RESEARCH ARTICLE OPEN ACCESS

SQL Injection Attack Prevention Using 448 Blowfish Encryption Standard

K. Rajeswari M.Sc, M.Phil.,
Associate Professor
Department Of Computer Science
Tiruppur Kumaran College for Women, Tirupur – 641 687
C. Amsaveni, M.Phil
Research Scholor
Department Of Computer Science, Tirupur-641 687
Tamil Nadu

ABSTRACT

SQL Injection is a method where the intruder injects a contribution to the SQL Query with a specific end and goal to change the structure of the Query proposed by the programmer and picking up the access of the database which results modification or deletion of the client's information. In the injection it misuses a security weakness (vulnerability) happening in database layer of an application. SQL injection attack is the most well-known attack in web based sites and application nowadays. Some malicious codes get inject to the database by unapproved clients and get the entrance of the database due to absence of information approval. Information validation is the most critical portion of programming security that is not appropriately secured in the outline period of programming advancement life-cycle bringing about numerous security vulnerabilities. This proposed technique displays the procedures for identification and avoidance of SQL injection attacks. There are no any known full verification protections accessible against such sort of assaults. In this paper some predefined strategy for identification and the some current systems of preventions are examined. In here we use the 448 bit Blowfish along with the some security algorithms to enhance the existing model, to prevent the SQL Injection attacks. In Existing model we use RC4 and Normal Blowfish to Encrypt and secure the web data from the SQL Injection attack but now we enhance using the 448 bit Blowfish Encryption technique with less execution overhead.

Keywords:- SQLIA, Vulnerability, Blowfish, DES, Piggy back queries

I. INTRODUCTION

The data theft has occurred recently in all those web applications, over internet that personally affects the web users so much it has been also covered by OWASP (Open Web Application Security Project), The project issues the security to web related application that in unbiased source information on the best practices, various methods to address the SQL injection attacks such as full scope of the problem or have the limitations that prevention of Injection Attacks (SQLIA), Detecting SQL fragments injected into a Web application has proven extremely challenging. There are several tacks enterprises can take – prevention, remediation, and mitigation. When implementing prevention and remediation efforts, the enterprise strives to develop secure code and/or encrypt confidential data stored in the database. However, these are not always available options. For Example, in some cases the application source code may have been developed by a third party and not be available for modification. Additionally, patching deployed code requires significant resources and time. Therefore rewriting an existing operational application would need to be prioritized ahead of projects driving new business. Similarly, efforts to encrypt confidential data stored in the database can take even longer and require more resources. Given today's compressed development cycles, and limited number of developers with security domain experience, even getting the code rewrite project off the ground could prove difficult. SQL injection vulnerabilities are caused by software applications that accept data from an un-trusted source (internet users), fail to properly validate and sanitize the data, and subsequently use that data to dynamically construct an SQL query to the database backing that application. The numerous SQLIA techniques used by attackers are based on the many statement structure combinations offered by SQL, and sometimes also take advantage of additional features in

specific DBMS implementations, particularly Microsoft's SQL Server. They pursue different goals at various levels, from allowing other techniques to be used (SQLIA escalation) to actually extracting database data. The resulting threats are various and range from system fingerprinting to Denial-of-Service (DoS) and theft of confidential information. SQL Injections Attacks thus threaten the confidentiality, integrity and availability of databases' data and structure, of their hosting systems and of their dependant applications, and as such greatly require the attention of application developers and the deployment of effective prevention solutions.

An SQL Inject ion Attack is probably the easiest attack to prevent, while being one of the least protected against forms of attack. The core of the attack is that an SQL command is app ended to the back end of a form field in the web or application front end (usually through a website), with the intent of breaking the original SQL Script and then running the SQL script that was inject ed into the form fie ld. This SQL injection most often happens when you have dynamically gene rated SQL within your front -end application. The se at tacks are most common with legacy Active Server Page s (ASP) and Hypertext Preprocessor (PHP) applications, but hey are still a problem with ASP.NET web-based applications. The core reason behind an SQL Injection attack comes down to poor coding practices both within the front-end application and within the database stored procedures. Many developers have learned better development practices since ASP.NET was released, but SQL Injection is still a big problem between the number of legacy applications out there and newer applications built by developers who didn't take SQL Injection seriously while building the application.

