# Positive and Negative Regular Expression Filter in Prevention of SQL Injection Attacks

Ryandito Diandaru 13519157
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : 13519157@std.stei.itb.ac.id

***Abstract*—SQL Injection is a type of attack on websites that uses SQL databases. SQL injection attacks happen when the attacker manages to execute a command to the victim database, resulting in data breach or even on extreme cases, loss of data. Prevention procedures regarding SQL Injections are already built, and two of them are the technique of user input validation and data sanitization. Regular Expression is then used to implement both of these techniques, resulting in two categories which are positive filter, a regular expression made to only accept kind strings like that of the target database and negative filter which accept everything but some specified illegal character or string. Positive filters are stricter, therefore generally more secure. Whereas negative filters are more open to change and less prone to syntax errors.***

***Keywords—SQL; injection; regular; expression; filter; sanitization***

## I. INTRODUCTION

In this modern world of computing and internet technology, us humans have managed to find a way to store information in a digital form stored inside electronic warehouses we call databases, accessible over computers. Moreover, today's technology also lets us store information in what we call "the cloud", which means we don't have to store the information belonging to us inside our own electronic data warehouse, instead we use the help of companies that provide data storage services like Google Drive and Apple's iCloud just to name a few. This approach is considered useful since though it's common nowadays that a computing device provide up to 2 terabytes of storage, most people still prefer to store things on the cloud. The motivation of storing things on the cloud is not only driven by the need of more storage, but instead, it is also driven by the need for the files to be accessible by more than one person, i.e., the need for it to be shared, most commonly by a group of people under the same community, organization, or institution.

Databases discussed in this paper however, is discussed in a slightly different manner than storing files in a cloud as one does with Google Drive or iCloud. However, databases in this manner would still revolve around storing information inside data warehouses but the difference lie on the type of data that gets stored.

Consider a paper company with a number of employees and departments along with the information of the managers of each department. A well-run company would keep all the information regarding their employees and everything that comes with it including, salary phone number, the department they belong to, etc. The data stored would not merely serve as a keepsake, data organization is crucial in a company management, most companies refer to their databases to make decisions regarding company policies.

Such collection of information nowadays is commonly stored in a relational database system, managed by a Database Management System. A relational database is a type of database that allows access data and their relations with other pieces of data in the same database, or to be put more simply, one can call a database is in a relational database form if the data tuples are stored in a table. In our paper company example, information about each employee's name, phone number, salary, department, etc. could be stored in a table named *Employee* and department names, office addresses, etc. could be stored in a table named *Department*, both of them in a database named *CompanyData*.

Aforementioned database is accessed using a Structured Query Language (SQL), which is a programming language with which modification and manipulation of data from a database table is possible. With SQL, we can extract relevant information from even more than one table through queries using predefined keywords e.g., SELECT, JOIN, WHERE, etc., with each keyword serving its own purpose. A database is then managed using a Database Management System (DBMS), with sometimes slightly different query syntaxes for differing DBMSs. A few of the commonly used DBMSs are MySQL, PostgreSQL, SQLite, and MariaDB.

Data is stored in this manner since it provides an effective way to store, retrieve, and modify data. Unfortunately, data stored using SQL databases also raises a security threat of data breach. SQL Injection is a data breach threat in which the attacker breaks in and extract data "hidden" by the data owner, or in more extreme cases, the attacker may delete the data. For example, an attacker can get information of private information, for instance addresses of a bunch of people from an organization through an SQL Injection.

Although the security threat has been known for around 20 years since this paper was written, it's still the number 1 security threat according to OWASP's top 10 web vulnerabilities year 2017 [1]. Security measures and procedures

has been made by engineers all around the globe and yet the threat is still a relevant issue to this day.

One of the ways to prevent SQL Injection attacks is to sanitize inputs so that queries that get passed into the database guarantees a valid input and contains no malicious subqueries. One can almost tell right away whether a query contains a malicious subquery given the expression, but it is a matter of whether computers can detect said malicious subqueries given an expression.

Computers are able to identify a pattern as a substring of a given string by using pattern matching algorithms, be it Knuth-Morris-Pratt or Boyer-Moore, or any other existing algorithm. Another more common way is to use Regular Expressions. Regular Expressions are supported by a number of programming languages including Java and PHP just to name a few.

In this paper, queries passed into the database will be seen as strings and followed with a discussion about query sanitization using Regular Expression patterns.

