A Comparative Analysis of Techniques to Detect and Prevent SQL injection

Gandharva Deshpande

Department of Computer Science and Engineering, University of California, Riverside.

Nikita Aware

Department of Computer Science and Engineering, University of California, Riverside.

Abstract

SQL injection (SQLI) is one of the most well-known web hacking attacks. The OWASP organization (Open Web Application Security Project) listed SQL injections as the top threat to web application security. in their OWASP Top 10 2017 document. [1] This paper focuses on analyzing the SQL injection attack and understanding the root causes of the vulnerabilities in the language. Further, it proposes a comparative study of different techniques that are used for the detection and prevention of SQL injection attacks using a case study approach where we use an implemented model as a representative for the concerned approach. We compare the classical prevention approaches that use program analysis techniques like static and dynamic analysis to combat SQLIA. We further discuss and contrast the positives and the limitations of approaches that make use of the classical symbolic and concolic execution techniques alongside the modern-day Machine learning approach..

keywords: SQL, SQLI, SQLIA, Detection, Prevention, Static Analysis, Dynamic Analysis, Machine Learning

1. Introduction

1.1 Motivation

Cybercriminals have been notorious for exploiting the vulnerabilities in SQL largely over the past few decades which has made SQL injection attacks (SQLIA) one of the most common attacks. A recent study by Gartner Group [10] conducted on 300 plus Web sites showcased that most internet websites could be vulnerable to SQLIAs. In reality, SQLIAs have even successfully targeted renowned victims such as Travelocity, Guess Inc., etc.

SQLIA occurs with the infusion of malignant code in SQL queries, by means of webpage input. SQL injection attack occurs when the attacker inserts statements or operators externally into the query with the motive of bypassing authentication, extracting, modifying, or deleting data, evading detection, database fingerprinting, performing privilege escalation, executing remote commands, or access of the metadata.

Researchers have been proposing various methods to handle the SQL injection problem, however, each approach has its own set of pros and cons. A lot of new approaches have been suggested in the recent past to tackle this issue which opens a broad scope for analysis of these techniques. For each of these prominent techniques, we aim to discuss and analyze the positives and negatives providing a brief description of the core idea behind how the SQL injection attack was addressed respectively.

1.2 Our Contributions

The paper primarily aims to make the following contributions:

- Analyze and perform a comparative study on a few different modern (Machine Learning), and medieval (Static & Dynamic Analysis) techniques that prevent SQL injection using classical approaches like symbolic and concolic execution in their program analysis methodologies.
- Discuss the associated problems, limitations, and advantages pertaining to these techniques.
- Implement and demonstrate the prevention techniques to garner further insights into their capabilities and limitations which will help organizations and developers choose the technique best suited for them.

1.3 Paper Layout

The paper begins with an abstract of the problem (SQLI) that is being analyzed following which the motivation for the concerned research is presented. Further, it dives into an introduction in Section 2; talks about SQL as a Data querying language in Section 2.1, and then briefly discusses the SQL injection with demonstrations in Section 2.2. After which, different types of SQL injections are discussed in Section 2.3. From section 2.4 - Section 2.6, we discuss the various approaches to analyzing SQL codes viz. Static Analysis, Dynamic Analysis, and Machine learning. In Section 3, the paper talks about different techniques to detect SQL injection attacks. We start with discussing AMNESIA in Section 3.1 and discuss its working and limitations in Section 3.1.1 and Section 3.1.2 respectively. Then, we discuss WAVES in section 3.2.1 which makes use of dynamic analysis contrary to AMNESIA's static analysis. Section 3.2.2 discusses the limitations of WAVES and dynamic analysis techniques. Section 3.3 is dedicated to machine Learning approaches whereas in Section 3.3.1 we discuss a typical implementation of Naive Bayes which is a probabilistic prediction model followed by its limitations in section 3.3.2. In section 3.3.3, we talk about the Deep Neural network-based model SQLNN. Finally, we discuss related work, future work, and Conclusion in Section 4, Section 5, and Section 6 respectively.

