SQL Injection Countermeasures

Sivakumaran Sivashan, Faculty of Cyber Security, SLIIT

Abstract—**A lot of data is created very quickly every day and eventually this data is stored in databases. All processes and processors use the database management system due to the privacy of the stored data, and the data in this system must be protected and not fail before possible database attacks, so it is necessary to use security models, although they differ between them. Database injection has become a common desirable attack targeting web applications, allowing the attacker to access the database to modify SQL queries without authorization. SQL injection should be read and understood to avoid data loss and harm to the public. In this paper, I will explain SQL injection and its counter measures.**

Index Terms—**SQL, Injunction, Attacks, Database, Countermeasure**

I. INTRODUCTION

Everyone needs databases to store any data due to the speed and reasonable cost of the database. The advantage of using a database is that it automates various processes, saves resources and saves working time. For example, instead of manually verifying transactions, users can rely on computer reports stored in the database. The question for everyone is "Is the data in the database secure?"

In today's world security is one of the most important and difficult tasks that people around the world face in every aspect of their lives. SQL injection is one of the main methods used by attackers to hack a database, this type of attack exploits the vulnerabilities in the website or database server. The attacker pays risk query sections created to alter the intended effect so that the attacker can gain unauthorized access to a database, read or modify data, make the data inaccessible to other users, or damage the database server.

In recent years, SQL injection attacks have increased rapidly, becoming the first type to target web applications. In the first half of 2020, the average daily SQL injection attack worldwide reached 400,000. [8]

Careful configuration and programming and effective anti-attack security mechanisms are required to ensure the security of web applications and their databases. To solve SQL injection problems, many solutions have been suggested by researchers, and despite all these, there is no definitive solution to ensure complete safety.

II. SQL INJECTION IMPACTS

SQL is a database query language that enables web programming languages to communicate with database servers using dedicated libraries. For each language, communication is done by queries. The server provides results based on the query. Web applications may be the target of an attack. Queries injection when accepting input from the user or computer.

An injection attack occurs when data comes from an unreliable source, or generates dynamic queries based on incoming data, violating SQL injection authentication, data loss, and access denial, and damaging the entire database or host acquisition. The main effect of these vulnerabilities is:

Confidentiality: Loss of confidentiality of sensitive data stored at the level of search databases viewed by non-owners and unauthorized use of such data in illegal activities. [9]

Integrity: Successful injection provides the opportunity to transfer or delete data from infected databases. [10]

Authentication: Queries do not verify the username and password, which creates an unauthorized user to query the database as an authorized user to access the data without even knowing the password or username. [11]

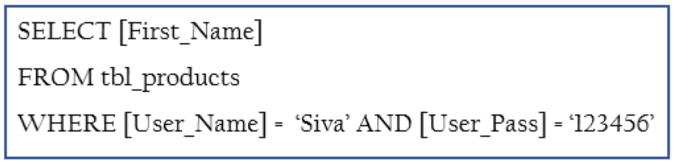
Authorization: Allows the attacker to modify or enhance the privileges of the affected database, thus giving the controller greater control over the system. [12]

Functionality: Simultaneous data processing on databases is an important feature that helps users access and share data simultaneously, and the attack destroys these functions. [13]

III. SQL INJECTION TYPES

Tautologies: Used to subject code to a condition so that it is always accurately evaluated, and this process is performed in the absence of input verification: [14]

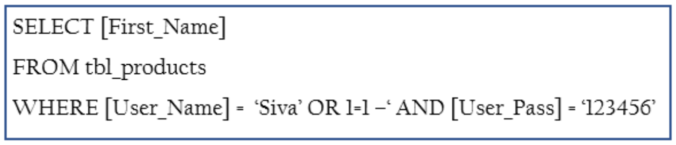
Example without Injection:



```
SELECT [First_Name]
FROM tbl_products
WHERE [User_Name] = 'Siva' AND [User_Pass] = '123456'
```