## II. Literature Review

### A. SQL Injection

SQL Injection attacks comes from user inputs from websites, where the attacker can input an SQL command inside a user input field. The attacker puts in something other than the expected value type to be input from said fields and thus bypasses a command to the SQL database. When the attacker knows that a certain website is vulnerable to such SQL Injection attack, they can virtually retrieve any information or do any sort of update to the database, since the attacker can already run queries. This situation is called a data breach, where unauthorized parties manage to gain access to data that they originally don't have access to. In a more extreme case however, data breach situation may not happen at all because the attacker decides to delete all data from the database.

Consider an online shopping website and the example SQL query below.

```
SELECT *
FROM Items
WHERE Name = '<name of the item>';
```

Let our online shopping website has a search bar feature where customers can search for items by their names. An SQL query to search by item name would look something like the given query above, with the <name of the item> placeholder would be replaced by the user input. For example, if a user wants to search for item 'Apple' then the query would look like the following:

```
SELECT *
FROM Items
WHERE Name = 'Apple';
```

This will work for as long as inputs from the user is valid, assuming none of the item names only consists of alphanumeric characters. What gets put in the placeholder is what we need to be extra careful with. The input could contain strings of dangerous characters that are by themselves an expression of SQL and thus contains SQL queries in itself. If the user input is not sanitized and thus the website is vulnerable to SQL injection attacks, an attacker is able to pass through SQL statements with said dangerous characters. For example, the following input could get passed through and delete the entire table without the website owner knowing.

```
Apple'; DROP TABLE Items;--
```

When the above expression is to be input into the search bar, the command that gets passed to the database is to actually search for a product named 'Apple' but then drop the Items table right away. SQL expressions are divided using semicolons, notice how the above expression consisted parts of 2 different queries. If the above query gets passed, then we get the following SQL expression passed to the database.

```
SELECT *
FROM Items
WHERE Name = 'Apple'; DROP TABLE Items;--';
```

The double dash ('--') after the semicolon on the second sub-expression is an SQL comment syntax, so that the trailing characters at the end of the expression doesn't generate a syntax error.

If the above expression is successfully executed on the database, that means the Items table is successfully deleted.

### B. SQL Injection Prevention

There are a number of varying ways on how to prevent an SQL injection attack since there are also a number of ways that an SQL Injection attack can occur, but this paper is limited to the discussion of SQL Injection attacks through user input fields on the website. In this paper, SQL Injection attack prevention will only revolve around the defense strategy against malicious user inputs, i.e., by validating user inputs and sanitizing the data by detecting dangerous characters. [2]

User input validation means that anything that gets passed through to the actual SQL query is validated to guarantee that the input is in the same type/format as what needs to be passed as an input. For example, an online store that sells only fruit has item names with letters of the English alphabet. An item name-based search request from the user would have to be validated and be decided whether the input string from the user only consisted of alphabet characters. If the input is valid then the input gets passed into the actual query and declined if otherwise.

Data sanitization means that inputs are modified so that dangerous characters are safe to be passed inside the actual query that gets executed. Sanitization is usually done by adding a backslash character to non-alphanumeric characters so that

they are not treated as a part of the SQL query, but instead they are treated as a plain character. [3]

A number of programming languages already provide a function to sanitize data. For example, PHP provides the mysql_real_escape_string() function to append a backslash character to certain characters of a string.

This paper will discuss the relevance of Regular Expression patterns in both of the aforementioned SQL Injection prevention techniques.

### *C. Regular Expression*

Regular expression patterns are template patterns in which matching strings can take the form of [4]. Several programming languages support Regular Expression including PHP, the programming language this paper will discuss Regular Expression in. Regular Expression is used in exact matching, it works by defining a Regular Expression pattern and deciding whether there exists a substring that matches the aforementioned regular expression pattern. The following tables are the commonly used Regular Expression syntax, retrieved from the Cheat Sheet section of regexpal.com [5].

TABLE 1
CHARACTER CLASSES IN REGULAR EXPRESSION

| Pattern | Use |
|---|---|
| `.` | any character except newline |
| `\w \d \s` | word, digit, whitespace |
| `\W \D \S` | not word, digit, whitespace |
| `[abc]` | any of a, b, or c |
| `[^abc]` | not a, b, or c |
| `[a-g]` | character between a & g |

TABLE 2
ANCHORS IN REGULAR EXPRESSION

| Pattern | Use |
|---|---|
| `^abc$` | start / end of the string |
| `\b` | word boundary |

TABLE 3
ESCAPED CHARACTERS IN REGULAR EXPRESSION

| Pattern | Use |
|---|---|
| `\. \* \\` | escaped special characters |
| `\t \n \r` | tab, linefeed, carriage return |
| `\u00A9` | Unicode escaped © |

