IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

Enhancing SQL Injection Security Through Cryptographic Measures

[1]D. Bhagya Sree, [2] G. Kotireddy, [3]B. Pavan Kalyan Dinesh, [4] B. Sashi Kiran, [5] S. Nikhila
[1] Cyber Security [2]Cyber Security, [3]Cyber Security,[4] Cyber Security, [5] Cyber Security
Raghu Engineering College, India

Abstract: This paper discusses the critical matter of SQL Injection Attacks (SQLIA), which are widely recognized as a prevalent vulnerability in web applications and are among the top 10 web security concerns identified by the Open Web Application Security Project. As social networking and e-commerce have proliferated, so have assaults such as spamming and phishing, which pose grave risks to the security of user data. SQLIA permits unauthorized database access, modification, and deletion; therefore, hackers must exploit details such as table and field names. In order to address this issue, we suggest implementing a divide-and-conquer strategy that utilizes randomization-based encryption algorithms and the Hirschberg algorithm to bolster security while decreasing the complexity of time and space. By substantially enhancing security and resilience against prevalent attack techniques such as dictionary and brute force assaults, our solution outperforms current methodologies. Through the implementation of cutting-edge encryption methods and algorithmic improvements, our objective is to construct a resilient collection of tools that can effectively protect databases against unauthorized entry. This will consequently guarantee an elevated level of security for both users and their data.

***Key Words* - SQL, SQLIA, Hirschberg, encryption.**

I. Introduction

The significant incidence of SQL Injection Attacks (SQLIA) in web applications emphasizes the urgent requirement for increased vigilance and strong safeguards against these susceptibilities. SQLIA comprises an estimated 14-15% of web application assaults, as stated in the White Hat report on web security vulnerabilities for 2011. *(Gupta, H., Mondal)* Consequently, it is imperative to adopt proactive measures in order to mitigate the associated risks. In a time when online transactions and interactions predominate, it is critical to comprehend and effectively mitigate diverse attack vectors, such as denial of service attacks, social engineering, and phishing.*(Selvamani, K)* Notwithstanding progress made in the implementation of encryption methods and defensive coding practices, SQL Injection Prevention (SQLIA) continues to pose a substantial risk as a result of inadequate input validation and the continuous evolution of sophisticated attack strategies. It is important to highlight that conventional defensive programming methods struggle to fully mitigate SQLIA threats. *(Namdev, M)* These methods frequently concentrate on particular attack subsets and present difficulties when it comes to retrofitting legacy software.In order to effectively address SQLIA, innovative approaches are suggested that utilize randomized techniques and encryption algorithms. Our contribution is the creation of a Secure Hashing Algorithm-based encryption strategy that is specifically designed for web applications. This approach provides improved resilience against SQLIA attacks. In order to validate the effectiveness of our proposed solution, we also present a Java-based utility for ciphertext generation and perform empirical analyses. In addition, the growing dependence on web-based services underscores the need for a comprehensive strategy towards safeguarding web applications. This strategy should go beyond conventional security measures like intrusion detection systems and firewalls. The critical nature of comprehensively addressing SQLIA is underscored by the widespread use of vulnerable database technologies and dynamic scripting languages (e.g.,php, ASP.net). Despite the fact that numerous mitigation strategies have been investigated in prior research, a number of them have practical and efficacylimiting deficiencies. With the intention of protecting sensitive data and reducing potential risks presented by malicious entities, our strategy is to fortify web applications against SQL Injection assaults through the implementation of a combinational approach that incorporates randomized techniques, defensive coding practices, and encryption.