Structured Query Language ('SQL') is a textual language used to interact with relational databases. There are many varieties of SQL; most dialects that are in common use at the moment are loosely based around SQL-92, the most recent ANSI standard. The typical unit of execution of SQL is the 'query', which is a collection of statements that typically return a single 'result set'. SQL statements can modify the structure of databases (using Data Definition Language statements, or 'DDL') and manipulate the contents of databases (using Data Manipulation Language statements, or 'DML'). In this paper, we will be specifically discussing Transact-SQL, the dialect of SQL used by Microsoft SQL Server.

This statement will retrieve the 'id', 'forename' and 'surname' columns from the 'authors' table, returning all rows in the table. The 'result set' could be restricted to a specific 'author' like this:

```
select id, forename, surname from authors where forename = 'john' and
surname = 'smith'
```

An important point to note here is that the string literals 'john' and 'smith' are delimited with single quotes. Presuming that the 'forename' and 'surname' fields are being gathered from user-supplied input, an attacker might be able to 'inject' some SQL into this query, by inputting values into the application like this:

```
Forename: jo'hn
Surname: smith
```

The 'query string' becomes this:

```
select id, forename, surname from authors where forename = 'jo'hn' and
surname = 'smith'
```

They are easy to detect and exploit; that is why SQLIAs are frequently employed by malicious user for different reasons. E.g. financial fraud, theft, confidential data, deface website, sabotage, espionage, cyber terrorism, or simply for fun. Throughout 2010, Government, Finance and Retail verticals faced different, but equally important, outcomes. Attacks against Government agencies resulted in defacement in 26% of SQL injection attacks, while Retail was most affected by credit card leakage at 27% of SQL injection and finance experienced monetary loss in 64% of attacks. Furthermore, SQL Injection attack techniques have become more common more ambitious, and increasingly sophisticated, so there is a deep to need to find an effective and feasible solution for this problem in the computer security community. Detection or prevention of SQLIAs is a topic of active research in the industry and academia. To achieve those purposes, automatic tools and security system have been implemented, but none of them are complete or accurate enough to guarantee an absolute level of security on web application. One of the important reasons of this shortcoming is that there is lack of common and complete methodology for the evaluation either in terms of performance or needed source code modification which in an over head for an existing system. A mechanism which will easily deployable and provide a good performance to detect and prevent the SQL injection attack is essential one.

SQL Injection Attacks (SQLIA) Process

SQLIA is hacking technique which the attacker adds SQL statements through a web application's input field or hidden parameter to access to resources. Lack of input validation in web applications causes hacker to be successful. Basically SQL process structured in three phases:

i. An attack sends the malicious HTTP request to the web application.

ii. Create the SQL Statements.

\iii. Submits the SQL statements to the back end database

Problems occurred of SQLIA

The result of SQLIA can be disastrous because a successful SQL injection can read sensitive can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administrative operations on the Database (such as shutdown the DBMS), recover the content on the DBMS file system and execute commands (xp cmdshell) to the operating system. The main consequences of these vulnerabilities are attacks on:

i) Authorization Critical data that are stored in a vulnerable SQL database may be altered by a successful SQLIA, a authorization privilege.

ii) Authentication If there is no any proper control on username and password inside the authentication page , it may be possible to login to a system as a normal user without knowing the right username and/or password.

iii) Confidentially Usually databases are consisting of sensitive data such as personal information, credit card numbers and/ or social numbers. Therefore loss of confidentially is a big problem with SQL Injection vulnerability. Actually, theft of sensitive data is one of the most common intentions of attackers.

iv) Integrity By a successful SQLIA not only an attacker reads sensitive information, but also, it is possible to change or delete this private information..

Vulnerabilities

Insufficient Input Validation Input validation is an attempt to verify or filter any input for malicious behavior. Insufficient input validation will allow code to be executed without proper verification of its intention. Attacker taking advantages of insufficient input validation can utilize malicious code to conduct attacks.

Privileged account A privileged account has a degree of freedom to do what normal accounts cannot. Its action may also exempt from auditing and validation. This present vulnerability since a jeopardized privileged account, such as an administrator account, can compromise much more than what a jeopardized regular account can.