TABLE 4
GROUPS & LOOKAROUND IN REGULAR EXPRESSION

| Pattern | Use |
|---|---|
| `(abc)` | capture group |
| `\1` | backreference to group #1 |
| `(?:abc)` | non-capturing group |
| `(?=abc)` | positive lookahead |
| `(?!abc)` | negative lookahead |

TABLE 5
QUANTIFIERS & ALTERNATION IN REGULAR EXPRESSION

| Pattern | Use |
|---|---|
| `a* a+ a?` | 0 or more, 1 or more, 0 or 1 |
| `a{5} a{2,}` | exactly five, two or more |
| `a{1,3}` | between one & three |
| `a+? a{2,}?` | match as few as possible |
| `ab\|cd` | match ab or cd |

After the Regular Expression pattern has been constructed, then pattern detection is possible. PHP detects the existence of the pattern using the series of Regular Expression functions. PHP Regular Expressions functions are the following, retrieved from w3schools.com.

TABLE 6
REGULAR EXPRESSION FUNCTIONS

| Function | Use |
|---|---|
| `preg_match()` | Returns 1 if the pattern was found in the string and 0 if not |
| `preg_match_all()` | Returns the number of times the pattern was found in the string, which may also be 0 |
| `preg_replace()` | Returns a new string where matched patterns have been replaced with another string |

## III. Filtering Inputs Using Regular Expression

SQL Injection prevention techniques discussed in this paper is limited to only discuss user input validation and data sanitization, whereas regular expression patterns will be generated to validate said inputs. Input string filtering methods will be classified into two different categories, which are Positive Filter and Negative Filters. Both of which are adaptations of the User input validation and data sanitization methods respectively.

### *A. Positive Filter*

Positive filtering means that a Regular Expression is constructed so that inputs are restricted to only be of some specific string that may or may not represent an actual object.

Consider the formerly introduced online shopping website example that only sells fruit with the assumption that for every fruit sold on the online shop, their name only contains characters that are part of the English alphabet, i.e., letters through A-Z, insensitive towards capitalization. Thus, a regular expression built should only allow string of alphabet characters. The regular expression for the fruit online shop is as follows:

```
fruitName = "/^([A-Z])+(\ ([A-Z])+)*$/i"
```

The above expression means that a fruit name should contain at least one alphabet character and also a fruit name can be consisted of more than one word, separated by no more than one space character. The above expression forces that the beginning and the end of the input should be part of the input string, i.e., the input should only be the word being searched.

Although a positive filter for checking fruit names, i.e., strings only containing letters of the alphabet with no more than one consecutive space character as a delimiter has been successfully constructed, positive filtering is not limited to such expressions. Countless of possible regular expression patterns can be constructed to check various naming rules.

### *B. Negative Filter*

Negative filtering means that a Regular Expression is constructed so that it accepts everything except predefined "forbidden" characters.

On the previous section, the online fruit store example can only sell fruit whose name characters exist in the English alphabet, with no more than one consecutive space character. In the case of there exists a fruit that has a name containing a number for instance, or in the case where users can search for fruit names in different languages which requires special Unicode characters, a positive filter regular expression would involve enumerating every special character there is to know when it comes to naming fruit. A negative filter however, only checks for what should not be found in the user input, and thus the need to enumerate every character is exempted.

SQL Injection attacks that occur through input boxes on websites happen when the input gets passed to the placeholder in the actual SQL query that is going to be executed when the user presses the execute command button. It is needed to be identified which characters are forbidden so that a negative filter can guarantee that strings containing said characters never gets executed into the SQL query. The regular expression for negative filter with the set of forbidden characters being a semicolon, an apostrophe, and a double quote is as follows:

```
illegalChar = "/^[^;\'\"\(\)]+$/i"
```

The above expression will yield a match if there are no forbidden characters present in the string being checked. For single character checking, the above expression is already flexible, meaning that any other new character that one wishes to be made forbidden can be appended to the list of forbidden characters in between the square brackets.

Negative filter is however, not limited to only checking illegal characters. Oftentimes, SQL injection attacks via user input contain SQL keywords such as "DROP" or "UPDATE". Negative filter method for SQL keywords can be used to add another layer of security.

```
illegalWord =
"^((?!(drop|table|update)).)*$"
```

The above expression filters out the words "drop", "table", and "update", therefore any string input containing words mentioned will not be validated by the Regular Expression pattern. Filtering words is also flexible, more words can be appended to the string between the parentheses after the exclamation point, along with an or symbol if one wishes to add more words to be filtered out.

