A Security Novel for a Networked Database

Mohammed Hamdi[1,2], Mejdl Safran[1,3] and Wen-Chi Hou[1]
Department of Computer Science
[1]Southern Illinois University, Carbondale, IL, USA
[2]Najran University, [3]King Saud University, Saudi Arabia
{mhamdi, mejdl.safran, hou}@siu.edu

***Abstract*— The security of databases is an important characteristic for database systems. It is intended to protect data from unauthorized access, damage or loss. With the advance of the methods of penetration and piracy, and with the increased reliance on databases that are connected with the Internet, the protection of databases has become one of the challenges faced by various emerging institutions, especially with the increasing of electronic crimes and thefts. In light of this, the focus is on analyzing and reviewing the cryptosystem architecture for networked databases. In this paper, we will discuss the process of encryption and decryption at the application and storage levels. Moreover, strategies of encryption inside the database by using the property of Transparent Data Encryption will be addressed. These methods will give a clear analysis of how data stored in databases can be protected and secured over the network. Additionally, these methods will help to overcome problems that are usually faced by administrative beginners, who work in the enterprises and manage their databases. Finally, we will discuss SQL injection, as a database attack and present the techniques of defense that prevent the adversaries from attacking the database.**

Keywords: Database, Security, Cryptography, SQL.

I. Introduction

A database is a storage place that contains the logical data elements represented by tables of columns and rows. These tables model relevant objects by joining them together through a mathematical relationship. The database is a set of mutually interrelated data that are managed by special applications. The database engine is designed to facilitate handling and search within this data, and enable the users to add and modify it [2], [3].

Database Management System (DBMS) is a set of computer programs that control the design, construction, use and storage of databases. This system allows the institutions and companies to control a huge data they own quickly and securely. The DBMS is a package of software that provides an interface between users and databases, and gives facilities to perform operations, namely, manipulating the data and managing the database structure itself. Additionally, the DBMS has the advantage of controlling redundancy and concurrence, enforcing integrity, and avoiding inconsistency, to name a few [17].

One of the great challenges faced by networked databases is protecting them from adversaries' attacks. The security in database is not only a great challenge to most of organizations and companies over the years, but also goes beyond to be the most concern for the governments. The database is considered to be the heart of any organization since it holds their sensitive information. However, with the advance of the methods of penetration and piracy, and with the increased reliance on databases that are connected with the Internet, the protection of databases has become one of the challenges faced by various establishments and institutions, especially with the increasing of electronic crimes and thefts [1],[2],[3], [4].

Access control is an essential technique used to protect the information stored on the database. User authentication is a core concept of access control to database. It is a process of verifying an identity stated by a system entity in order to limit the actions (e.g., search, insert, delete, etc.) on objects to specific users [1], [2]. The access control has three basic elements:

1) *Subject:* which is a process represented by a user or application that is capable to access the object.
2) *Object:* which is any object in the database such as tables/relations, tuples, views, operations, etc. that contains data, where its access is governed.
3) *Access right:* which elaborates the way in which a subject may access an object. For example, insert, delete, and select or only select from a specific table in a database.

Generally speaking, data protection techniques can secure the data at rest that refers to inactive data that is physically stored in a database and existed as a backup form [7]. Several enterprises might leave unused data in physical storages unprotected, thinking that no one can get access to it since it is physically stored in their databases. As a result, their data has become an easy objective to be accessed by unauthorized users. Given the growing concern about this type of data and its developed security's drawbacks over the years, enterprises have tended to protect this data. As a result, the companies first start determining the problems of protecting the data at rest and then set security protocols to secure these data types from unauthorized accesses.

Data at rest encryption emphasizes that cryptography can be performed on the physical storages of database [4]. In addition, this cryptosystem recommends that the keys should be updated on a regular basis and stored separately from the data. Periodic auditing of sensitive data should be part of the policy and should take place periodically. Also, the amount of the sensitive data that needs to be stored should be minimized. All of this is to prevent data visibility from unauthorized access or theft. In general, getting access to the data at rest stored in networked databases requires a number

978-1-4799-3010-4/14 $31.00 © 2014 IEEE
DOI 10.1109/CSCI.2014.55

of components. These components are interrelated technologies that allow the users to access the data at rest:

1) *An application programming interface (API)* is a set of routines, data structures, and protocols provided by libraries or operating systems to support the construction of programs. The API contains the procedure call made by the application. The API works to determine the data that goes to DBMS.