2 Background

2.1 SQL

Structured Query Language (SQL) is a language that is primarily used to perform definition, manipulation, and control operations on relational databases. It was developed in the 1970s allowing database administrators to query data however with time it was normalized as a query language and became widespread due to its ease of use. Today, developers extensively use SQL for Data Analytics and writing integration scripts.[2]

2.2 SQL Injection

Following is an example of a typical SQL injection attack.

```
<?PHP

$query = "SELECT * FROM users WHERE userid = '" . $_POST['userid'] . "'";

$query .= " AND password = '" . $_POST['password'] . "'";

?>
```

The above POST request is sent to the database if altered with the following Query :

```
SELECT * FROM users WHERE userid=" VALID USERID" OR 1=1;
```

would always result in a non-empty result which could potentially provide the attacker access without entering the password leveraging the property of the 'OR' logic gate which outputs 'TRUE' if even a part of the expression is 'TRUE'.

We demonstrated the SQL injection on a website built using PHP with a MySQL database connection.

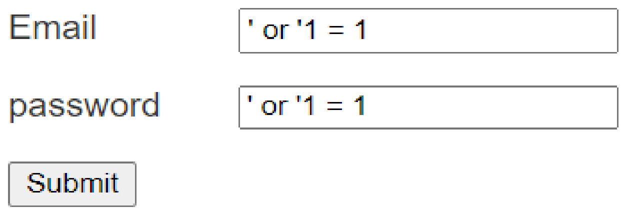
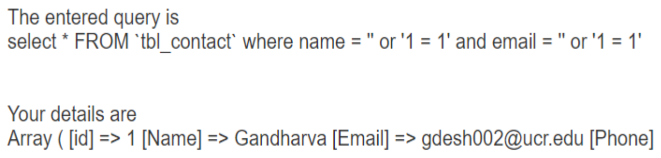


Fig 1. a

The input Form in Fig. 1.a redirects to the following:



```
The entered query is
select * FROM `tbl_contact` where name = '' or '1 = 1' and email = '' or '1 = 1'

Your details are
Array ( [id] => 1 [Name] => Gandharva [Email] => gdesh002@ucr.edu [Phone]
```

Fig 1. b

The user input channels serve as the primary vector for SQL injections and the best solution to the problem is controlling and scrutinizing the user input to watch for patterns that could potentially lead to an attack. A lot of research has been going on techniques to tackle such attacks. However, introducing new techniques opens up a huge scope for research on comparative analysis of these techniques which compares the state-of-the-art approaches to the modern techniques.

2.3 SQL Injection Attacks: Types

The first major distinguished classification of SQLI attacks was made by Halfond et al. on the basis of the intent behind the attack in 'A Classification of SQL Injection Attacks and Countermeasures (2006)'.[6] They primarily classified SQL attacks into the following types.

1. Tautologies: The ultimate goal of a tautology-attack is to insert a code snippet inside conditional statements such that they always result in TRUE.

   Eg: The aforementioned query in Fig 1. b

   SELECT * FROM TABLE WHERE NAME "" OR "1 = 1".

   Since, clause 1 = 1 is always TRUE, it qualifies as Tautology.

2. Union Query: In this type of attack, the attacker targets a parametric vulnerability in order to change the data set returned by a query.

   ```
   SELECT Email, Phone FROM userTable WHERE Id= 1 UNION ALL SELECT Messages, 1 FROM messageTable.
   ```

   The use of the UNION clause causes the result of the first query to be unioned with all the rows in messageTable.

3. Piggybacked Queries: Most databases use ';' as a character that signifies the end of a query. This type of attack is observed when the attacker inserts one query along with another.

   ```
   SELECT name, email FROM userTable WHERE name='gandharva' AND email =0; drop table userTable.
   ```

   The use of semicolon pertains to the end of a query. The second query which is inserted after is malicious and may drop the whole table. Stored

4. Alternate Encodings: They are used to evade the security mechanism and avoid defensive detection by modifying the injected text. These attacks are prominently executed with the char() function that is used to encode hexadecimal values but return actual characters
5. Inference: They are targeted at blocking feedback provided by websites in case of errors. Blind injection and timing attacks are a few types of inference-based attacks.