Fig. 1. SQL query without injection

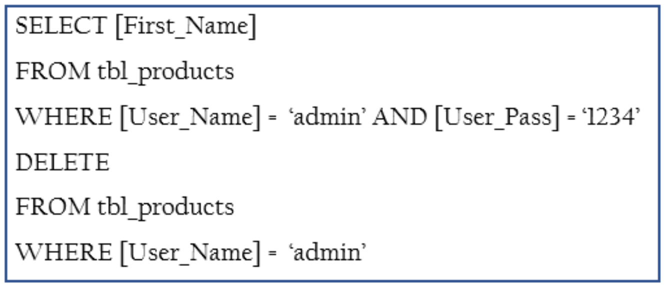
Example with Injection:



```
SELECT [First_Name]
FROM tbl_products
WHERE [User_Name] = 'Siva' OR 1=1 –' AND [User_Pass] = '123456'
```

Fig. 2. SQL query with injection

Piggyback Queries: The attacker sends multiple queries embedded in one query, where the second query contains malicious queries, so it runs as part of the main query. [14]



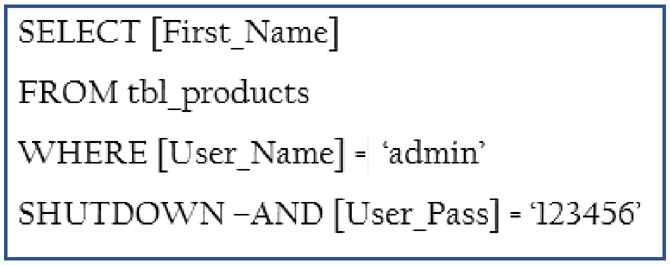
```
SELECT [First_Name]
FROM tbl_products
WHERE [User_Name] = 'admin' AND [User_Pass] = '1234'
DELETE
FROM tbl_products
WHERE [User_Name] = 'admin'
```

Fig. 3. SQL query with injection

Alternative Encryption: This technique, used to escape the detection of unwanted characters, combines the "CHAR" function with the number code and returns the original character. For example, the attacker used charcoal (44) instead of the bad character (').

Illegal / Logical: This method relies on entering data so that the query is incorrect, so the system shows these errors when you receive errors while executing the query, and what does the error show to the attacker? Depending on him and the techniques and back-end will collect important information about the type and structure of the database and the methods he uses, and the components that will work together and help the attack process. [14]

Stored Procedure: The attacker uses procedures stored in a database system. These saved procedures are a set of codes that perform certain tasks without being written down each time, and contain some dangerous tasks that the attacker can exploit by shutting down the system. [14]



```
SELECT [First_Name]
FROM tbl_products
WHERE [User_Name] = 'admin'
SHUTDOWN –AND [User_Pass] = '123456'
```

Fig. 4. Execute shutdown procedures select info

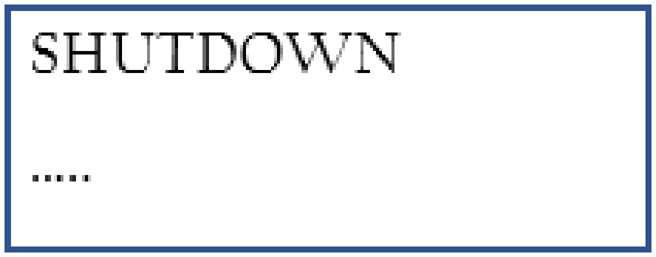
Union query: A secondary query linked to the main query using the word union to get information about other tables from the database. The attacker can get information about the data type or columns and their properties. [14]

IV. SQL INJECTION COUNTERMEASURES

Disable unused features: Disable all features you do not use. You do not need to keep the functions of your server unused because they are dangerous to you.

Custom error message: When running the query, check if it returns an error. If there is an error, place the custom error. SQL errors give more information to the malicious user.

Escape Functions: Setting up escape functions is very easy and allows you to quickly protect your server against most attacks.