Description: Union query injection is called as statement injection attack. In this attack attacker insert additional statement into the original SQL statement. This attack can be done by inserting either a UNION query or a statement of the form ";< SQL statement >" into vulnerable parameter. The output of this attack is that the database returns a dataset that is the union of the results of the original query with the results of the injected query. For example, SELECT * FROM user WHERE id='1111' UNION SELECT * FROM member WHERE id='admin' --' AND password='1234';

Stored Procedure

Attack Intent: Performing privilege escalation, performing denial of service, executing remote commands.

Description: In this technique, attacker focuses on the stored procedures which are present in the database system. Stored procedures run directly by the database engine. Stored procedure is nothing but a code and it can be vulnerable as program code. For authorized/unauthorized user the stored procedure returns true/false. As an SQLIA, intruder input "; SHUTDOWN; --" for username or password. Then the stored procedure generates the following query: For example, SELECT accounts FROM users WHERE login= '1111' AND pass='1234 '; SHUTDOWN;--; This type of attack works as piggyback attack. The first original query is executed and consequently the second query which is illegitimate is executed and causes database shut down. So, it is considerable that stored procedures are as vulnerable as web application code.

Extra Functionality Extra functionalities meant to provide a broader range of vulnerability, since combinations of this functionality may result in unintended actions. For example, xp_cmdshell is meant to provide users with a way executing operating system commands, but commonly used to added unauthorized users into the operating system.

Database Server Fingerprint The database server fingerprints contains information about the database system in use. It indentifies the specific type and version of the database, as well as the corresponding SQL language dialect. A compromise of this asset may allow attackers to construct malicious code specifically for the SQL language dialect in question.

Database Schema The database schema describes the server's internal architecture Database Structure information such as table names, size and relationships are defined in the data schema. Keeping this asset private is essential in keeping the confidentiality and integrity of the database data .A compromise in the database schema may allow attackers to know the exact structure of the database, including table, rows and column headings.

Database Data T he database data is the most crucial asset in any database system. It contains the information in the tables described in the database schema, such as prices in an online store, personal information of clients, administrator passwords, etc. A compromise in the database data will usually result in failure of the system's intended functionality, thus, its confidentiality and integrity must be protected.

Network A network interconnects numerous hosts together and allows communication between them. A compromise in a network will most likely compromise every host in the network. Some networks may also be interconnected with other networks, furthering the potential damage, should an attack be successful.

Piggy-backed Query

Attack Intent: Extracting data, adding or modifying data, performing denial of service, executing remote commands

Description: In the piggy-backed Query attacker tries to append additional queries to the original query string. On the successful attack the database receives and executes a query string that contains multiple distinct queries. In this method the first query is original whereas the subsequent queries are injected. This attack is very dangerous; attacker can use it to inject virtually any type of SQL command. For example, SELECT * FROM user WHERE id='admin' AND password='1234'; DROP TABLE user; --'; Here database treats above query string as two query separated by ";" and executes both. The second sub query is malicious query and it causes the database to drop the user table in the database.

II. RELATED WORK

Reviews and analysis of SQL injection with various prevention mechanisms

In Random4: An Application Specific Randomized Encryption Algorithm to prevent SQL injection, the authors proposed a solution to the problem of unauthorized access to the database by preventing it using an encryption algorithm based on randomization. This approach is based on SQLrand and randomization algorithm is used to convert input into a cipher text incorporating the concept of cryptographic salt. However, the main flaws in this approach are, Use of lookup table is not efficient way, cannot handle second order SQL injection attack and it also requires more space to store the look up table.

Gaurav Shrivastava and Kshitij Pathak proposed a model for SQL injection prevention using tokenization. In this paperl, they extract the where clause from the input query and put the remaining query (after where clause) in a temporary variable. They applied tokenization only on the remaining query which converts the tokens into hierarchical form such as left and right child. They performed validation of each token by comparing the value of left and right child to the root condition This model prevents all type of SQL injection attacks which are occurred only after the where clause .

Debabrata Kar and Suvasini Panigrahi proposed a lightweight approach to prevent SQL Injection attacks by a novel query transformation scheme and hashing. They used a novel query transformation scheme that transforms a query into its structural form instead of the parameterized form. In order to store the transformed queries, they proposed to apply a suitable hashing function to generate unique hash keys for each transformed query. This approach can also be easily implemented on any language or database platform with little modification. However, this approach cannot prevent second order SQL injection attempts since the parameter values (especially the string values) are

removed during the transformation process. Another drawback is that for query transformation, they used query transformation scheme lookup table.

