0 THEN "hour=hour".
        ELSE "hour"="hour+12".
   3.e.Append all the values of "year",
       "month", "day", "hour", "minute",
       "second" and "z" and store in
        TimeStamp.
4. The TimeStamp 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.
```

Figure 2. Pseudocode of Virtual database encryption

C. Database 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 3 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*.

```
1.Get the input from the previous section &
pass to the RSA_publicKey()
2.Get the RSA encrypted value and pass it
to hexStringFromBytes to convert it into
hexadecimal values
  2.a.Take a variable String hex="";
  2.b.Take integer variables msb,
      lsb=0, i
  2.c.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
  3.a.Get an instance of MD5
  3.b.Update the data using MD5 object
  3.c.Take String variable hexEncoded
      and pass the hex value using
      digest.
  3.d.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"
```

Figure 3. Database Encryption algorithm, RSA_publicKey()

D. 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.

```
1. value := Database Encryption algorithm,
RSA_publicKey()
2. key := TimeStamp 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 value1
and password as value2
6. DO
    FOR the length of the string in step 2
   6.a.Create UserAccount Object
   6.b.Set UserName
   6.c.Similarly for all values that need
       to be changed into negative data
   6.d.Set UserModDt(timestamp)
7. Insert <value+"*"+timestamp> tuple in
the database table
```

Figure 4. Pseudocode of Negative Database Conversion algorithm

After the input is manipulated using the algorithm in Figure 4, it will be saved to the database. For example, let the SSN be the attribute that we want to encrypt in the negative database. If the SSN value of customer "Niveeta" after the Database Encryption algorithm, RSA_publicKey() layer is "4@AGD", Table 1 shows how the respective database table tuples will look like.

Table 1. Negative Database snapshot

SSN	Name
4 +"*"+ (Time Stamp as Key)	NIVEETA
@+ "*" +(Time Stamp as Key)	NIVEETA
A+ "*"+ (Time Stamp as Key)	NIVEETA
G+ "*" +(Time Stamp as Key)	NIVEETA
D+ "*" +(Time Stamp as Key)	NIVEETA

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 (e.g. <SSN, Name>). 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 1.

III. PERFORMANCE

Database caching: In our framework we are using system-derived timestamps as keys. Thus the complexity of the database caching algorithm O(n), when the whole database needs to be searched for a particular tuple.

Virtual database encryption: This layer depends on the timestamp generation and the conversion of the data into ASCII values. Thus the computation time is O(n) where n is the length of the used password.

Negative Database conversion: The main task of this algorithm is to generate a set of data that is to be populated to the negative database. The input from the previous module to this module is an encoded value which is an 8 digit value, hence 8 row data has to be calculated and then populated to the database. This 8 digit value remains the same length because of some constraints made in the previous module. Hence this module is not affected by the size of input, hence has O(1) complexity.

Hence the overall complexity of the security framework is O(n), when we combine the complexities of all the modules. This is however very high compared to a simple web-based application. This can be compensated by the level of security this framework provides in comparison to the low- security, high risk of data for other applications.

IV. CONCLUSION

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. This paper provides evidence on the retrieval of data from a *negative database*. In all the previous *Negative* Database algorithms [3], the storage of actual data as negative data was explained, however was not implemented. To the best of our knowledge, the retrieval of negative data has not yet been addressed in the literature, and this paper has made a progress on that end. In this work, we propose a framework that allows the negative representation of the original set of data, resulting in returning invalid results for malicious queries (e.g. through login). 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. This paper provides evidence on the storage and retrieval of data, for a benign user and access denial for a malicious user. This would only deal with the INSERT and SELECT query of the database. The most challenging future research based on this project could be to UPDATE the negative database. The negative database has the manipulated value as well as the timestamp of each INSERT operation. The main challenge in this scenario is to update the database value checking the timestamp; this can be done by finding the values, which is the same process as search. However the updated value should have a negative set of itself along with the system time. Building real world applications can be one of the challenging works based on the algorithms explained in this paper.

V. FUTURE WORK

We can investigate the different methods for reducing the size of the Negative Database, and to efficiently retrieve the data from the Negative Database.

REFERENCES

[1] Esponda, F. (2005). *Negative Representations of Information, Ph.D. Dissertation*, The University of New Mexico

[2] Esponda, F., Ackley, E.S., Helman, P., Jia, H. & Forrest, S. (2007). *Protecting data privacy through hardto-reverse negative databases.* International Journal of Information Security, 6

[3] Negative Databases. (2006). Retrieved May 29, 2008, from http://esa.ackleyshack.com/ndb/

[4] F. Esponda, E.D. Trias, E.S. Ackley, and S. Forrest. *A Relational Algebra for Negative Databases*. UNM Computer Science Technical Report TR-CS-2007-18, November 2007.

[5] F. Esponda, E.S. Ackley, P. Helman, H. Jia, and S. Forrest. *Protecting Data Privacy through Hard-to-Reverse Negative Databases.* International Journal of Information Security (IJIS), Volume 6, Number 6, pp.403-415, October, 2007.

[6] F. Esponda, E.S. Ackley, S. Forrest and P. Helman. *On-line Negative Databases.* International Journal of Unconventional Computing, Volume 1, Number 3, pp.201-220, 2005.

[7] Fernando Esponda, Stephanie Forrest, and Paul Helman. *Enhancing Privacy through Negative Representations of Data.* (pdf) UNM Computer Science Technical Report TR-CS-2004-18, March 2004.