   Assume the following query,

   ```
   SELECT name, email FROM userTable WHERE name = 'gandharva'' and 1 =0 -- AND password = 'abc'.
   ```

   The attacker might receive an error message for this query, however, he can try his luck by changing the query due to the lack of confidence between input validation or logical error

   ```
   SELECT name, email FROM userTable WHERE name = 'gandharva'' and 1 =1 -- AND password = 'abc'
   ```

6. Stored Procedures: They are often used for DDOS, DOS types of attacks when the attacker aims at executing a procedure already stored in the database.

2.4 SQL: Static Analysis Approach

Static Code Analysis is a set of primary rules set to analyze the source code at compile time using techniques like symbolic, concrete, and concolic executions to identify potential vulnerability sites, bugs, and errors. The static program analysis of SQL ensures that the verification and testing process is automated and fastens the code review process. It helps detect sites prone to SQLIA and helps developers prevent the attack. However, the major drawback of static analysis of a program lies in the presence of False positives and False negatives. Further, it is also time-consuming and suffers from path explosion in cases of symbolic execution. Various approaches to mitigate these problems have been stated and are currently being researched.

2.5 SQL: Dynamic Analysis Approach

Contrary to the static examination of code, Dynamic analysis refers to testing and evaluation of code by executing it in real-time. It is performed with the primary objective of bug tracing and error detection at runtime rather than statically analyzing the code. However, the primary obstacle that lies in the way of using dynamic analysis to analyze Web applications is providing efficient interface mechanisms. Moreover, Dynamic analysis often poses problems with authentication and often results in heavy consumption of resources.[9] Various approaches have been put forward by researchers to perform Dynamic analysis on SQLIA prone sites to learn about the behavior of the program in case of attacks.

2.6 SQL: Machine Learning Approach

The trend of solving the problem of SQLIA detection and prevention using supervised and unsupervised learning techniques has been prevalent among researchers in the past decade. With the introduction of Deep Learning algorithms, massive research has been taking place in the field of applying Machine Learning to security threat detection. However, the major concern lies in the computing resources and time spent to execute such complex algorithms. Recently, immense research has been done on the use of ML algorithms like Naive Bayes, Neural nets, SVMs etc to detect SQLIAs.

3. SQLI Detection

Research in the field of SQLI detection has been prevalent and most developers write security mechanisms to protect their systems. However, these are not enough to combat SQLI in an absolute sense. Hence, researchers have subsequently proposed numerous techniques to detect SQL-based attacks. We will discuss a few of them and analyze the strengths and weaknesses of these techniques.

3.1 AMNESIA [8]

3.1.1 AMNESIA - Working

W. Halfond et al proposed AMNESIA as a technique to detect SQLI using static Analysis and runtime monitoring. This technique was primarily designed for Java-based web applications.

It uses static analysis to create a model consisting of legally generatable queries by the web application. Then, it dynamically checks the compliance of the generated queries with the model which was built using static analysis. This is called runtime monitoring. Any query which could potentially violate the statically generated model is classified as an SQLIA (SQL injection Attack) and prevented from execution.

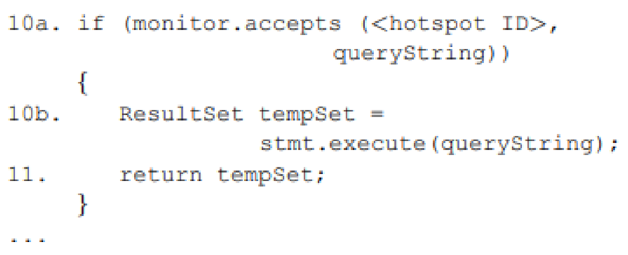
The system uses a four-step procedure.

1. Identify hotspots: Static Analysis of the program is used to identify sites that are prone to SQLIA. In Java-based applications interactions with the database occur through special methods in the JDBC library. These functions are primarily used for the detection of such hotspots. Eg:

   Connection con = DriverManager.getConnection()

2. Build Model: SQL Query model is built at each hotspot by generating all possible queries that could be executed at that location in the codebase. AMNESIA uses JSA (Java String Analysis) technique to generate grammar that is used in static analysis. With the help of JSA, the system defines a grammar for the SQL statements that could be generated and performs a depth-first search traversal on the generated parse Trees. It performs symbolic execution to determine the possible sets of SQL statements for the respective hotspot.