II. Literature Survey

SQL Injection Attacks (SQLIAs) present a substantial peril to the security of databases as they facilitate illicit data manipulation and access via susceptible web applications. Intentions underlying SQLIA encompass unauthorized entry into databases, extraction and modification of data, escalation of user privileges, and malfunctioning applications. These malicious attacks capitalize on the susceptibility of web application parameters, inadvertently infiltrating the database in the absence of the administrator. An innovative strategy for identifying and thwarting SQLIA attacks entails the creation of a randomization-based encryption algorithm tailored to each application. *(Som, S., Sinha)* Although SQLIA can be utilized to exploit databases in a variety of ways, current solutions generally only mitigate a subset of these attack methods. Prior to the development of a tool, it is essential to conduct an exhaustive literature review, taking into account organizational capabilities, economic viability, and schedule constraints. It is consequently critical to ascertain the appropriate programming language and operating system. Throughout the development process, programmers necessitate external assistance from seasoned colleagues, reference materials, and online resources. *(Ezumalai, R.)* The SQLIA can have a significant influence on database security, potentially leading to unauthorized modification of records and tables, deletion of databases, or both, all without the knowledge of the user or administrator. The compromise of confidential data by these attacks frequently goes undetected for an extended period of time following the intrusion. SQL Injection attacks are readily executable through the use of basic web browsers. To detect SQLIA nodes, it is critical to comprehend the architecture of web applications. The 3-tier logical view architecture consists of the following components: the user interface tier (frontend), which is where user interactions take place; the business logic tier (intermediate layer), which processes user requests and executes server-side programming; and the database tier (backend), which utilizes the database server to facilitate data storage and retrieval. In order to bolster the overall security of web applications, mitigation strategies must sufficiently target vulnerabilities that span these architectural layers.

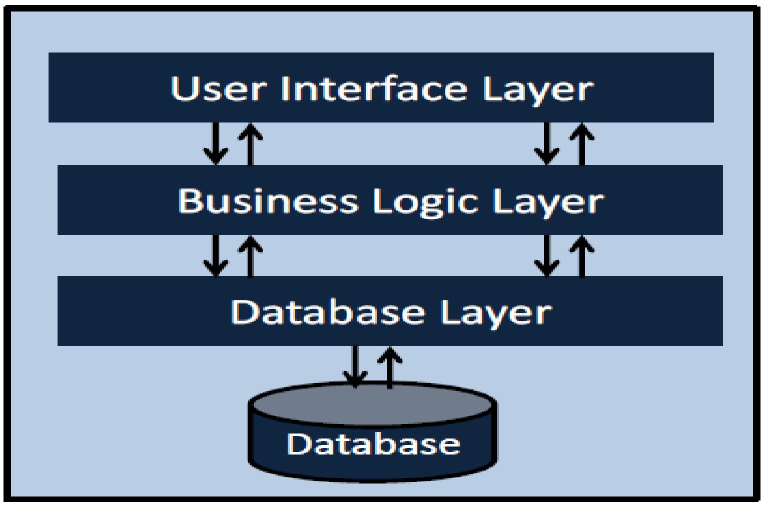


Fig.1 Web 3-tier architecture

III. Proposed System Architecture

This methodology utilizes runtime monitoring to identify and avert SQL Injection Attacks (SQLIA) through dynamic analysis of the login page redirection of every application to a specialized verification page. The underlying principle of its operation is to detect and prevent SQLIA attempts, while permitting authorized accesses to continue without interruption. SQL Injection Attack (SQLIA) refers to the malevolent injection of SQL statements into concealed parameters or input fields of web applications. This allows for unauthorized access or manipulation of data. Given the increasing frequency of SQLIA incidents, which present substantial security vulnerabilities, the suggested remedy combines the Secure Hashing Algorithm and Hirschberg Algorithm to ensure a strong defense. The technique strengthens application security by employing cryptographic mechanisms and utilizing Secure Hashing Algorithm, thereby increasing resilience against SQLIA attempts. Furthermore, the integration of the Hirschberg Algorithm facilitates effective runtime monitoring through the utilization of a divide-and-conquer strategy to examine incoming requests in search of possible SQL injection patterns. By taking this proactive approach, SQLIA can be detected and prevented in real-time, without causing any disruption to legitimate user interactions. The effectiveness of the technique resides in its capacity to dynamically evaluate and address incoming queries, thus reducing the growing risk presented by SQL injection attacks. By leveraging the combined capabilities of algorithmic analysis and cryptographic hashing, the suggested solution offers an all-encompassing defense mechanism against SQLIA. This ensures the protection of sensitive data and the ongoing integrity of web applications.