In 2011, Kai-Xiang Zhang et al. suggest SQL injection attacks, a class of injection flaw in which specially crafted input strings leads to illegal queries to databases, are one of the topmost threats to web applications. Based on their observation that the injected string in a SQL injection attack is interpreted differently on different databases, they propose a novel and effective solution TransSQL to solve this problem. TransSQL automatically translates a SQL request toa LDAP-equivalent request. After queries are executed on aSQL database and a LDAP one, TransSQL checks the difference in responses between a SQL database and a LDAPone to detect and block SQL injection attacks. Their Experimental results show that TransSQL is an effective and efficient solution against SQL injection attacks.

In 2012, RamyaDharam et al. present a framework which can be used to handle tautology based SQL Injection Attacks using post-deployment monitoring technique. Their framework uses two pre-deployment testing techniques i.e. basis path and data flow testing techniques to identify legal execution paths of the software. Runtime monitors are then developed and integrated to observe the behavior of the software for identified execution paths such that their violation will help to detect and prevent tautology based SQL Injection Attacks. I

In 2012, XI-Rong Wu et al. proposed a new method named k-centers (KC) to detect SQL injection attacks (SQLIAs). The number and the centers of the clusters in KC are adjusted according to unseen SQL statements in the adversarial environment, in which the types of attacks are changed after a period of time, to adapt different kinds of attacks. The experimental results show that the proposed method has a satisfying result on the SQLIAs detection in the adversarial environment

III. BACKGROUND STUDY ATTACKS

Tautology Attack: The main objective of tautology-based attack is to inject code in conditional statements so that they are always evaluated as true. Using tautologies, the hacker wishes to either bypass user authentication or insert inject-able parameters or extract data from the database. A typical SQL tautology has the form, where the comparison expression uses one or more relational operators to compare operands and generate an always true condition. Bypassing authentication page and collecting data is the most common example of this kind of attack.

Example Query:

```
SELECT * FROM user WHERE id='1' or
_1=1'-_AND password='1234'; –or 1=1
```

the most commonly known tautology.

Logically Incorrect Query Attacks: The main objective of the Illegal/Logically Incorrect Queries based SQL Attacks is to gather the information about the back end, database of the Web Application. When a query is rejected, an error message is returned from the database which includes useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application. In fact attacker injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical error by purpose. In this example attacker makes a type mismatch error by injecting the following text into the input field:

1. Original URL:http://www.toolsmarketal.com/veglat/?id_nav=223455

2. SQL Injection: http://www.toolsmarket-al/veglat/?id_nav=223455‘ 3. Error message showed: SELECT name FROM Employee WHERE id=223455\‘. From the message error we get the name of table and fields: name; Employee; Id By the gained information attacker can organize more perfect attacks. The Illegal/Logically Incorrect Queries based SQL attack is considered as the basis step for all the other attacking techniques.

Union Query: attackers join injected query to the safe query by the word UNION and then can get data about other tables from the application

Example: Following executed from the server:

```
SELECT name, phone FROM tbl_user WHERE
userid=$id1
```

By injecting the following Id value into:

```
$id1= 1 UNION ALL SELECT credit Card
Number,
1 FROM Credit CardTable
```

Then we will have the following query:

```
SELECT name, phone FROM tbl_user WHERE
userid1 =1
UNION ALL SELECT creditCardNumber, 1
FROM
Credit CardTable
```

This will join the result of the original query with all the credit card users to the attacker. The proposed implemented system contains the mechanisms, which will protect the web application from the above discussed SQL injection attacks.

IV. PROPOSED METHODOLOGY

A. AES Encryption and Decryption The attributes and data in the input query are encrypted using AES (Advanced Encryption Standard) algorithm which is fast, and requires little memory. Once the query is arrived at server side, which is decrypted by using the same key and in turn converts into various token which are stored in to another dynamic table. The performance comparison of cipher text over normal text shows that, cipher text is very difficult and time consuming to crack.

Query tokenization Query tokenization technique converts the input query into various tokens. These tokens are generated by detecting single quote, double dashes and space in an input query. All string before a single quote, before double dashes and before a space constitutes a token. Tokenization process executes in following four essential steps and then forwarded to the server side.

Step 1: Process the input query by replacing all the unnecessary characters which are used to make attacks on query.

