ISSN: 1857-7881 (Print) e - ISSN 1857-7431

ESJ Natural/Life/Medical Sciences

Prevention of SQL Injection Attack Using Blockchain Key pair based on Stellar

Jeremiah O. Abimbola,
Department of Computer Science, Changchun University of Science and Technology,Jilin, China

Chen Zhangfang,
Department of Computer Science, Changchun University of Science and Technology,Jilin, China

Submitted: 18 August 2020
Accepted: 02 December 2020
Published: 31 December 2020

Corresponding author:
Jeremiah O. Abimbola

DOI: 10.19044/esj.2020.v16n36p92

Cite as:
Abimbola J.O, Zhangfang C.(2020). Prevention of SQL Injection Attack Using Blockchain Key pair based on Stellar. *European Scientific Journal, ESJ, 16 (36)*, 1. https://doi.org/10.19044/esj.2020.v16n36p92

Abstract
Currently, SQL injection is the most common attack on web applications where malicious codes are injected into the database by unauthorized users using user input fields and this could lead to data loss or in a worst case, to database hijacking; a situation no database administrator or web developer ever wants to experience. Two of the most recent types of these attacks are first-level and second-order attacks. A lot of researches have been done in this area, some of which are outstanding and capable of preventing first level attack but not second order attack. In order to improve the quality of protections, a new method is proposed in this paper to minimize the level of attack on databases by using stellar blockchain keypair. Using string manipulation on user inputs, the client application randomized the SQL query and sends it to the proxy server, the proxy server, in turn de-randomizes it with the help of the private key and sends the de-randomized query to the database server for processing and the overhead time is estimated and analyzed. This method proved to be more than 50% effective compared to previous methods using the same model. It also shows strengths in terms of processing and computational time. Experimental implementation and simulation using the stellar keypair demonstrates that the model presented is capable of detecting and preventing SQLIA all forms of SQL injection attacks including the second-order injections.

Subject: Computer Science

Keywords: SQL Injection, Attack, Database, keypair, Security

Introduction

Many institutions use dynamic database web applications to build a collaborative environment and provide better services to their customers. For example, educational institutions rely heavily on databases containing very sensitive student records to make informed decisions. A single bridge of any record caused by an attack can potentially result in a wrong or bad decision ultimately. There are many attacks threatening database security such as static leakage, linkage leakage, dynamic leakage, spoofing and the most common one; SQL injection attacks. (Khaleel Ahmad, 2010), tagged SQLIA in this study. SQLIA endangers the confidentiality, integrity, functionality and availability of databases of any web application. In addition, they are the most effective method for illegally collecting data from the database, through which hacker can get access to the database and steal sensitive information(Md. Fazlul Haque, 2017). Consider an example of a login page where a legitimate user enters the username and password to enter a secure page to view personal details or upload his comments on a social media site. When the user submits the data, the SQL query is generated and submitted to the database for verification. If it is valid, the user is allowed access to the system. This means that there is a communication between the login page and the database to verify the combination of the username and password which results to access granted upon verification. Using SQL Injection, the hacker may enter specially created SQL commands to bypass the validation of the login form to view the script. (Panda & Ramani, 2013; Singh, 2017; Wang Degao, 2019). This is only possible if the inputs are not properly sanitized (i.e. made invulnerable) and sent directly to the database via the SQL query. SQL Injection vulnerabilities provide an attacker with the means to expose a database. (Faker, Muslim, & Dachlan, 2017; Lawal, Sultan, & Shakiru, 2016; M. & Amsaveni, 2016). The impacts of SQLIA are very high which includes but not limited to:

I.Confidentiality: Most time, databases contain very sensitive data such as user credit card details, social security number and so on. Therefore loss of confidentiality is a major problem with SQL injection vulnerability as unauthorized users can gain access to crucial information

II.Integrity: Successful SQL injection attack permits unauthorized external sources to make modifications that is, private information can be read, changed or deleted by the attacker.

III.Authorization: Sensitive data stored in a vulnerable SQL database may be altered or attacker can gain elevated privileges.