By employing client-side script validation, specifically with JavaScript, a multitude of SQL injection attacks against web applications can be thwarted. Although it fails to encompass every possible attack vector, it implements essential security protocols to prevent unauthorized ingress. The procedure entails implementing measures such as restricting the quantity of inputs and special characters, although it may not be practical to apply these constraints universally to all applications. Additionally, client-side security measures are readily circumvented. While this methodology successfully mitigates assaults incorporating tautology or erroneous queries, it continues to be susceptible to blind injection techniques. As a result, server-side validation techniques are favored owing to their extensive scope and ability to withstand sophisticated attack methodologies.

A. Software System Architecture

During the development phase, the first stage is design, which involves defining the techniques and principles necessary to precisely construct a physical implementation of a device, process, or system. The design phase, which ensues after the analysis and specification of software requirements, encompasses three critical technical tasks: design, coding, implementation, and testing. These activities are essential for the construction and validation of the software. The design process encompasses crucial decisions that have a significant impact on the success of software implementation and the simplicity of its maintenance, which in turn affects the dependability and maintainability of the system. By bridging the gap between customer specifications and the final software or system implementation, design promotes quality throughout the development process. The process enables the conversion of specifications into a concrete software model, which is carried out in two stages: preliminary design, which concentrates on the conversion of specifications into data. Component diagrams in the Unified Modeling Language (UML) depict the structural composition of larger software systems by illustrating how components interconnect. In the context of UML, deployment diagrams depict the tangible placement of artifacts on nodes. They distinguish between software and hardware components (artifacts), which are represented by nodes, and the ways in which they are interconnected (e.g., REST, RMI) for systems such as websites. By doing so, they illustrate how components are distributed and interact throughout the physical infrastructure.