2) *IPC mechanism (Inter Process Communication Mechanism)* is a collection of methods that provide the communication mechanisms among multiple processes. It contains TCP/IP sockets which are constrained by OS and network.

3) *Network Protocol (NP)* is used to transport the data stream over the network. It involves TCP/IP, SSL, TLS, and HTTPS.

4) *Data Stream Protocol (DSP)* transfers the stream of data between a database server and its client. It works independently on a number of different networks. The data stream protocol can be optimized to work with a particular DBMS.

5) *Database Management System (DBMS)* is a software program that uses a standard ways of cataloging, retrieving and running queries on data. It provides methods for the data to be modified, controlled, and extracted by users, or other programs. For example MySQL, SQL Server, and Oracle.

II. Types of Cryptographic Algorithm

Cryptography is a study of mathematical techniques that convert sensitive data (i.e., plaintext) into an encrypted form of data (i.e., cipher text) as shown in Figure 1. This method cannot allow unauthorized activities to get access to database and keep adversaries away from getting access to the data. The aim of the cryptography in database security is to keep data secret from unauthorized people, and defend against many types of security threats on database. In other words, using the cryptographic mechanisms makes the database more secure and confidential. For that, some strong encryption methods such as symmetric key and asymmetric key cryptosystems will be used to demonstrate the concept behind the cryptographic system for database security [3], [5], [6].

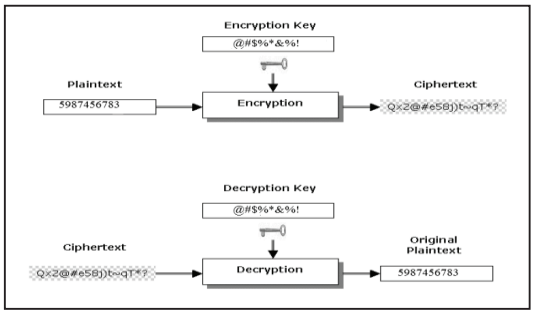

Fig 1: Encryption and Decryption Operation

A. Symmetric Key Cryptosystem

Encryption by using symmetric key involves taking a plaintext and converting it into encrypted form of data (cipher text) by using one secret key shared by the sender and the receiver [5]. The encryption function (E_K) takes a message M (i.e., plaintext) and a secret key K as inputs and produces encrypted form of data C (i.e., cipher text) as shown in the equation 1:

$$C = E_K(M) \qquad (1)$$

The decryption function D_K takes the cipher text C and the secret key K as inputs and produces the original message M as an output as shown in the equation 2:

$$M = D_K(C) \qquad (2)$$

The symmetric-key cryptosystem uses block cipher technique which encrypts data one block at a time. DES, AES and Triple DES are typical encryption algorithms that apply this technique. In the block cipher technique, any blocks of data smaller than the fixed block size must be padded, which is considered one of the main disadvantages [6]. Stream cipher is used in this kind of cryptography, which can encrypt one bit at a time. Comparing to the block cipher technique, the padding in the stream cipher technique is avoided. The key in symmetric encryption can be derived from passwords that can be remembered by people. Or, it can be randomly generated to offer maximum security. This algorithm tends to be very fast compared to asymmetric algorithms described in the next section.

B. Asymmetric Key Cryptosystem

Asymmetric encryption is a form of cryptosystem in which encryption and decryption are implemented using different keys. One is a public key and the other one is a private key. It is also known as public-key encryption. The public key includes pairs of keys, a public key which will be made openly available and a private key [5]. The data are encrypted by the sender's public key, and only the holder of the private key can decrypt it.