IV.Authentication: Poorly written server-side codes could open up the database for attackers to gain access. For example, SQL codes that do not properly validate username and passwords could give unauthenticated access to attackers without prior knowledge of the password or username.

V.Functionality: SQL injection attack could partially or fully corrupt the intended function of any SQL database. Every database must be able to handle concurrent processing to enable simultaneous access, data sharing and consistent updates for users.

There are many forms of these attacks, some of which are Bypass Authentication - using tautology, Unauthorized Knowledge of Database - using illegal/incorrect Queries, Unauthorized Remote Execution of Procedure, Injected Additional Query - Using Piggy-Backed Queries and Injected Union Query as discussed in (Elshazly, Fouad, Saleh, & Sewisy, 2014; Saravana, 2014; Shrivastava & Tripathi, 2012; Sun, Wei, Liu, & Lau, 2007).

Second Order Attacks

Proxy Server Models

(Elshazly et al., 2014) suggested a method to solve SQLIA by introducing the concept of a proxy server. The proxy server is placed in between the two communicating devices. This allowed for the filtering of possible SQL-injection attempts.

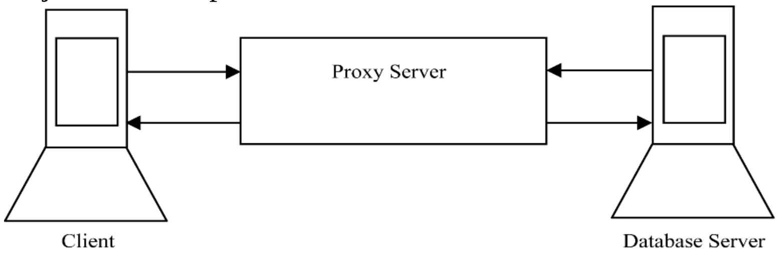

Figure 1. Proxy server Architecture Model

The process involved; analyzing the structure of the SQL query commands, building a parser that will check allowable patterns of SQL statements, constructing a list of common SQL commands, creating a proxy server that will alert the database administrator of possible SQL injection commands, preventing SQL injection attack on the database using the proxy server and proving that the SQL injection can be prevented using the filter developed to work on the proxy server. While this method seemed to work as at the time this research was carried out, there are disadvantages with this method in that it can create false positives; this means that legitimate words from variables can also be filtered out in the filtering process. Also, this method cannot work if the data is encrypted because the strings cannot be viewed in plain text without decryption. The use of a randomized key on SQL keywords was later introduced by (Boyd

& Keromytis, 2004) and (Perkins et al., 2005). The random key length was thirty-two bytes. The implementation of this technique involves building a proof of concept; a proxy server that sits between the client and the database server. If an SQL injection attack has occurred, the proxy's parser will fail to recognize the randomized query and will reject it. For example, the query on the left side then becomes the query on the right once the random key has been added to every SQL keyword in the query.

```
Select gender, avg(age)          select123 gender, avg123(age)
  From cs101.students              from123 cs101.students
 Where dept = %d                      where123 dept = %d
  group by gender                  group123 by123 gender
```

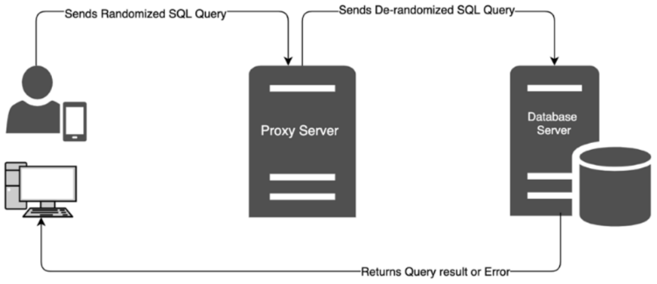
Sends Randomized SQL Query
Sends De-randomized SQL Query
Proxy Server
Database Server
Returns Query result or Error

Figure 2.SQLRand Model

The only setback for this technique is in the length of the key used as it is very susceptible to brute force attack as also seen in (Patil, Laturkar, Athawale, Takale, & Tathawade, 2017). A dynamic technique introduced by (Alazab & Khresiat, 2016; Gupta et al., 2018) using Normal use model which is straightforward, simple to execute and very compelling in avoiding SQL injection attacks, however, there are two major challenges with this model; decreasing the size of the achieved query repository and performing quick and efficient comparison at runtime.