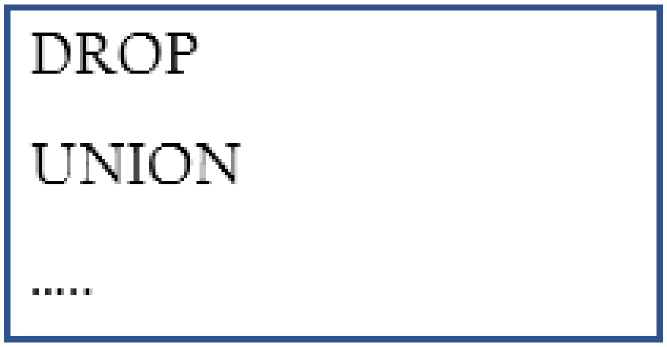


```
SHUTDOWN

.....
```

Fig. 5. Type of danger function

Block some keywords: Do not allow certain characters or keywords. There are actually words that the user does not have to enter the query. If this kind of word is sent back, it should be removed because it may have come from an attack.



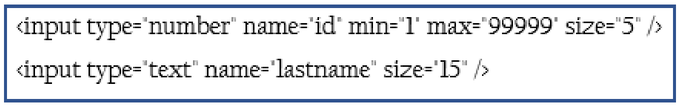
```
DROP

UNION

.....
```

Fig. 6. Type of danger keywords

Control the amount of data: You can control the amount of data included by the user. We found that some injections require a certain number of characters (you can limit ID numbers to five characters, for example the login would be 15).

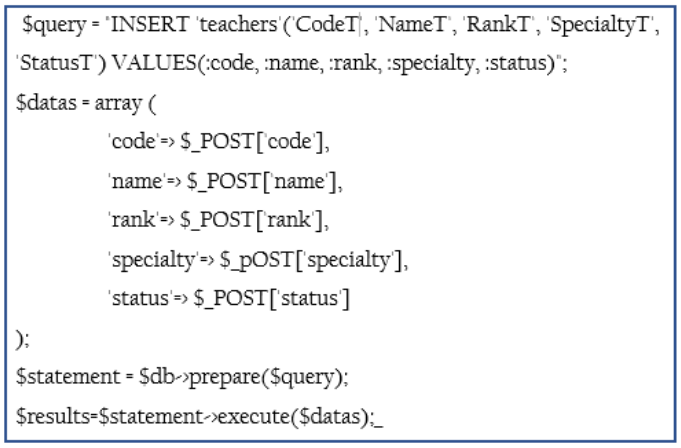


```
<input type="number" name="id" min="1" max="99999" size="5" />
<input type="text" name="lastname" size="15" />
```

Fig. 7. HTML input with limitation of size

Use Product Statements: Use Product Statements. This solution is very effective in protecting against SQL injections. This will ask you for a while (not much) but will effectively protect your server.

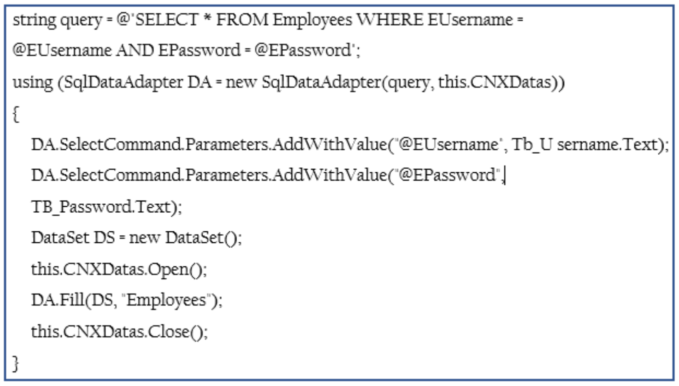
PHP



```
 $query = 'INSERT 'teachers'('CodeT', 'NameT', 'RankT', 'SpecialtyT',
'StatusT') VALUES(:code, :name, :rank, :specialty, :status)";
$datas = array (
            'code'=> $_POST['code'],
            'name'=> $_POST['name'],
            'rank'=> $_POST['rank'],
            'specialty'=> $_pOST['specialty'],
            'status'=> $_POST['status']
);
$statement = $db->prepare($query);
$results=$statement->execute($datas);_
```