In this cryptographic algorithm, the encrypting process is more secure because it is difficult to derive the private key from the public key, and the private key does not need to be exchanged [5]. Generally, asymmetric encryption transforms public and private keys). Typically, the public key is used to encrypt data and only the private key can be used to do the decryption. For example, in MySQL database, the asymmetric key encryption is used to protect the data without exposing the decryption keys to unauthorized people. In this database system, data are represented as cipher text using the public key. The secret private key is encrypted by the user's password as another layer of protection using the asymmetric encryption. Therefore, when a user wants to access the data, the entered password will be used to get the private key (which is already encrypted); then the cipher text itself will be decrypted by the private key [16], [17]. RSA, ELGammal, Diffie- Hellman algorithms are examples of the public-key cryptography where they use different mechanisms for performing public-key cryptography.

III. ENCRYPTION ARCHITECTURE IN NETWORK DATABASE

In this section we will discuss the three dimensions of encryption that are application level encryption, database level encryption, and storage level encryption. The discussion will be supported by the cons and pros of each and some real examples.

A. Application Level Encryption (ALE)

The encryption and decryption operations are performed at the application layer that creates the data. The data are stored into and retrieved from the databases as encrypted data and can only be decrypted at the application layer. The encryption keys are separated from the encrypted data. As a result, the data is only accessible by authorized application users. Attackers who may get access to the database will face encrypted data that can only be decrypted at the application layer; and at the same time the data are only accessible by authorized application users. However, performing encryption/decryption to the application layer has several drawbacks. The application needs to be modified to adopt the encryption/decryption process. Additionally, it may open security threats, since the application may retrieve data that are larger than the normal, requested data due to the encryption process [3], [4], [7].

To secure the data stored in database, cryptography can be performed either inside or outside the database, at the application layer. It has been confirmed that performing encryption/decryption outside the database is more secure and protects the confidentiality of the information since it provides good End-to-End encryption [6]. Next we will discuss the implementation of the encryption at the application layer followed by the process of encryption/decryption at the application server.

1) Implementing Encryption outside Database

The main key in implementing encryption outside database is applying the encryption at the application level where only the sensitive data is encrypted and stored in the database. The data entered by online clients is encrypted to securely travel via a network to reach an application server to be processed or stored in the database. The application server will provide centralized encryption services for the entire database environment. Therefore, the data from the starting point is being encrypted till arriving to the final point. [2], [3], [6]. This method is called the End-to-End encryption between client and the application server that is illustrated in Figure 2 [6].

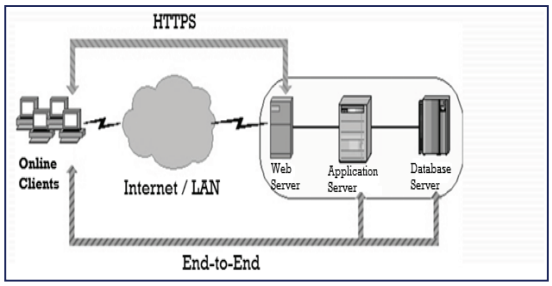

Fig 2: End- to- End Encryption.

2) Encryption/Decryption Process at Application Server

When the application server receives the data, it starts working to separate the encryption keys from the encrypted data that are stored in database and then injects encrypted data into the database. The encrypted data are stored on the basis of column and row in the database as shown in Figure 3. In the decryption operation, the encrypted data are retrieved from the database and then will be decrypted at the application server using keys that are stored in file storage at the application server not in the database. The approach of separation provides a high layer of protection for the database since the encryption keys do not leave the application server [6], [7].

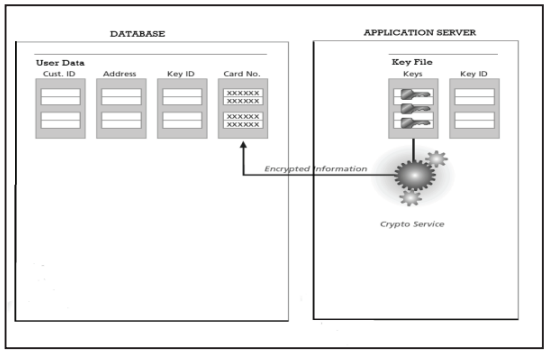

Fig 3: Encryption Process at Application Level

B. Database level Encryption (DLE)

The encryption technique here is performed with database itself. The architectural design is located somewhere between application encryption layer and storage encryption layer. The encryption can be done at columns and rows of database. The Oracle Advanced Security (OAS) has gathered Transparent Data Encryption (TDE) syntax with its Data Definitional Language (DDL) to encrypt the data directly in tables of database [3], [4], and [7]. The database with the support of TDE can automatically generate and mange the encryption keys to encrypt/decrypt the entire tables or some certain columns [3], [7]. Next we will give a summary of TDE and its encryption process.