Recently, (Appiah, Opoku-Mensah, & Qin, 2017) proposed a solution for SQL injection attacks by integrating the fingerprinting method and Pattern Matching to distinguish genuine SQL queries from malicious queries. The framework monitors and compares SQL queries to the database against a dataset of signatures from known SQL injection attacks. If the fingerprint method cannot determine the legitimacy of the query on its own, then the Aho Corasick algorithm is invoked to ascertain whether attack signatures appear in the queries.

Research Methodology

A new prevention mechanism is introduced to combat SQL injection attacks using Stellar Keypair algorithm. The keypair consists a public and private key. This method mimics the SQLRand method developed by (Boyd & Keromytis, 2004) using a random key on every SQL keyword, but this time, we're not just using a random key. Instead, we will use the public key and then verify the public key with the private key later on in the query process. To achieve this, we have decided to use a popular concept in programming known as separation of concerns; thus having three (3) tier architecture.

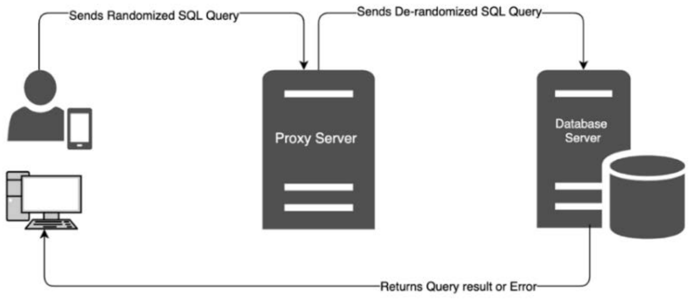
Sends Randomized SQL Query
Sends De-randomized SQL Query
Proxy Server
Database Server
Returns Query result or Error

Figure 3.Three-tier Architectural Model

Stellar Keypair

Stellar is a blockchain that tens of thousands of people use every day. It is decentralized, open-source, and developer-friendly, so anyone can issue assets, settle payments, and trade. It uses Ed25519 public-key signature system. From Stellar, only the Keypair is needed. In public-key cryptography, Edwards curve Digital Signature Algorithm (EdDSA) is a digital signature scheme using a variant of Schnorr signature based on Twisted Edwards curves (SHA512 and Curve25519). It is designed to be faster than existing digital signature schemes without sacrificing security. Public keys are 256 bits in length and signatures are twice that size.
To make it more difficult for attackers to brute force the entire process, the private key is used to validate the public key. First, we design a proxy server (an external server) that is located between the client and the database server where the keypair is generated. Since SQL keywords don't change, we can list them all out in an array to work with them further. The client requests the public key from the proxy server, joins the public key to every SQL keyword used in the query, splits the entire SQL statement including user inputs into tokens, checks every token that all SQL keywords have the public key, checks if there is a private key for that public key. If yes, send the full SQL statement to the DB server.

Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) can be used to build digital signature algorithms with a smaller key size than the Digital Signature

Algorithm (DSA) with the same level of security(Dinu et al., 2015; Romailler & Pelissier, 2017). With security in mind, such algorithms are generally based on the Discrete Logarithm Problem (DLP), currently, the best known algorithms to solve this problem over elliptic curves are less efficient than ones over finite groups. To provide security in the embedded ecosystem, the adoption of ECC was important where resources are constrained. The most widely used signature algorithm is ECDSA.

Keypair Generation

An entity *A*'s key pair is associated with a particular set of EC domain parameters D. This association can be assured cryptographically (e.g., with certificates) or by context (e.g., all entities use the same domain parameters) (Don Johnson, 2001). The entity *A* must have the assurance that the domain parameters are valid

Each entity *A* does the following;

1. Select a random or pseudorandom integer d in the interval [1, n-1]
2. Computer $Q = dG$.
3. A's public key is Q; A's private key is d.