Fig. 8. Example prepare statement using PHP

ASP.NET



```
string query = @'SELECT * FROM Employees WHERE EUsername =
@EUsername AND EPassword = @EPassword';
using (SqlDataAdapter DA = new SqlDataAdapter(query, this.CNXDatas))
{
   DA.SelectCommand.Parameters.AddWithValue("@EUsername", Tb_U sername.Text);
   DA.SelectCommand.Parameters.AddWithValue("@EPassword",
   TB_Password.Text);
   DataSet DS = new DataSet();
   this.CNXDatas.Open();
   DA.Fill(DS, "Employees");
   this.CNXDatas.Close();
}
```

Fig. 9. Example prepare statement using ASP.NET

V. SQLI ATTACK EVALUATION

Researchers have suggested several techniques to solve the SQL injection problem. These technologies range from development best practices to fully automated structures for detecting and preventing SQLIAs. Here we will review these proposed techniques and summarize their advantages and disadvantages.

It is very clear that SQLI attacks are an important class of web application attack. Figure 12 highlights the most recent when the full list is too large. Any website is vulnerable to SQLI attacks if not properly protected. Although a great deal of research has been done on the different types of SQLI attacks and methods, there are shortcomings in the research that deal with effective techniques to prevent and prevent them from occurring. In this section, prevention techniques used against different types of SQLI attacks are described.

Web applications created with HTML are always vulnerable to SQL injection attacks. By entering the HTML command as input, attackers can manipulate the program to get what they want. SQL-DOM is one of the protections created to handle embedded HTML commands. SQL-DOM can convert HTML into structured data, making it difficult for hackers to enter HTML commands as inputs so that conversion is not used.

Another block method is SQLrand. The best way to convert an instruction-set randomization application to the SQL language is to add a random number to the alternate end, where the attacker cannot guess the sub-number, and no injection he can make. Accordingly, according to Lloyd and Chromitis, "any malicious user attempting an SQL injection attack will be frustrated because the user rating entered in the" random "query will always be classified as a set of keywords, resulting in an expression error" [2].

Many preventive methods, such as AMNESIA, SQL Check, SQL guard, and CANDID, have been shown to be successful in preventing SQL injection, but they have not yet been able to prevent all types of SQL injection. All of the methods listed below have failed to prevent a stored practical attack, but the technique proposed by Hound et al. It uses a positive method of pollution and syntax evaluation to prevent all types of SQL injections.

According to Madman and Munch [18] staining is the best way to prevent SQL injection. It is completely dynamic because it is accurate and efficient.

SQL-DOM: This technology uses standard analysis and application code runtime tracking to detect and prevent SQLI attacks. The four main steps are hotspot identification, SQL-query modeling, tool activation and runtime monitoring. This technique is used to prevent totology, illegal, pickpocket, union, inference, alternative encryption, SQLIA, SQLI + DNS smuggling, but does not prevent saved procedures. [1]

SQLrand: This technique uses SQL randomization. Whenever the attacker tries to pay for the attack, the database analyst temporarily stores it and exits. This technique is used to prevent tautology, pickpocket, union and inference, but cannot prevent illegal, stored process and alternative coding. [2]