3. Instrument Application: A function call is being made at each hotspot to the runtime monitor.



```
10a. if (monitor.accepts (<hotspot ID>,
                          queryString))
     {
10b.    ResultSet tempSet =
                   stmt.execute(queryString);
11.     return tempSet;
     }
...
```

Fig 3.a

   Fig 3.a demonstrates, the database operation on line 10.b is secured by a call on line 10.a to the monitor.

4. Runtime Monitoring: Dynamically generated queries are parsed through the SQL Query model and violating queries are detected and prevented from executing.

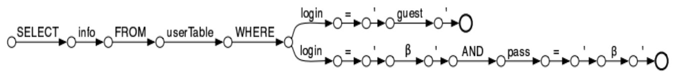
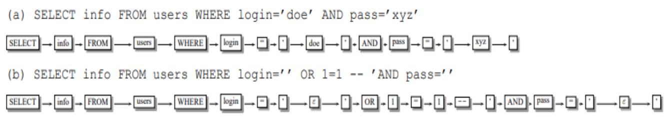


Figure 3: SQL-query model for the servlet in Figure 2.



(a) SELECT info FROM users WHERE login='doe' AND pass='xyz'

(b) SELECT info FROM users WHERE login='' OR 1=1 -- 'AND pass=''

Fig 3.b

Fig 3.b illustrates the automata generated for a hotspot. The first Query passes the automata however, the second query fails due to the presence of the 'OR' clause and is rejected by the automata, thereby is classified as SQLIA.

3.1.2 AMNESIA - Limitations

1. The primary limitation of the system is that its strength solely lies in the strength of the static analysis.
2. Since it uses symbolic Execution, the biggest problem it could face is path explosion.
3. It is specific to Java-based web applications and may not be useful for other web applications.
4. Since the approach uses grammar-based parsing it is subjected to cases of ambiguity and special cases where the attacker could generate an inferential attack after understanding the design of the automata.
5. The use of static analysis often results in overhead due to the large number of paths generated.

3.2 WAVES [9]

3.2.1 WAVES - Working

In contrast to AMNESIA's static analysis for SQLIA detection, Y. Huang et al. use dynamic analysis to solve the problem. The mentioned approach WAVES which stands for Web Application vulnerability and Error scanner; uses carefully designed malicious patterns as inputs and are purposefully used as input providing developers an insight into the code's behavior under an attack. This approach doesn't need prior source code since it uses a black-box approach.

WAVES uses a crawler that detects all the sites in the HTML code pertaining to the <FORM> tag since that serves as the entry point for most attacks. However, this approach was found to be erroneous hence, they used reverse Engineering to enhance vulnerability detection.

The success of the SQLIA largely depends on the script present on the server-side for input validation. Weak validations often lead to False Negatives which are combatted using Deep injection in WAVES. It uses a self-learning knowledge base called IKM (Injection knowledge manager) which learns which variables to put in the injection pattern generating valid data for the rest. After injection, WAVES analyzes the resulting pages with the help of a Negative Response Extraction algorithm that eliminates the False negatives.

3.2.2 WAVES - Limitations

1. WAVES fails to detect Private information Disclosure and Denial of Service exploits.
2. It generates large overheads in cases of sophisticated impact strategies.

3.3.1 Naive Bayes - Working [10]

Naive Bayes is a statistical probabilistic machine Learning algorithm primarily based on the Bayes theorem for the conditional probability distribution. The probability of a query being an attack is calculated based on the number of matching features for a given input query.

The algorithm is divided into 3 steps.

1. Preprocessing: In this step, each query is split into tokens and categorized into types.

   Eg: SELECT * FROM USER OR „B‟=‟B‟--;

   The above query is tokenized into a token Table as follows:

TOKEN TYPE	OCCURRENCES
KEYWORDS	2
NUMERALS	2
PUNCTUATION	5
OPERATORS	1
COMMENTS	0
ROLES	0

Table 1. a

2. Probabilistic classification: The probability of the query being malicious is then calculated by the Bayesian classification model. This process is performed using the previously obtained features (token types) for a tagged dataset. Thus, through this process, the model is trained to classify dynamic queries.

3. Final Classification: The final classification stage is when the model is actually tested against unknown queries. This is performed by utilizing the Bayes rule and prior generated values.