Algorithm 1 EdDSA Signature

```
Require: M, $(h_0, h_1,...,h_{2b-1})$, $B$ and $A$
1: $a \leftarrow 2^{b-2} + \sum_{3 \le i \le b-3} 2^i h_i$
2: $h \leftarrow H(h_b,...,h2_{b-1}, M)$
3: $r \leftarrow h \bmod$
4: $R \leftarrow r \cdot B$
5: $h \leftarrow H(R, A, M)$
6: $S \leftarrow (r + ah) \bmod \ell$
7: return $(R, S)$
```

The Client App

The client application developed with php 7.2 is responsible for sending the necessary requests in order to complete this process. The processes are as follows;

- Requests Public Key from the proxy server; this process is made possible via an HTTP client known as Guzzle. Guzzle is a PHP HTTP client that makes it easy to send HTTP requests and trivial to integrate with web services. (Guzzle, 2020) Simple interfaces for building query strings and can send synchronous and asynchronous requests using the same interface.

```
$client = new GuzzleHttp\Client();
$res = $client->request('GET', 'https://api.github.com/user', [ 'auth' =>
['user', 'pass'] ]); echo $res->getStatusCode();
```

- Joins public key with every SQL keyword used in the query; since we have listed every SQL keyword into any array, first we can split the query to get only the SQL keywords using *explode(separator, string, limit),* hereafter we can check the query for SQL keywords against the array using the *in_array(array1, array2)* function and merge the public key from the first step to every keyword.
  For example; if the public key is 4wdflasdxsdfsdfSEA
  SELECT * FROM shopping WHERE username='$username' AND password='$password';
  The result thereafter is
  SELECT4wdflasdxsdfsdfSEA * FROM4wdflasdxsdfsdfSEA shopping WHERE4wdflasdxsdfsdfSEA username='$username' AND4wdflasdxsdfsdfSEA password='$password';
- Split the entire SQL statement including user inputs into tokens; this is to ensure that the public key is attached to every SQL keyword this time. The tokens are stored in a different array.
- Checks each token that all SQL keywords have the public key
- Queries the Proxy server for the private key of the public key used; this is an extra layer of security incase an attacker has enough computing power to get generate a key that is similar to the public key.
- If private key exists, send full SQL statement to DB server

The Proxy Server

The implementation of this server done with ASP.NET Core 2.1 could have been done easily on the application server but this separation provides a layer of security. This ensures that this part is not exposed together with the application, even if the application is attacked. The proxy server is located between the client application and the database server. It serves as to;

- Generates keypair and Send Public Key to the Client App; With the keypair class in Stellar, we are able to generate two keys, 256 bits long.
- Stores private key of the public key sent
- Retrieves private key for the client app to verify the public key

The test tool used for this purpose is SQLMAP. SQLMAP is an open-source penetration testing tool written in python that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, a number of niche features for the ultimate penetration tester and a wide range of switches from database fingerprinting, database collection of data to access to the underlying file system and commands execution on the operating system via out-of-band connections. It is capable of detecting six (6) SQL injection attack techniques; boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band. SQLMAP works by passing a potentially vulnerable parameter into the query link.

Performance Evaluation

To evaluate the performance of this model, metrices such as the response time, processing time and overhead imposed by the additional server in the middle tier are used. To achieve this, a separate study was created to simulate a number of users using round-robin to login at the same time. The response time of the proxy server and the database server was evaluated for 1 user, 10 users, 50 and 100.