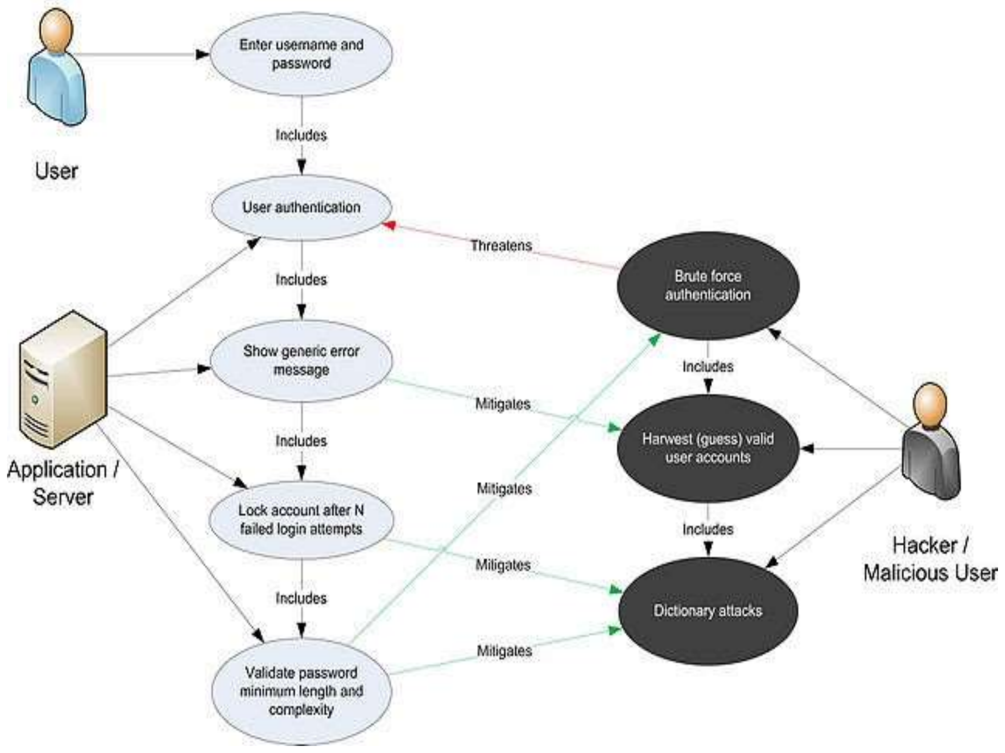

User
Enter username and password
Includes
User authentication
Threatens
Includes
Application / Server
Show generic error message
Mitigates
Includes
Lock account after N failed login attempts
Mitigates
Mitigates
Includes
Validate password minimum length and complexity
Mitigates
Brute force authentication
Includes
Harvest (guess) valid user accounts
Includes
Dictionary attacks
Hacker / Malicious User

Fig.2 Component Diagram of SQL attack.

IV. Research Methodology

SQL injection attacks are a common web security risk that compromises the confidentiality and integrity of sensitive user data by exploiting input elements in web forms that have been inadequately designed. It transpires within the application between the business logic and user interface layers. *(Karunanithi, J. S)* An illustrative instance exemplifies the fundamental concept of SQL injection: an intruder can obtain illicit access by inputting SQL statements such as 'OR '1'='1" in place of valid credentials. This is due to the fact that '1=1' is invariant and '--' designates the remaining statement as a remark, thereby circumventing authentication.

There are numerous varieties of SQLi attacks:
1) Tautology: the manipulation of input in order to validate logical expressions.
2) Illicit/Incorrect queries: extracting information from a database by utilizing error messages.
3) Piggy-Backed Queries: utilizing special characters such as ';' to append malevolent queries.
4) Blind Injection: To circumvent conventional attacks by concealing database information, hackers must infer data using true/false queries and frequently employ timing attacks for analysis.

Every category presents unique vulnerabilities, including unauthorized entry, data manipulation, and deletion. Effectively preventing SQL injection attacks requires solutions to mitigate the vulnerabilities introduced by un validated inputs.

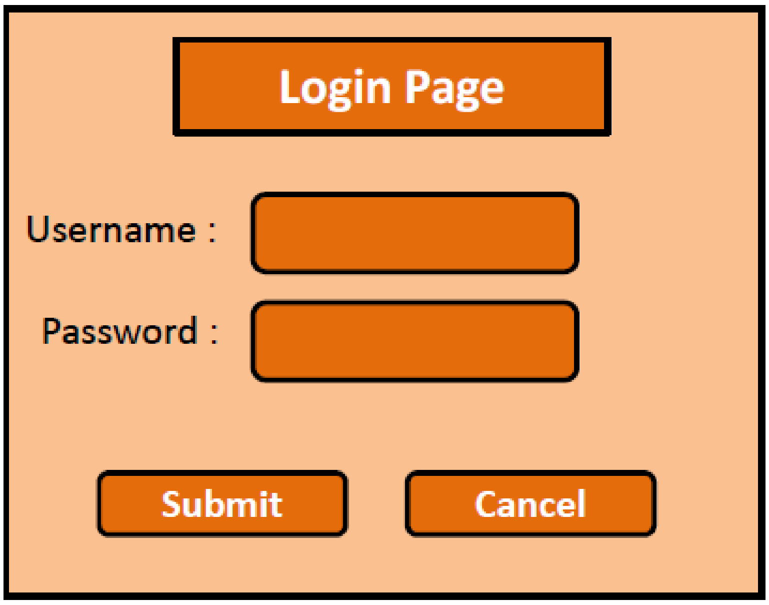
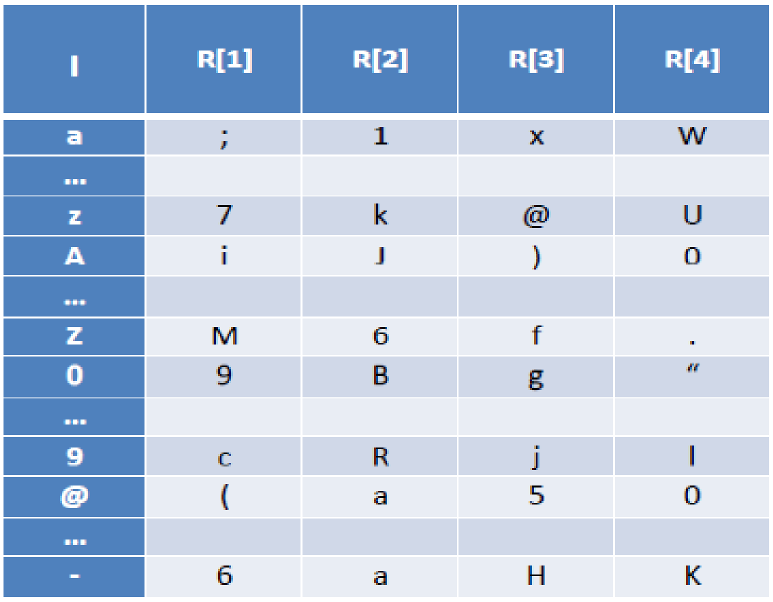