3.3.2 Naive Bayes - Results

	Accuracy	*Precision*	*Recall*
Naïve-Bayes Algorithm	93.3	1.0	0.89

3.3.3 Naive Bayes - Limitations

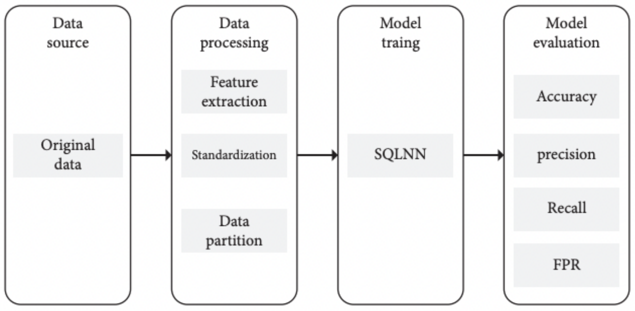
1. The success of the Naive Bayes Model largely depends on the dataset used for training.
2. Naive Bayes classification is often not absolute because it assumes that every token independently affects the outcome. However, the existence of tokens in a query is largely interdependent.
3. Probability calculation for large datasets and queries during runtime could create considerable overheads.

4. This model often faces the problem of false positives and special techniques to be taken into account to mitigate this problem.

3.4.1 SQLNN (Deep Learning) - Working [12]

Neural networks are the hotspot of the current era. Almost every problem of classification and detection has space for a neural network-based model. SQLIAs have been no exception and large research is being carried out to tackle the problem using Feedforward, Recurrent neural networks, and LSTM architectures.

Wei Zhang et al. propose SQLNN; a novel approach to tackle SQLIA using Deep Neural networks. SQLNN uses SQL injection data from which relevant features are identified and extracted, which are then used to train the model. Their approach to the problem is as follows:



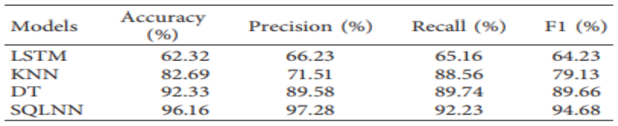
1. Preprocessing: The system begins preprocessing data by tokenizing each data fragment where each token represents one word. A data fragment is defined as a set of tokens. In this case, the model filters out tokens with an occurrence rate > 0.8 and # of occurrences < 2.
2. The system then delves into the training phase where it uses the distinguished features and Activation functions like RELU, along with backpropagation mechanisms.

3. Various approaches have been presented along with different activation functions, and algorithms like ADAM, SGD, etc for the training purpose of the model.
4.

5. The choice of hyperparameters largely influences the accuracy of deep neural networks.
6. After training the Neural network, an extensive evaluation is performed by comparing the model with techniques like RNNs and LSTMs.

3.4.2 SQLNN (Deep Learning) - Results

Table 8: Model comparison results.



Models	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
LSTM	62.32	66.23	65.16	64.23
KNN	82.69	71.51	88.56	79.13
DT	92.33	89.58	89.74	89.66
SQLNN	96.16	97.28	92.23	94.68

3.4.1 SQLNN (Deep Learning) - Limitations

1. The primary limitation of the model is the consumption of resources like memory.
2. Neural network models are often subjected to overfitting based on the choice of the dataset if early stopping is not implemented.
3. Infusing additional parameters increases the accuracy of the model enhancing the performance but has a tradeoff with time for model training.

3.4.2 Reinforcement learning

Reinforcement learning is a branch of ML that facilitates the use of an agent which is allowed to make a set of possible actions and in turn receives rewards for actions that lead to a goal and punishments for actions that take it away from the goal. This technique is best used to maximize the reward.

The use of reinforcement learning to tackle the problem of SQLIA is relatively novel and research in this field has just started to take place. As a case study, A problem was formulated as follows: to convert the SQLIA detection and prevention problem into a reinforcement learning problem. The agent was directed to obtain code snippets or data from a website through an SQLI vulnerability. The set of actions allowed to the agent was inputting queries to the system that would access the system. Access to the system would provide the agent with a reward. The aforementioned example is just one way of utilizing Reinforcement learning to find ways to mitigate SQLIA.