## IV. Implementation and Analysis

From the fruitName, illegalChar, and illegalWord regular expression patters, will be conducted an experiment with a number of testcases to verify its ability to filter out string inputs. Note that successes in the examples given do not represent success in other cases not mentioned in this paper due to the variability of the usage of regular expression. However, this experiment will show that the constructed regular expression pattern is successful in verifying the given cases. (will be explained in the corresponding sections)

### *A. Positive Filter Test*

The fruitName pattern example represents a case where the input is only allowed if every character on the string is a member of the English alphabet, followed by strings of the same kind separated with no more than one whitespace character. This paper however, is limited to only giving the solution for cases of this kind.

Experiment is conducted on phpliveregex.com on a PHP environment, result showing only 'array()' for a string testcase means the string is not found given the pattern. The following are the experiment results.

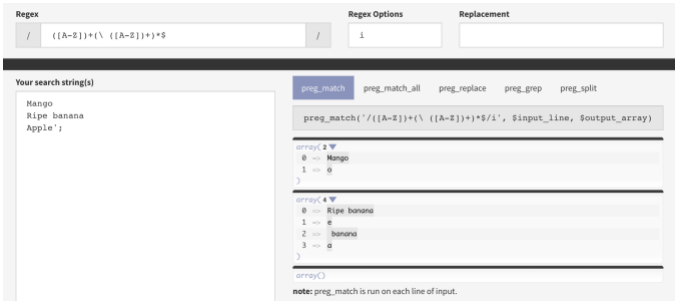


*Fig. 1.1. phpliveregex results for fruitPattern example.*

TABLE 7
POSITIVE FILTER TEST RESULT

| String | Result | Verdict |
|---|---|---|
| Mango | Found | Pass |
| Ripe banana | Found | Pass |
| Apple’; | Not found | Pass |

The results above shows that the pattern accepts only testcase strings that are valid for the pattern, i.e., testcases that does not contain any non-alphabet character.

### *B. Negative Filter Test*

To show the already given examples, Negative Filter test will be divided into two sections, the character example and the keyword example.

#### *1. Negative Character Filter*

Even though the given example of negative character almost certainly has not covered every “dangerous” there is to filter, the given regular expression is deemed open enough to newly defined forbidden characters, meaning that if there is a case where a new character is to be defined as a forbidden character, the addition to the list need not to make a new regular expression altogether, instead, one is expected to merely add said new character to the already defined set of forbidden character.

Experiment is conducted on phpliveregex.com on a PHP environment, result showing only ‘array()’ for a string testcase means the string is not found given the pattern. The following are the experiment results.

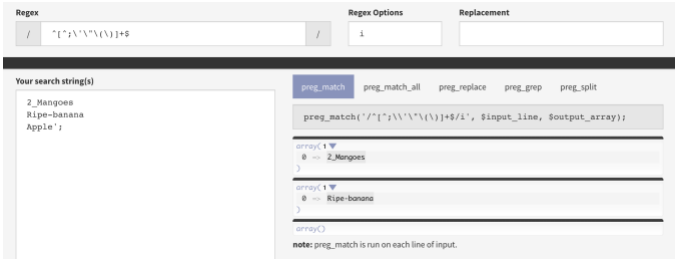


*Fig. 1.2. phpliveregex results for illegalChar example.*

TABLE 8
NEGATIVE CHARACTER FILTER TEST RESULT

| String | Result | Verdict |
|---|---|---|
| 2_Mangoes | Found | Pass |
| Ripe-banana | Found | Pass |
| Apple’; | Not found | Pass |

The results above shows that the pattern accepts only testcase strings that are valid for the pattern, i.e., any string for as long as it does not contain forbidden characters. Said forbidden characters being a semicolon, an apostrophe, and a double quote.

#### *2. Negative Keyword Filter*

Similar to the negative character filter example, the given example for negative keyword filter has almost certainly not covered every risky SQL keyword there is to filter. However, if it is desired for a new keyword to be defined as a forbidden keyword, an addition to the already defined set of illegal word in the regular expression would be sufficient.

Experiment is conducted on phpliveregex.com on a PHP environment, result showing only ‘array()’ for a string testcase means the string is not found given the pattern. The following are the experiment results.