Users	Proxy Server (Time in seconds)	Database Server (Time in seconds)	Overall time in seconds
1	0.0010	0.00034	0.00134
10	0.012	0.005	0.017
50	0.049	0.023	0.072
100	0.13	0.09	0.22

Table 1.Response Time Evaluation

In comparison with other techniques like AMNESIA, SQLRAND and AUTORAND, there is a slight improvement as shown in the table below

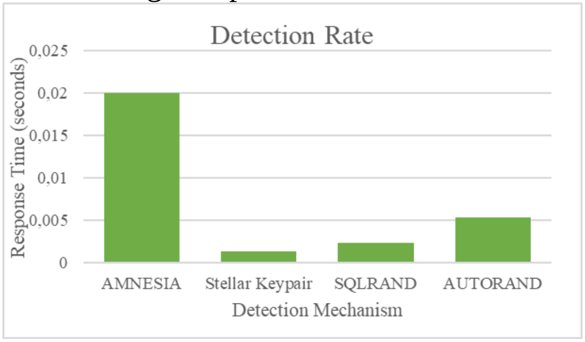

Figure 4. Detection rate of preventive schemes

Although the response times for the proxy server are subjective because they really depend on the speed of the internet connection at the time of implementation. Its overhead ranges from 120 to 490 microseconds for 10 to 100 concurrent users respectively. We also measured the response time needed to detect the SQL injection attack for the type of attack.

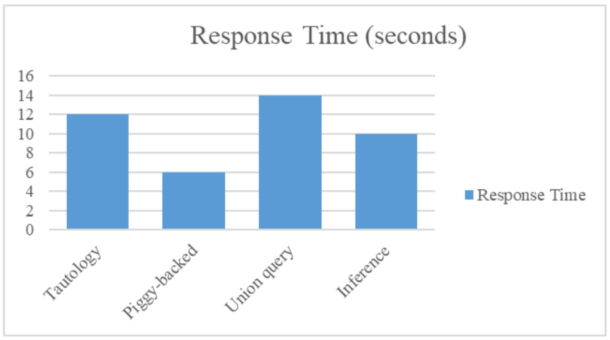

Figure 5.Response time of preventive schemes

The table below shows the ways in which different prevention techniques are used against the attacks and their effectiveness are compared with each other. These comparisons are important for us to make a better choice. The table shows the results of the comparison.

Schemes	Tautology	Logically Incorrect Queries	Union Query	Stored Procedure	Piggy-backed queries	Inference Attack
AMNESIA	YES	YES	YES	NO	YES	YES
SQLRand	YES	NO	YES	NO	YES	YES
AutoRand	YES	NO	YES	NO	YES	YES
CANDID	YES	NO	NO	NO	NO	NO
SQLGuard	YES	YES	NO	NO	NO	NO
SQLIPA	YES	YES	YES	NO	YES	YES
Negative Tainting	YES	YES	YES	NO	YES	YES
Positive Tainting	YES	YES	YES	YES	YES	YES
Stellar Keypair	YES	YES	YES	YES	YES	YES

Table 2.Efficiency of various techniques

Conclusion

This study presents the results of a methodological and systematic review of different literatures on the different types of SQL injection attacks, effects of SQLIA techniques and preventive techniques. We were able to successfully create a new method for SQLIA prevention using the stellar keypair by expanding existing well-known models. These models (SQLRand and AutoRand) could not prevent second-order injection attacks, stored procedures and logically incorrect queries. This method contains details of the implementation using the Ed25519 keys (public and private keys) which is the improvement model mentioned above. Its architecture has also been explained extensively and explicitly using a proxy server. Finally, an evaluation was carried out to check the effectiveness of this model and to compare it with other similar models in implementation. However, there is still room for extensibility i.e. if there is enough computing power to break down the Ed25519 keys, a stronger keypair algorithm is needed. For better implementation, a stronger internet connection should be implored to make connection between the client application and the proxy server. After imploring this method, sensitive data on the database could still be encrypted to provide another layer of security.