Step 2: Detect Single Quote, Double Quote, Double slashes and space in the input query.The fig.3. shows how tokens are formed by detecting spaces ,single quote and double dashes in input query for the below given input query. " SELECT eid , ename FROM Employee WHERE salary > 2000 "

Step 3: Break the input query into various useful tokens.

Step 4: Store the tokens in a Dynamic table.

Step 5: Query Forwarding – After tokenization, the encrypted input query and dynamic token table are forwarded to server.

Comparison with existing system

SQL Injection Types	SQLIA Prevention Technique	
	RC4	Blowfish
By pass Authentication	Prevented	Prevented
Unauthorized knowledge of Database	Prevented	Prevented
Injected Additional query	Not Prevented	Prevented
Second order SQL Injection	Not Prevented	Not Prevented

Table 1: Comparison of existing algorithms

We propose a model for preventing SQL injection attacks by combining two well-known encryption techniques Blowfish (Symmetric key encryption) and RC4 (Asymmetric key encryption) at different levels of the proposed model.

In the proposed scheme, access to the database will be provided only by the server to all the authenticated users. If a new user wants to access the database he will have to register himself with the server. During the registration process, we require every new user to provide username and password.

Along with its regular process of checking the availability of user name, on successful completion, the server generates user key which is hexadecimal value of the password and stores it in the user table.

A. RC4 Methodology with SQL Injection

RC4 Algorithm

RC4 is a stream cipher, symmetric key algorithm. The same algorithm is used for both encryption and decryption as the data stream is simply XORed with the generated key sequence. The key stream is completely independent of the plaintext used. It uses a variable length key from 1 to 256 bit to initialize a 256-bit state

table. The state table is used for subsequent generation of pseudo-random bits and then to generate a pseudorandom stream which is XORed with the plaintext to give the ciphertext. The algorithm can be broken into two stages: initialization, and operation. In the initialization stage the 256-bit state table, S is populated, using the key, K as a seed. Once the state table is setup, it continues to be modified in a regular pattern as data is encrypted. The steps for RC4 encryption algorithm is as follows:

- o be encrypted and the selected key.
- Create two string arrays.
- Initiate one array with numbers from 0 to 255.
- Fill the other array with the selected key.
- Randomize the first array depending on the array of the key.
- Randomize the first array within itself to generate the final key stream.
- XOR the final key stream with the data to be encrypted to give cipher text

RC4 is a state cipher which is also termed as stream cipher. RC4 was designed by Ron Rivest. In this plain text digits are encrypted and processed one at a time. The processing speed is higher than block ciphers. RC4 algorithm is well suited for real time processing. In RC4 both encryption and decryption is performed using the same algorithm. The RC4 algorithm is of two stages:

•Initialization –Data encryption using a respective key, Creating arrays, Assigning values and selected key to the arrays.

•Operation – Swapping and XOR the final output to obtain the cipher text

Performance Evaluation of RC4 Algorithm

Query	Encryption Time	Decryption Time
Encrypted query for username and password =chinchu	528ns	528ns
Encrypted query for username and password = ajiths	518 ns	518 ns

Table 2: Performance in nano seconds

The above stated performance evaluation of RC4 algorithm is very clear that it works on 528 nano seconds to encrypt and 518 nano seconds to decrypt the text, from this encryption the SQL injection attacks possibility is less but the percentage of SQL Injection attack is very high, because the hacker hacks the programmer before the encrypts the text or after the encryption, also the hacker inject the code using their techniques.

B. Blowfish Methodology

Blowfish is a symmetric block cipher that can be effectively used for encryption and safeguarding of data. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for securing data. Blowfish was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms. Blowfish is unpatented and license-free, and is available free for all uses.

Blowfish is a variable-length key block cipher. It is suitable for applications where the key does not change often, like a communications link or an automatic file encryptor. It is significantly faster than most encryption algorithms when implemented on 32-bit microprocessors with large data caches.

Blowfish Algorithm is a Feistel Network, iterating a simple encryption function 16 times. The block size is 64 bits, and the key can be any length up to 448 bits. In such a way we need to use 448 bits of encryption fully at each time of encryption because 448 bit encryption not allows an hacker to inject the query in encrypted format and also here the technique is followed that after we analyze the data for encryption we need to check the round of feistel cipher, because the cipher made the blowfish bit encryption very strong one. Although there is a complex initialization phase required before any encryption can take place, the actual encryption of data is very efficient on large microprocessors.