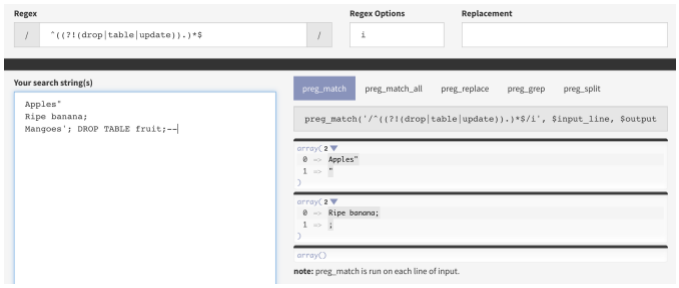


*Fig. 1.2. phpliveregex results for illegalWord example.*

TABLE 9
NEGATIVE KEYWORD FILTER TEST RESULT

| String | Result | Verdict |
|---|---|---|
| Apples" | Found | Pass |
| Ripe banana; | Found | Pass |
| Mangoes'; DROP TABLE fruit;-- | Not found | Pass |

The results above shows that the pattern accepts only testcase strings that are valid for the pattern, i.e., strings that do not contain any forbidden keyword, i.e., “drop”, “table”, and “update” (case insensitive). Also note that in practice, Negative word filtering may still yield an error since it does not filter out forbidden characters that may result in a syntax error.

After the implementation of either the positive and negative filtering, results from the regular expression checker may then be decided if whether any illegal input should be automatically modified so it becomes safe (delete characters, adding escape characters, etc.) or reject the input altogether.

### *C. Analysis*

Based on the results of tests on the three examples given, positive and negative filters have their own advantages and drawbacks.

Positive filters are generally more secure than negative filters since it only allows strings of the same kind as a certain category's name, but positive filtering limits the category name and thus possible future modification to the name may require an update to the regular expression.

Negative character filter is less prone to error since it may be used to filter out symbols that may be part of the SQL syntax. Unlike the negative character filter, the negative keyword filter would still yield SQL syntax errors and it is possible to coincidentally filter out names that are actually valid names, but it may effectively block any potential malicious SQL keyword.

TABLE 9
ADVANTAGE AND DRAWBACK ANALYSIS

| Filter | Advantage | Drawback |
|---|---|---|
| Positive | • Stricter, and thus generally more secure | • Limited to only a specific type of string<br>• Still prone to syntax errors |
| Negative character | • Less prone to error | - |
| Negative keyword | • Blocks SQL keywords altogether | • Still prone to syntax errors<br>• Limit names that may coincidentally the same as SQL keywords. |

## V. CONCLUSION

Regular expression can be used as an alternative to filtering and sanitization of database inputs. positive filtering is a regular expression construction which only accepts strings of the actual string type in the database, and negative filtering is a regular expression construction which accepts everything unless it contains anything from a given set of illegal characters/strings. Positive filtering is stricter than negative filtering, thus it is generally more secure than negative filters but it is relatively close ended, whereas negative filtering is more open to change.

## VIDEO LINK ON YOUTUBE

https://youtu.be/D0tsiJBKl7w


## ACKNOWLEDGMENT

First and foremost, I would like to say *Alhamdulillahirabbil'alamin* I am able to finish this paper in the holy month of Ramadhan, and for everything I have been given in life and the gift of life itself. To both of my parents, although no words would be enough to express how much I thank them, who are there since the day 0 of my life, and through the ups and downs of it all. To all of my professors, especially Prof. Dwi Hendratmo Widyantoro, Ph.D., who has been being my professor in the study of Algorithmic Strategies this semester, may all the lessons learnt, course-related or not, be useful for years to come. To all computer scientists and engineers throughout the ages, whose contribution has fueled technological progress and innovation to this day. To all my friends and loved ones, whose acts of kindness and support, no matter how small, shall not go unnoticed.


## REFERENCES


[1] "OWASP Top Ten Web Application Security Risks | OWASP", *Owasp.org*, 2021. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed: 11- May- 2021].

[2] A. Weiss, "Prevent Web Attacks Using Input Sanitization", *eSecurityPlanet*, 2012. [Online]. Available: https://www.esecurityplanet.com/endpoint/prevent-web-attacks-using-input-sanitization/. [Accessed: 11- May- 2021].

[3] A. Weiss, P. Rubens and S. Ingalls, "SQL Injection Prevention | How to Prevent an SQL Attack", *eSecurityPlanet*, 2021. [Online]. Available: https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks/. [Accessed: 11- May- 2021].

[4] S. Kulkarni, "An Introduction To Regular Expressions | DigitalOcean", *DigitalOcean*, 2014. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-regular-expressions. [Accessed: 11- May- 2021].

[5] "Regex Tester - Javascript, PCRE, PHP", *Regexpal.com*, 2021. [Online]. Available: https://www.regexpal.com/. [Accessed: 11- May- 2021].


## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Surabaya, 11 Mei 2021

Ryandito Diandaru 13519157