Although research in this field is showing promising results it is still new and blooming. Limitations like static environment, choosing the right heuristic, and prior knowledge of vulnerability location are some of the obstacles that need to be better tackled. [11]

A comparative study of SQLIA detection and prevention techniques based on selective parameters

	Static Analysis	Dynamic Analysis	Machine Learning
Case study	AMNESIA	WAVES	SQLNN, Naive Bayes, Reinforcement learning
Strengths	Strong detection mechanism, faster in a few cases.	Identifies runtime vulnerabilities, offers protection without code access	Higher accuracy, promising results for Reinforcement learning and Neural NW models
Weaknesses	Path explosion, language-specific	large overheads, can't detect DOS and sophisticated impact types	Overfitting, False positives, Static Environment assumption for reinforcement learning
Performance	Satisfactory	Satisfactory	Excellent but largely depends on training data
Memory Consumption	High only in case of large trees for symbolic execution	Low; since it's precisely dynamic and used for runtime analysis.	High; Neural network models and Probabilistic models consume huge computation resources
Nature	Detective	Detective	Preventive and Detective
Evaluation	Based on source code tags	Dynamic (Runtime Query Generation)	Based on the train and test Data using Metrics like F1, Recall, precision and Accuracy

Table 1.c

4 Related Work

Research in the field of SQLIA (SQL injection attack) has been prominent for the past few decades. In 2005, William G.J. Halfond et al proposed AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks where they used a combined approach involving static and dynamic analysis.[5] With the introduction of Machine Learning; Zhichao Zhang et al proposed a neural network-based approach to tackle SQL injection vulnerability.[4] One of the most recent techniques to handle the attack was mentioned in 'SQL Injection Attack Detection and Prevention Techniques Using Deep Learning' by Ding Chen et al. [3]. The mentioned approach uses a convolutional neural network (CNN) and a multilayer perceptron (MLP) to compare the initial experiments and then uses the Word2vec model and deep neural network to detect SQL injections.

5 Conclusion

Our proposed research is primarily directed at contrasting and comparing the various approaches used to tackle the SQLIA in the past few decades. We analyzed the classical approaches that use static and dynamic analysis techniques with symbolic and concolic executions through case studies like AMNESIA and WAVES respectively. We discussed how these techniques are used to combat SQLIA and the limitations associated with each of them. Furthermore, we discussed the advent of Machine learning techniques in the field. With the example of the Bayesian classification model and SQLNN, we compared how Machine learning models fared in the task of detecting and preventing SQLIAs. Finally, we presented how the novel reinforcement learning technology is being introduced to tackle the attack discussing its advantages and limitations.

6 Future Work

We aim to achieve the following milestones and continue our research.

- We will try to implement and demonstrate the aforementioned techniques to garner better insights into the limitations and advantages of the respective techniques.
- Dive into finer granularity and the impact of the mentioned techniques.
- Generate a complete analytical chart comparing these approaches.
- Study more about the novel Deep learning and reinforcement learning implementations and compare them with the classical static and dynamic program analysis techniques and discuss the overhead, runtime, accuracy, and memory tradeoffs.

7 References:

[1] https://owasp.org/

[2]https://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf

[3]https://iopscience.iop.org/article/10.1088/1742-6596/1757/1/012055/pdf

[4] Zhichao Z 2016 Computers and Modernization. C 10 pp 67-71

[5]https://viterbi-web.usc.edu/~halfond/papers/halfond05ase.pdf

[6] A Classification of SQL Injection Attacks and Countermeasures William G.J. Halfond et al.

[7] Comparison of SQL injection detection and prevention techniques (Conference: Education Technology and Computer (ICETC), 2010 2nd International)

[8] AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks by W. Halfond, Orso et al.

[9] Web Application Security Assessment by Fault Injection and Behavior Monitoring Yao-Wen Huang, Shih-Kun Huang,

[10]https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6993127

[11]https://portswigger.net/daily-swig/machine-learning-offers-fresh-approach-to-tackling-sql-injection-vulnerabilities

[12]https://downloads.hindawi.com/journals/scn/2022/4836289.pdf

[13]https://www.ijcsmc.com/docs/papers/August2017/V6I8201701.pdf

[14]https://iopscience.iop.org/article/10.1088/1742-6596/1757/1/012055/pdf

[15]https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/