Fig.3 Sample Login form

The cryptographic method under consideration utilizes randomization to convert input data into ciphertext, supplemented with cryptographic salt for heightened security. Web form input fields customarily consist of numeric values, capital and small letters, and a maximum of ten special characters. For a 6-character input, there are a total of 72^6 combinations per character, comprised of 26 lowercase letters, 26 uppercase letters, 0-9 numerals, and 10 special characters. In order to ensure the confidentiality of the input, four random values are assigned to each character from a predefined lookup table. The nature of the character that follows each character in the input sequence determines the random value assigned to that character. For instance, the algorithm selects R[1] from the retrieval table if the following character is a lowercase letter and the current character is 'a'. An alternative approach would be to select R[2] if the following character were capitalized, R[3] if it were a digit, and R[4] if it were a special character or no character. Ensuring security is improved by tailoring these random values to individual applications, as this decreases the probability of decryption by malicious entities. By tailoring the cryptographic scheme to specific applications, its robustness against potential assaults is enhanced, thereby guaranteeing comprehensive protection for sensitive data that is transmitted via web forms.



I	R[1]	R[2]	R[3]	R[4]
a	;	1	x	W
...				
z	7	k	@	U
A	i	J	)	O
...				
Z	M	6	f	.
0	9	B	g	"
...				
9	c	R	j	I
@	(	a	5	0
...				
-	6	a	H	K