Several enterprises attempt to obtain an efficient protection for their database systems such as e-Business (e.g., Amazon, eBay). SQL and Oracle design TDE as a new encryption feature to protect the entire database at rest without affecting the applications [11], [12]. TDE is an encryption technique used to perform real-time I\O encryption and decryption of the important information inside database. It is performed in database layer and uses the encryption keys stored in database for the encryption process. TDE protects data at rest by encrypting the physical data file, rather than the data itself [13]. The encryption of database file is performed at the page level. The pages in ciphered database are encrypted before they are written to the disk. As a result, TDE provides a highly protection for

the data, especially the data at rest without any modification for any existing applications [7], [11], [12]. In SQL server, TDE performs this operation and uses a database encryption key (DEK). This key is stored in database boot record and is executed as a symmetric key, which is secured by a certificate [10], [13].

C. Storage Level Encryption (SEL)

The storage level encryption is represented to encrypt the data in the subsystem of storage and thus protects the data at rest. This technique of encryption requires an operating system context to encrypt database files and the entire directories or the media files that reside on them [3], [7].

Because it uses subsystems that manage the encryption keys, the encryption process does not support the separation of tasks between the system administrator and DBA (database administrator). Therefore, this method can add a weakness of implementing the encryption since it requires Intrusion Prevention Systems (IPS) to monitor access to the keys themselves. Moreover, these subsystems don't know how the infrastructure of database works, thus the distinct encryption keys will be used for distinct clients. Obviously, we can observe that the encryption technique cannot be related with client's privileges. In the case of encrypting at storage level, the users who get access to operating system can typically see data, where is encrypted on the system, and the external key management can be observed by them. Thus, this will create a kind of difficulty in the management of these keys at the database files or table columns [7]. On the other hand, storage-level encryption has an advantage of preventing any changes to applications to protect the database [3].

1) Storing the Encrypted Character Data in Database

In the architecture of encryption at storage level, the layer of encryption and decryption is added and located after the DBMS and the application layers. Implementing this architecture of encryption will be more efficiency over the character data of query. The process works without changing the internal architecture of the applications and DBMS [3], [14].

Inside this model, there is a metadata module that designed within the encryption and decryption of storage layer and contains some mapping and transformation rules. This metadata is used to store the characteristic values of the encrypted data together with the encrypted data themselves. Additionally, the metadata is used to transform SQL statements into new SQL that can be applied on the encrypted data [14]. The storage scheme in this layer is extended to store the characteristic values of the encrypted data. When a relation schema is represented as

$$R (X_1, ..., Xr, ..., Xn)$$

where *Xr* is the field wants to be encrypted, the corresponding encrypted relation schema will be as

$$RE (X_1, ...XrE, ..., Xn, XrS)$$

where *XrE* is the encrypted space of this field and *XrS* is pairs coding functions of *Xr*.

This encrypted relation schema in storage of database will be stored in

$$RE (X_1, ...XrE, ..., Xn, XrS).$$

where *RE* is storage of encrypted value of *Xr* .

Since the storage schema is extended, *XrS* is index field that works as storage for the characteristic value of *Xr* in *R* [14].

IV. DATABASE VULNERABILITY

SQL injection is an attack on a website that targets the data residing in its database. This is a result of weakness of input validation, and the injection technique can exploits security vulnerability in the database layer of an application and gain direct access to its functions [8], [10]. The attackers attempt to insert and add arbitrary data, often database query, into string running by database. Many web applications take a user input that may be used to attack the application itself. For example, a user input may be used in the construction of a SQL query submitted to the application's database to change the database content, an example of a SQL injection attack is shown in Figure 4. Next we will discuss some of the types of SQL injection that may break the protection of databases. Then some of the solutions that secure the database by preventing SQL injection will be shown and discussed.