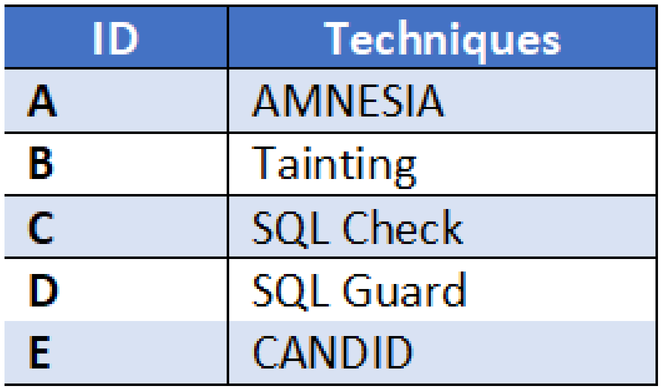
AMNESIA: This technology uses standard analysis and application code runtime tracking to detect and prevent SQLI attacks. It consists of four steps: defining hotspots, generating SQL query forms, activating the tool, and tracking runtime. This technique is used to prevent Tautology, illegal, pickpocket, union, inference and alternative encryption, but does not prevent stored procedures. [3]

Tainting: This highly automated method uses the concept of positive stain and awareness syntax evaluation to combat SQLI attacks. This technology uses reliable data sources and only allows data from these reliable data sources. This technique is used to prevent Tautology, illegal, pickpocket, union, inference, alternative encryption and stored process. [4]

SQLCheck: Using context-free grammar and translator analysis techniques, this algorithm prevents command injection attacks. This technique is used to prevent Tautology, illegal, pickpocket, union, inference and alternative encryption, but does not prevent stored procedures. [5]

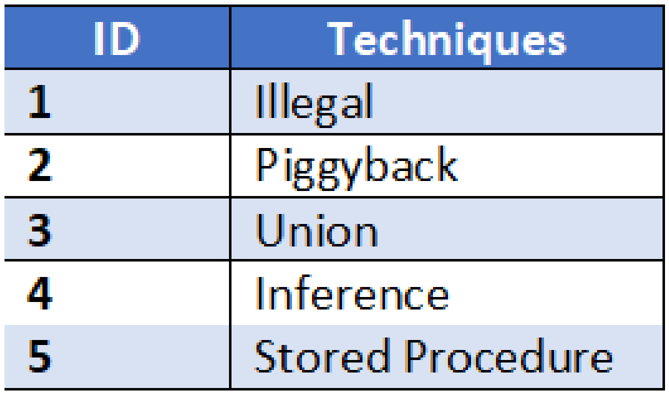
SQLGuard: compares the runtime of user input with the analysis tree in the SQL report before and after user input. Technology detects and eliminates SQLI attacks by combining the proposed query framework with instant query. This technique is used to prevent tautology, illegal, pickpocket, union, inference and alternative encryption, but does not prevent stored procedures. [6]

Candid: SQL injection attacks were detected by comparing the query structure that can program filter inputs. This technique is used to prevent tautology and to some extent against illegal, pickpocket, union, inference and alternative encryption, but cannot prevent stored practices. [7]



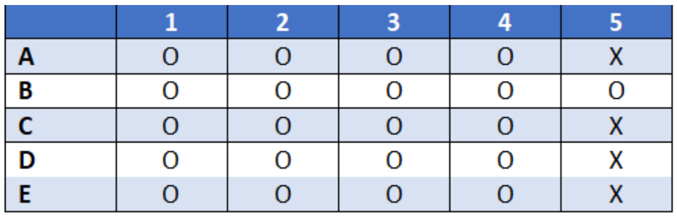
ID	Techniques
A	AMNESIA
B	Tainting
C	SQL Check
D	SQL Guard
E	CANDID

Fig. 10. Prevention Techniques



ID	Techniques
1	Illegal
2	Piggyback
3	Union
4	Inference
5	Stored Procedure

Fig. 11. Attack Types



	1	2	3	4	5
A	O	O	O	O	X
B	O	O	O	O	O
C	O	O	O	O	X
D	O	O	O	O	X
E	O	O	O	O	X

Fig. 12. Comparison of prevention techniques

VI. CONCLUSIONS

This report allowed me to present some of the most popular SQL injections, thus highlighting the importance of database security. These injections make it possible to access, modify or delete data, snatch the user's account, access the database structure, violate management rules, or obtain some information about the server. Some simple protections prevent the database from falling victim to such attacks, such as blocking certain keywords, using escape functions, or disabling error messages.