Fig.4 Lookup table for Cryptography Approach

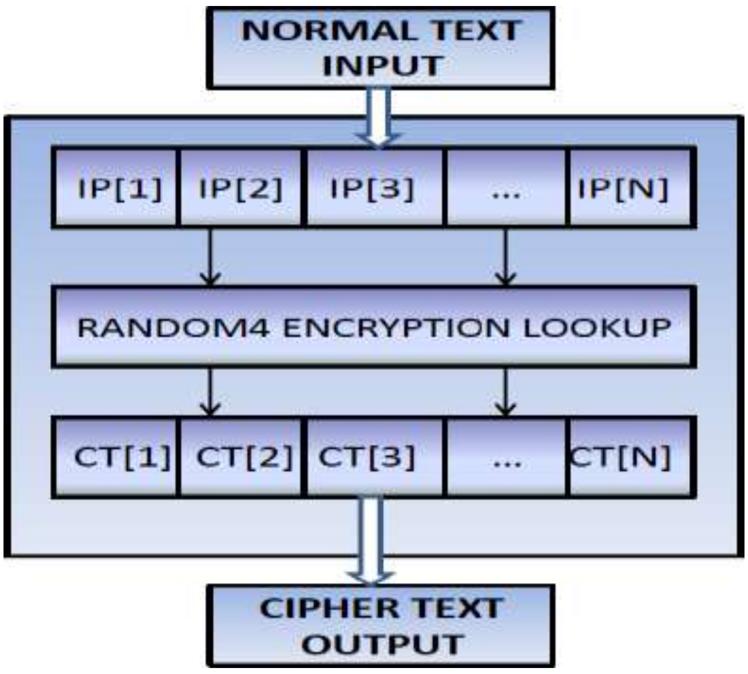


Fig.5 Framework for Cryptography Approach

V. RESULTS AND DISCUSSION

During the unit testing phase, a comprehensive manual review of field entries and activation of pages via identified links was performed; functional tests were scripted with great attention to detail. The primary goals encompassed guaranteeing the accurate operation of input fields, facilitating the smooth activation of webpages via specified hyperlinks, and ensuring prompt responses devoid of any delays. The testing primarily aimed to validate the accuracy of entry formats, ensure that duplicate entries were prohibited, and verify that users were redirected to the appropriate pages upon activating links. The objective of system testing was to detect and redress any software errors. The process involved testing the software in order to detect any potential flaws or vulnerabilities, guaranteeing compliance with specifications and user anticipations, and averting undesirable malfunctions. The exhaustive testing procedure encompassed a multitude of test varieties, each designed to fulfill distinct testing criteria. The primary objective of integration testing was to identify interface defects through the incremental integration of software components. The aim was to ensure that the interactions between integrated components were error-free and seamless. The test outcomes demonstrated that every test case was executed successfully, with no defects encountered; this validates the integrity and dependability of the integrated software components. During the course of these testing phases, a comprehensive evaluation was undertaken to substantiate the software system's functionality, performance, and dependability. The lack of detected defects throughout the integration testing process serves as evidence of the testing procedures' efficacy and the superior quality of the integrated software components.

II. CONCLUSION

The significant prevalence of SQL Injection Attacks (SQLIA) highlights the urgent requirement for enhanced security protocols in order to protect user information within web applications. The development and evaluation of an application-specific randomized encryption algorithm designed to detect and prevent SQLIA have been the subject of this paper. By conducting a thorough examination, we have evaluated its effectiveness in comparison to established methodologies, quantifying its performance metrics to illustrate its efficacy. A thorough examination of the diverse attack vectors linked to SQLIA reveals the substantial security risk it presents, underscoring the need for resilient defensive approaches. User data stored in web applications is of the utmost sensitivity and confidentiality; therefore, it is critical to strengthen security measures in order to prevent potential breaches.By utilizing randomized encryption algorithms that are customized for particular applications, our proposed solution provides a proactive method for preventing SQLIA attempts. By means of empirical comparisons with alternative methodologies, we have successfully illustrated the superior performance and effectiveness of our strategy in reducing SQLIA vulnerabilities.

In conclusion, ensuring the security of web applications continues to be contingent upon addressing SQLIA. The significance of employing customized encryption solutions to effectively counter this ubiquitous threat is emphasized by our findings. Ongoing efforts to strengthen defenses against SQLIA and other cyber threats are crucial in maintaining user confidence and data integrity in web environments, given the dynamic nature of the digital environment.

References

[1] Gupta, H., Mondal, S., Ray, S., Giri, B., Majumdar, R., & Mishra, V. P. (2019, December). Impact of SQL injection in database security. In *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)* (pp. 296-299). IEEE.

[2] Selvamani, K., & Kannan, A. (2011, April). A Novel Approach for Prevention of SQL Injection Attacks Using Cryptography and Access Control Policies. In *International Conference on Power Electronics and Instrumentation Engineering* (pp. 26-33). Berlin, Heidelberg: Springer Berlin Heidelberg.