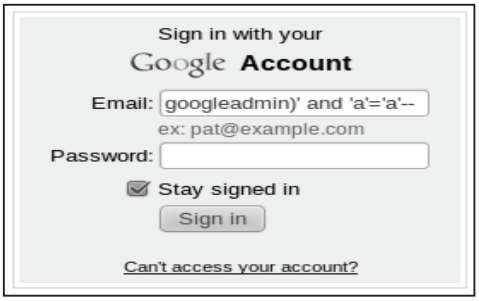

Fig 4: SQL injection.

A. Types of SQL Injection Vulnerabilities

1) Escape Characters

This type of SQL injection happens when a user input is not filtered from special symbols that are inserted into sentences form of SQL, which can lead to manipulate the original SQL statements used by the database for authentication purposes [15]. The following sentence can show this vulnerability:

$query = SELECT cardinfo FROM cartable, userstable WHERE username = ' + userName + ';

The query input is performed directly to call all records of the specified username from its tables of users (username). However, if the username is changed by the attackers with adding the special symbol such as (‘), for example, *blah' OR*

'x' = 'x', the statement will be inserted directly into SQL code within the web application as the following script

SELECT carinfo FROM cartable, usertable WHERE username = 'blah' OR 'x' = 'x'

This SQL code can enable the attacker to return the entire database and cause many problems for the application's users when their sensitive information is being stolen.

2) Incorrect Type Handling

This type of injection happens when the type of the user input does not match the type needed by the system. This usually happens when the programmer does not monitor the user input to see whether it matches the needed type or not. For example, this can happen when a field of numeric type (NUMERIC) in query syntax is used, and the programmer does not monitor the user input to see if the input is a numeric type or not. The following example illustrates the idea of incorrect type handling.

*SELECT * FROM info WHERE id = " + a_variable + ";*

Obviously, *a_variable* in the SQL statement above can be a number linked to the field *id*. However, if the field (*id*) has the type string; and the programmer does not check the user inputs whether they are strings or numbers, this field is possible to be manipulated by attackers through passing subversive sentences. For example: "*2; DROP TABLE users"*. As a result, the SQL statement will now become the following statement, which makes a severe change on the database, i.e., the user table will be deleted from the database.

*SELECT * FROM info WHERE id= 2; DROP TABLE users;*

B. *Solutions to Prevent SQL Injection*

1) Application Remediation

There are some protection methods that prevent SQL injection in most programming languages that target the Internet applications. For example, in the programming language Perl DBI, the task of DBI::quote is used by some symbols (ESCAPES SPECIAL CHARACTERS) instead of this symbol (') [8], [15]. We assume that the variable ($sql) holds a reference to a DBI object:

$query = $sql->car("SELECT * FROM users WHERE name = ".$sql->quote($user_name));

Perl DBI allows using placeholders (PLACE HOLDERS), and that allows you to bind (BIND) data to a statement separately to form a SQL statement. As for databases that do not support holders place originally, (DBI) can emulate them (EMULATES) automatically by applying function (DBI ::quote) to the values [18].

$query = $sql->prepare("SELECT * FROM users WHERE name = ?");$query->execute($user_name);

The interest of this method is that we do not need to remember the (DBI::quote) for each value, because they are binding appropriately, according to the database management system (DBMS) that we use.

2) Database Remediation

The development of security privileges on the databases is the simplest example of the protection of databases, and thus little of applications allow the user to delete a table or even a whole database. This strategy does not solve the problem of penetration SQL, but reduce the severity of the potential harm [9], [15].

Most databases offer the possibility of query processing in the database layer through stored procedures. This method provides several security benefits such as filtering user input and working under different privileges to restrict the actions to be within the scope of actions already saved in the stored procedures [10].

This method does not totally solve the problem of SQL injection, because if the user input is not parameterized or not filtered properly. For example, the following are two stored procedure [10].