REFERENCES

[1] R. A. Katole, S. S. Sherekar and V. M. Thakare, "Detection of SQL injection attacks by removing the parameter values of SQL query," 2018 2nd International Conference on Inventive Systems and Control (ICISC), Coimbatore, 2018, pp. 736-741, doi: 10.1109/ICISC.2018.8398896.

[2] L. Ma, D. Zhao, Y. Gao and C. Zhao, "Research on SQL Injection Attack and Prevention Technology Based on Web," 2019 International Conference on Computer Network, Electronic and Automation (ICCNEA), Xi'an, China, 2019, pp. 176-179, doi: 10.1109/ICCNEA.2019.00042.

[3] E. Pearson and C. L. Bethel, "A design review: Concepts for mitigating SQL injection attacks," 2016 4th International Symposium on Digital Forensic and Security (ISDFS), Little Rock, AR, USA, 2016, pp. 169-169, doi: 10.1109/ISDFS.2016.7473537.

[4] P. N. Joshi, N. Ravishankar, M. B. Raju and N. C. Ravi, "Encountering SQL Injection in Web Applications," 2018 Second International Conference on Computing Methodologies and Communication (ICCMC), Erode, 2018, pp. 257-261, doi: 10.1109/ICCMC.2018.8487999.

[5] A. Jana, P. Bordoloi and D. Maity, "Input-based Analysis Approach to Prevent SQL Injection Attacks," 2020 IEEE Region 10 Symposium (TENSYMP), Dhaka, Bangladesh, 2020, pp. 1290-1293, doi: 10.1109/TENSYMP50017.2020.9230758.

[6] C. V. Gonzalez and G. Jung, "Database SQL Injection Security Problem Handling with Examples," 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2019, pp. 145-149, doi: 10.1109/CSCI49370.2019.00031.

[7] H. Alsobhi and R. Alshareef, "SQL Injection Countermeasures Methods," 2020 International Conference on Computing and Information Technology (ICCIT-1441), Tabuk, Saudi Arabia, 2020, pp. 1-4, doi: 10.1109/ICCIT-144147971.2020.9213748.

[8] A. Sadeghian, M. Zamani and A. A. Manaf, "A Taxonomy of SQL Injection Detection and Prevention Techniques," 2013 International Conference on Informatics and Creative Multimedia, Kuala Lumpur, Malaysia, 2013, pp. 53-56, doi: 10.1109/ICICM.2013.18.

[9] G. Su, F. Wang and Q. Li, "Research on SQL Injection Vulnerability Attack model," 2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), Nanjing, China, 2018, pp. 217-221, doi: 10.1109/CCIS.2018.8691148.

[10] Li Qian, Zhenyuan Zhu, Jun Hu and Shuying Liu, "Research of SQL injection attack and prevention technology," 2015 International Conference on Estimation, Detection and Information Fusion (ICEDIF), Harbin, China, 2015, pp. 303-306, doi: 10.1109/ICEDIF.2015.7280212.

[11] M. R. Borade, and N.A. Deshpande, "Extensive Review of SQLIA's Detection and Prevention Techniques", International Journal of Emerging Technology and Advanced Engineering,, Vol. 3, No. 10, 2013.

[12] R. Johari, and P. Sharma, "A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection", In Communication Systems and Network Technologies (CSNT), 2012 International Conference on, pp. 453458, 2012.

[13] V. Nithya, R. Regan, and J. Vijayaraghavan, "A Survey on SQL Injection attacks, their Detection and Prevention Techniques", Int. J. Eng. Compu. Sci., Vol. 2, No. 4, pp. 886-905, 2013.

[14] C. Anley, "Advanced SQL Injection in SQL Server Applications", White paper, Next Generation Security Software Ltd., 2002.

[15] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks," ACM Trans. Inf. Syst. Secur., vol. 13, no. 2, pp.1-39, 2010.