[3] Namdev, M., Hasan, F., & Shrivastav, G. (2012). A Novel Approach for SQL Injection Prevention Using Hashing & Encryption (SQL-ENCP). *Int J Comput Sci Informat Technol (IJCSIT)*, *3*(5), 4981-7.

[4] Madhusudhan, R., & Ahsan, M. (2022, March). Prevention of SQL Injection Attacks Using Cryptography and Pattern Matching. In *International Conference on Advanced Information Networking and Applications* (pp. 624-634). Cham: Springer International Publishing.

[5] Karunanithi, J. S. (2018). SQL injection prevention technique using cryptography.

[6] Avireddy, S., Perumal, V., Gowraj, N., Kannan, R. S., Thinakaran, P., Ganapthi, S., ... & Prabhu, S. (2012, June). Random4: An application specific randomized encryption algorithm to prevent SQL injection. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications* (pp. 1327-1333). IEEE.

[7] Som, S., Sinha, S., & Kataria, R. (2016). Study on sql injection attacks: Mode detection and prevention. *International journal of engineering applied sciences and technology*, *1*(8), 23-29.

[8] Jemal, I., Cheikhrouhou, O., Hamam, H., & Mahfoudhi, A. (2020). Sql injection attack detection and prevention techniques using machine learning. *International Journal of Applied Engineering Research*, *15*(6), 569-580.

[9] Ezumalai, R., & Aghila, G. (2009, March). Combinatorial approach for preventing SQL injection attacks. In *2009 IEEE International Advance Computing Conference* (pp. 1212-1217). IEEE.

[10] Aliero, M. S., Ardo, A. A., Ghani, I., & Atiku, M. (2016). Classification of Sql Injection Detection And Prevention Measure. *IOSR Journal of Engineering*, *6*(02).

[11] Aliero, M. S., Ghani, I., Zainudden, S., Khan, M. M., & Bello, M. (2015). Review on SQL injection protection methods and tools. *Jurnal Teknologi*, *77*(13), 49-66.

[12] Balasundaram, I., & Ramaraj, E. (2011). An authentication mechanism to prevent SQL injection attacks. *International Journal of Computer Applications*, *19*(1), 30-33.

[13] Kar, D., & Panigrahi, S. (2013, February). Prevention of SQL Injection attack using query transformation and hashing. In *2013 3rd IEEE International Advance Computing Conference (IACC)* (pp. 1317-1323). IEEE.

[14] Kar, D., & Panigrahi, S. (2013, February). Prevention of SQL Injection attack using query transformation and hashing. In *2013 3rd IEEE International Advance Computing Conference (IACC)* (pp. 1317-1323). IEEE.

[15] Rajeswari, K. C., & Amsaveni, C. (2016). SQL injection attack prevention using 448 blowfish encryption standard. *International Journal of Computer Science Trends and Technology (IJCST)*, *4*, 325.

[16] Bangre, S., & Jaiswal, A. (2012). SQL Injection Detection and Prevention Using Input Filter Technique. *International Journal of Recent Technology and Engineering (IJRTE)*, *1*(2), 145-150.

[17] Yunus, M. A. M., Brohan, M. Z., Nawi, N. M., Surin, E. S. M., Najib, N. A. M., & Liang, C. W. (2018). Review of SQL injection: problems and prevention. *JOIV: International Journal on Informatics Visualization*, *2*(3-2), 215-219.

[18] Temeiza, Q., Temeiza, M., & Itmazi, J. (2017, August). A novel method for preventing SQL injection using SHA-1 algorithm and syntax-awareness. In *2017 Joint International Conference on Information and Communication Technologies for Education and Training and International Conference on Computing in Arabic (ICCA-TICET)* (pp. 1-4). IEEE.