Fiestel Network

A Feistel network is a general method of transforming any function (usually called an F function) into a permutation. It was invented by Horst Feistel and

has been used in many block cipher designs. The working of a Feistal Network is given below:

- Split each block into halves
- Right half becomes new left half
- New right half is the final result when the left half is XOR'd with the result of applying f to the right half and the key.
- Note that previous rounds can be derived even if the function f is not invertible.

Blowfish Algorithm

Blowfish is a variable-length key, 64-bit block cipher. The algorithm consists of two parts: a key-expansion part and a data- encryption part. Key expansion converts a key of at most 448 bits into several subkey arrays totaling 4168 bytes.

Data encryption occurs via a 16-round Feistel network. Each round consists of a keydependent permutation, and a key- and data-dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookups per round.

Blowfish uses a large number of subkeys. These keys must be pre-computed before any data encryption or decryption.

Generating the Subkeys

The subkeys are calculated using the Blowfish algorithm:

- Initialize first the P-array and then the four Sboxes, in order, with a fixed string. This string consists of the hexadecimal digits of pi (less the initial 3): P1 = 0x243f6a88, P2 = 0x85a308d3, P3 = 0x13198a2e, P4 = 0x03707344, etc.

- XOR P1 with the first 32 bits of the key, XOR P2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P14). Repeatedly cycle through the key bits until the entire Parray has been XORed with key bits. (For every short key, there is at least one equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc., are equivalent keys.)

- Encrypt the all-zero string with the Blowfish algorithm, using the subkeys described in steps (1) and (2). – Replace P1 and P2 with the output of step (3).

- Encrypt the output of step (3) using the Blowfish algorithm with the modified subkeys. – P3 and P4 with the output of step (5).

- The process, replacing all entries of the P array, and then all four S-boxes in order, with the output of the continuously changing Blowfish algorithm.

V. RESULTS AND DISCUSSION

This paper has presented a lightweight method to prevent SQL injection attacks by applying query tokenization technique to convert SQL queries into number of useful tokens and then encrypting the table name, fields, literals and data on the query using 448 bit Blowfish Encryption algorithm. This approach avoids memory requirements to store the legitimate query in repository and facilitates fast and efficient accessing mechanism with database. Our experimental results show that this approach can effectively prevent all types of SQL injection attempts. This approach does not require major changes to application code and has negligible effect on performance even at higher load conditions due to its low processing overhead. It can also be easily applied to any other language & database platform without major changes. Further explore on the query transformation scheme is needed to make use of new encryption algorithm for preventing SQL injection attacks.

Total Database Requests sent by Client	Injected Queries	Valid Queries
600	90	430
800	100	600
1000	175	825

Table 3: Injected and Valid Queries

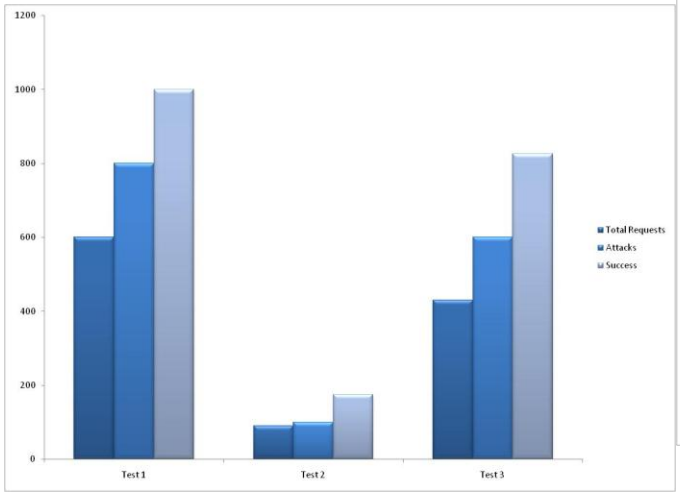

Fig 1 : Comparison of Injected and Valid queries

From the above chart shows that the total requests, attacks made, and success of the message, This application is experimented with number of requests in three different iterations, the basis of different SQL injection type like Bypass authentication, Unauthorized Knowledge of Database, injected additional query and Second order SQL Injection.