(GET_PASSWORD (userName)) and (GET_USER (userName, password))

The hacker could insert the code into (GET_USER CALL) if the password is not written correctly as follows:

(GET_USER('admin', "|| GET_PASSWORD('admin') ||"))

As mentioned previously, the risks will be related to the use of multiple strings. For example, if we take a website that shows a list of grocery items for a user name (userName), the query syntax would be:

*SELECT * FROM grocery WHERE userName = '$ userName';*

It is possible that the user enters the following sentence to get all items belonging to all users:

*SELECT * FROM grocery WHERE userName =" OR userName IS NOT NULL OR userName ="";*

Also, if the special symbols (quotes) are not used, the risk of SQL injection is still possible, especially when the type of *userid* is unchecked as follows.

*SELECT * FROM sections WHERE userid = $ userid;*

Therefore, it is possible that the attacker enters the following sentence which does not contain the special symbols (quotes) and still gets all items belong to all users

userid = 22 OR userid is not null OR userid = 55;

which contains no quotes and produces the following query.

*SELECT * FROM sections WHERE userid = 22 OR userid is not null OR userid = 55;*

The most appropriate solution to this problem is the definition of the variable (*userid*) that has a numerical value only, such as:

if (ctype_digit($userid)) {die(" inappropriate characters in userid,");}

The function of *ctype_digit* will check if all of the characters in the provided *userid* are numerical or not.

V. CONCLUSION

This paper focuses on the security of networked databases. It provides an analysis of the cryptosystem architectures for networked databases. It mainly represents and discusses the process of encryption and decryption at the application, database and storage levels. Additionally, it highlights some of the types of SQL injection vulnerabilities and discusses some methods to prevent SQL injection that can help to overcome such a problem for the administrative beginners and prevent the adversaries from getting access to the data stored in networked databases.

REFERENCES

[1] E. Bertino, R. Sandhu, "Database security- concepts, approaches, and challenges," IEEE Transactions on Dependable and Secure Computing. Los Alamitos, CA, USA, vol. 2, pp. 2-19, January-March 2005.

[2] S. Imran, I. Hyder, "Security issues in databases," IEEE Computer Society, second international conference on future information technology and management engineering. Sanya, China. December 2009.

[3] L. Bouganim, Y. Guo, "Database encryption," INRIA Paris-Rocquencourt. 2009.

[4] U. Mattsson, "Database Encryption - How to Balance Security with Performance," Protegrity Corp. February 25, 2005.

[5] B. Sarah, "Data Encryption," The Parliamentary Office of Science and Technology. Postnote. London. October 2006.

[6] N. Arshad, S. Shah, A. Mohamed, A, Mamat, "The design and implementation of database encryption." International Journal of Applied Mathematics and Informatics, vol. 1, pp. 115-122, September 2007.

[7] T. Baccam, "Transparent data encryption: New technologies and best practices for database encryption." SANS, Oracle, April, 2010.

[8] K. Wei, M. Muthuprasnna, S. Kothari, "Preventing SQL Injection Attacks in Stored Procedure." Software Engineering Conference. Australian, 2006.

[9] S. W. Boyd, A. D. Keromytis, "SQLrand: Preventing SQL Injection Attacks." In Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference. 2004.

[10] Microsoft® SQL Server™.

[11] An Oracle White Paper. "Oracle Advanced Security Transparent Data Encryption Best Practices". Oracle. 2012.

[12] An Oracle White Paper. "Oracle Transparent Data Encryption for SAP. Oracle". Oracle. 2010.

[13] J. Magnabosco, "Transparent Data Encryption." A Technical Journal and Community Hub from Red Gate. March 2010.

[14] Z. Wang, J. Dai, W. Wang, B. Shi, "Fast query over encryption character data in database." Communications in Information and Systems. International Press. vol. 4, no. 4, pp. 289-300. 2004.

[15] C. Dougherty, "Practical Identification of SQL Injection Vulnerabilities."US-CERT. 2012.

[16] MySQL.

[17] R. Ramarakrishnan, J. Gehrke, " Database Mangment Systems," The McGraw- Hill, 2003.

[18] PerDBI.