References:

1. Alazab, A., & Khresiat, A. (2016). New Strategy for Mitigating of SQL Injection Attack. International Journal of Computer Applications, 154(11), 1-10. doi:10.5120/ijca2016911974
2. Appiah, B., Opoku-Mensah, E., & Qin, Z. (2017, 24-26 Nov. 2017). SQL injection attack detection using fingerprints and pattern matching technique. Paper presented at the 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS).
3. Bandhakavi, S., Bisht, P., Madhusudan, P., & Venkatakrishnan, V. N. (2007). CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations. Paper presented at the Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS, Alexandria, Virginia, USA,.
4. Boyd, S. W., & Keromytis, A. D. (2004, 2004). SQLrand: Preventing SQL injection attacks, Columbia.
5. Chinchu, M. M., Yeldose, A., & Kumar, D. S. (2015). An Analysis of SQL Injection Prevention using the Algorithms RSA , RC4 and RC5. International Journal of Current Engineering and Technology, 5(6), 3665-3670.
6. Dinu, P. S., Kumar, D. S., & Rahman, M. A. (2015). Preventing SQL injection Attacks Using Cryptography Methods. International Journal of Scientific Research Enginnering &Technology, 4(5), 582-585.
7. Don Johnson, A. M., Scott Vanstone (2001). The Elliptic Curve Digital Signature Algorithm (ECDSA). Paper presented at the Certicom Research,Canada,Canada.http://www.cs.miami.edu/home/burt/learning/Csc609.142/ecdsacert.pdf
8. Elshazly, K., Fouad, Y., Saleh, M., & Sewisy, A. (2014). A Survey of SQL Injection Attack Detection and Prevention. Journal of Computer and Communications, 02(08), 1-9. doi:10.4236/jcc.2014.28001
9. Etienne Janot, P. Z. (2014). Preventing SQL Injections in Online Applications: Study, Recommendations and Java Solution Prototype

Based on the SQL DOM. Paper presented at the Application Security Conference, Belgium.
10. Faker, S. A., Muslim, M. A., & Dachlan, H. S. (2017). A Systematic Literature Review on SQL Injection Attacks Techniques and Common Exploited Vulnerabilities. International Journal of Computer Engineering and Information Technology, 9(12), 284-291.
11. Garg, R., Gupta, P., & Sachdeva, R. K. (2017). Study on SQL Injection Attacks: Detection and Prevention. International Journal for Research in Applied Science & Engineering Technology, 5(VII), 198-203.
12. Gupta, A., Dhankhar, A., & Solanki, K. (2018). New Technique for preventing SQL Injection Attack Based on Normal Use Model. IOSR Journal of Computer, 20(5), 73-83.
13. Guzzle. (2020). Guzzle Releases. In Guzzle (Ed.), (pp. 1-3).
14. Halfond, W. G. J., & Orso, A. (2005). AMNESIA: Analysis and monitoring for NEutralizing SQL-injection attacks. In (pp. 174-183). Long Beach, California, USA.
15. Katole, R. A., Sherekar, S. S., & Thakare, V. M. (2018, 19-20 Jan. 2018). Detection of SQL injection attacks by removing the parameter values of SQL query. Paper presented at the 2018 2nd International Conference on Inventive Systems and Control (ICISC).
16. Khaleel Ahmad, J. S., K.P. Yadav. (2010). Classification of SQL Injection Attacks. VSRD Technical & Non-Technical Journal, 1(4), 235-242.
17. Lawal, M. A., Sultan, A. B. M., & Shakiru, A. O. (2016). Systematic literature review on SQL injection attack. International Journal of Soft Computing, 11(1), 26-35.
18. Lee, I., Jeong, S., Yeo, S., & Moon, J. (2012). A novel method for SQL injection attack detection based on removing SQL query attribute values. Mathematical and Computer Modelling, 55(1-2), 58-68. doi:10.1016/j.mcm.2011.01.050
19. Liangyan, Y. (2018). Summary of Key Technologies of SQL Injection Vulnerability Detection and Defense. Journal of Anhui Vocational College of Electronic Information, 17(03), 19-22.
20. Lianqun, M. K. W. B. H. Y. Y. (2017). New SQL injection attack detection method based on hidden Markov model. Information Network Security, 09(1), 115-118.
21. Lihong, K. L. Y. H. L. (2017). Web security SQL injection vulnerability and its defense. Network Security Technology and Applications, 11(1), 81-82.
22. Liu, M., & Wang, B. (2018). A Web Second-Order Vulnerabilities Detection Method. IEEE Access, 6, 70983-70988. doi:10.1109/ACCESS.2018.2881070