Comparison of Query Encryption in 448 bit Blowfish

Query	Encryption Time	Decryption Time
Encrypted query for username and password =chinchu	528ns	528ns
Encrypted query for username and password = ajiths	518 ns	518 ns
Encrypted query for username and password = chinchu	300 ns	300 ns
Encrypted query for username and password = ajiths	300 ns	300 ns

Table 4: Comparison of query encryption in 448 bit Blowfish

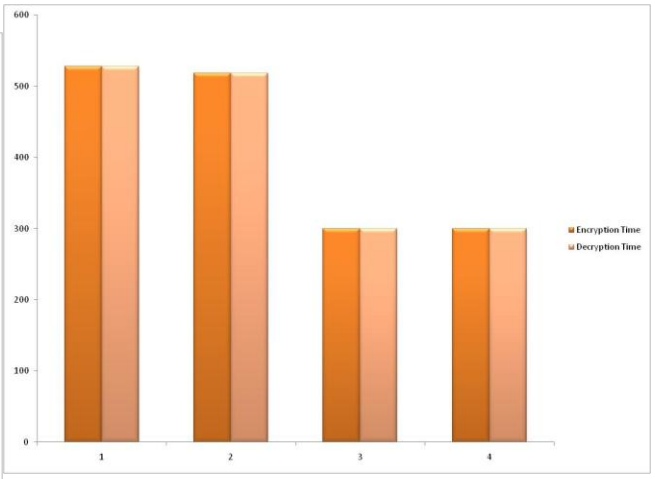

Fig 2: Comparison of Algorithm using Nanoseconds

Comparison of Algorithm

Algorithm	Injected Queries	Valid Queries
RSA	90	430
RC4	100	600
148 Bit Blowfish	175	825
448 Bit Blowfish	78	900

Table 5: Algorithm comparison in Query validation

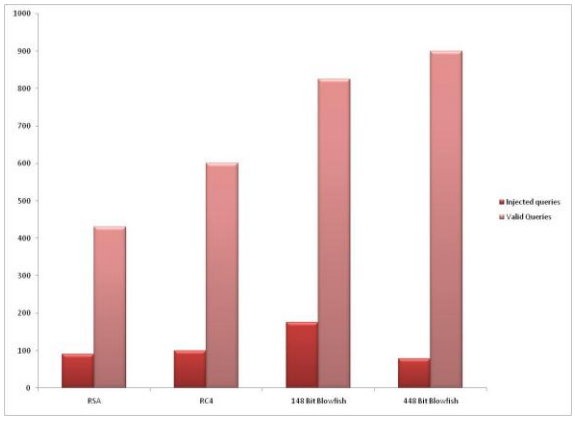

Fig 3: Comparison of Algorithms

The proposed a new approach that is completely based on the hash method of using the SQL queries in the web based environment, which is much secure and provide the prevention from the attackers SQL. But, our proposed strategy requires the alterations in the design of existing schema database and a new guideline for the database user before writing any new database. Through these guidelines, we found the effective outcomes in SQL injections Preventions. SQL injection attacks make the database vulnerable to unwanted access by non-reliable users that may not be good in terms of security. A secure database needs to restrict its user's activity according to the authentication of the user in order to work efficiently. In our scheme the proposed authentication process ensures user authenticity and efficient SQL query generation and in turn efficient database access and usage. The user name and SQL query both are encrypted and the query is only executed after the authenticity of the user is verified, hence making the process highly secured.

VI. CONCLUSION

Web applications need protection in their database to ensure security. SQL injection attacks allow attackers to spoof identity, tamper with existing data, can cause repudiation issues such as voiding transactions or even changing balances, allow the complete disclosure of all data, destroy the data or make it otherwise unavailable, and become administrators of the database server. If an application fails to properly construct SQL statements it is possible for an attacker to alter the statement structure and execute unplanned and potentially hostile commands. The longer key size is more secure but the encryption time and decryption speed is slow. In order to overcome this problem in Blowfish algorithm reducing of two S-boxes will increase the speed and provide the better security to data. The main advantage of optimized Blowfish is that the execution time is reduced to 0.2 milliseconds and the throughput is increased to 0.24bytes/milliseconds compare than original algorithms. The main objective was to analyze the performance of the most popular symmetric key algorithms in terms of Authentication, Flexibility, Reliability, Robustness, Scalability, Security, and to highlight the major weakness of the mentioned algorithms, making each algorithm's strength and limitation transparent for application.

REFERENCES

[1] Bruce Schneier, "The Blowfish encryption algorithm9", Dr. Dobb's Journal of Software Tools, 19(4), p. 38, 40, 98, 99, April 1994.

[2] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao. A Static Analysis Framework for Detecting SQL Injection Vulnerabilities, COMPSAC 2007, pp.87-96, 24-27 July 2007.

[3] Mei Junji, An approach for SQL injection vulnerability detection. Sixth International Conference on Information Technology, (2009): New Generations: pp. 1411-1414.

[4] R. Ezumalai, G. A. (2009). Combinatorial Approach for Preventing SQL Injection Attacks.2009 IEEE International Advance Computing Conference (IACC 2009). Patiala, India: pp.1212-1217.

[5] Tingyuan Nie Teng Zhang, A study of DES and Blowfish encryption algorithm, Tencon IEEE Conference, 2009.

[6] Shaukat Ali, Azhar Rauf, Huma Javed "SQLIPA : An authentication mechanism Against SQL Injection".

[7] W.G.J. Halfond, A. Orso, "AMNESIA: analysis and monitoring for Neutralizing SQL-injection attacks," 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA, 2005, pp. 174–183.

[8] Asha. N, M. Varun Kumar,Vaidhyanathan.G of Anomaly Based Character Distribution Models in the,"Preventing SQL Injection Attacks", International Journal of Computer Applications (0975 – 8887) Volume 52– No.13, August 2012

[9] Debabrata Kar, Suvasini Panigrahi, Prevention of SQL Injection Attack Using Query Transformation and Hashing, IEEE International Advance Computing Conference (IACC),2013.

[10] Avireddy. S, Perumal.V, Gowraj.N,Kannan R.S, Thinakaran.P, Ganapthi .S, Gunasekaran J.R, Prabhu.S, Random4: An Application Specific Randomized Encryption Algorithm to prevent SQL injection, ieee transactions on communications, vol. 60, no. 5, may 2012.

[11] Kai-Xiang Zhang, Chia-Jun Lin, Shih-Jen Chen, Yanling Hwang, Hao-Lun Huang, and Fu-Hau Hsu, "TransSQL: A Translation and Validation-based Solution for SQL-Injection Attacks", First International Conference on Robot, Vision and Signal Processing, IEEE, 2011

[12] Halfond W. G., Viegas, J., and Orso, A., A Classification of SQL-Injection Attacks and Countermeasures. In Proc. of the Intl. Symposium on Secure Software Engineering, Mar. 2006.

[13] Kemalis, K. and T. Tzouramanis. SQL-IDS: A Specification-based Approach for SQL injection Detection. SAC'08. Fortaleza, Ceará, Brazil, ACM 2008, pp. 2153-2158.

[14] Boyd S.W. and Keromytis, A.D., SQL rand: Preventing SQL Injection Attacks. Proceedings of the 2nd Applied Cryptography and Network Security (ACNS'04) Conference, June (2004), pp. 292–302.

[15] Shubham Shrivastava, Rajeev Ranjan Kumar Tripathi, Attacks Due to SQL injection & their Prevention Method for Web-Application, International Journal of Computer Science and information technologies, Vol 3 (2), pp.3615-3618, 2012.

[16] Sruthi Bandhakavi and Prithvi Bisht Preventing SQL Injection Attacks using Dynamic Candidate Evaluations, Alexandria, Virginia, USA, 2007

[17] R.Ezumalai (2009), An Combinatorial approach for SQL injection detection, IEEE

[18] Mahima Srivastava, Algorithm to prevent back end database against SQL Injection attacks, IEEE, 2014.

[19] RSA Laboratories, "RC6 Block Cipher", 2012, Historical: RSA Algorithm: Recent Results on OAEP Security: RSA Laboratories submissions

[20] W. G. J. Halfond, et al., "A Classification of SQL-Injection Attacks and Countermeasures," in Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA,2006.

[21] C. Bockermann, et al., "Learning SQL for Database Intrusion Detection Using Context-Sensitive Modelling (Extended Abstract)," in 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA '09), Berlin, Heidelberg, 2009, pp. 196--205

[22] K. Kemalis and T. Tzouramanis, "SQL-IDS: a specification-based approach for SQL-injection detection," in Proceedings of the 2008 ACM symposium on Applied computing (SAC'2008), New York, NY,USA, 2008, pp. 2153--2158.