23. Loughran, D. T., Salih, M. K., & Subburaj, V. H. (2018). All About SQL Injection Attacks. Journal of The Colloquium for Information System Security Education, 6(1), 24-24.
24. M., K. R., & Amsaveni, C. (2016). SQL Injection Attack Prevention Using 448 Blowfish Encryption Standard. International Journal of Computer Science Trends and Technology`, 4(4), 325-335.
25. Md. Fazlul Haque, M. B. A. M., Fuyad Al Masud. (2017). Enhancement of Web Security Against External Attack. European Scientific Journal, 13(15). doi:10.19044/esj.2017.v13n15p228
26. Panda, S., & Ramani, S. (2013). Protection of Web Application against Sql Injection Attacks. International Journal of Modern Engineering Research, 3(1), 166-168.
27. Patel, N., Mohammed, F., & Soni, S. (2011). SQL Injection Attacks: Techniques and Protection Mechanisms. International Journal on Computer Science & Engineering, 3(1), 199-203.
28. Patil, A., Laturkar, A., Athawale, S. V., Takale, R., & Tathawade, P. (2017, 17-19 Aug. 2017). A multilevel system to mitigate DDOS, brute force and SQL injection attack for cloud security. Paper presented at the 2017 International Conference on Information, Communication, Instrumentation and Control (ICICIC).
29. Perkins, J., Eikenberry, J., Coglio, A., Willenson, D., Sidiroglou-Douskos, S., & Rinard, M. (2005, 2016). AutoRand: Automatic keyword randomization to prevent injection attacks, Columbia.
30. Romailler, Y., & Pelissier, S. (2017, 25-25 Sept. 2017). Practical Fault Attack against the Ed25519 and EdDSA Signature Schemes. Paper presented at the 2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC).
31. Saravana, P. (2014). Efficient Method for Preventing SQL Injection Attacks on Web Applications Using Encryption and Tokenization. Internationational Journal of Latest Trends in Engineering and Technology, 4(4), 75-84.
32. Shrivastava, S., & Tripathi, R. R. K. (2012). Attacks Due to SQL injection & their Prevention Method for Web-Application. International Journal of Computer Sciecne …, 3(2), 3615-3618. Retrieved from http://ijcsit.com/docs/Volume 3/Vol3Issue2/ijcsit2012030266.pdf
33. Singh, J. P. (2017). Analysis of SQL Injection Detection Techniques (Vol. 28). Montreal, Quebec, Canada: CIISE, Concordia University.
34. Sun, S.-T., Wei, T. H., Liu, S., & Lau, S. (2007). Classification of sql injection attacks. University of British Columbia, …, 1(4), 1-6. Retrieved from

https://courses.ece.ubc.ca/412/term_project/reports/2007-fall/Classification_of_SQL_Injection_Attacks.pdf

35. Wang Degao, X. W. C., Wang Liming, Liu Xiangdong. (2019). Design and Implementation of SQL Injection Attack and Prevention Experiment. Journal of Dalian Nationalities University, 21(05), 441-444.
36. Xin, Z. Y. (2017). Anti-SQL injection strategy based on HttpModule. Network Security Technology and Applications, 11(1), 60-61 64.
37. Xuan, X. (2019). Research on SQL Anti-injection Attack. China New Communications, 21(05), 64-64.
38. Yiğit, G., & Arnavutoğlu, M. (2017). SQL Injection Attacks Detection & Prevention Techniques. International Journal of Computer Theory and Engineering, 9(5), 351-356.
39. Yonghua, C. (2019). Research on SQL Injection Recognition Algorithm Based on Random Forest. Modern Information Technology, 3(15), 146-149.
40. Yonghui, X. J. L. Q. Y. (2019). SQL injection attack detection based on deep convolutional neural network. Journal of Jimei University (Natural Science Edition, 24(03), 234-240.