EXTENDED ABSTRACT

REGULAR EXPRESSIONS BASED SQL INJECTION DETECTION

R. Senthan,[*,1] E. Y. A. Charles,[2] and S.R. Kodituwakku[1]

[1]University of Peradeniya, Sri Lanka
[2]University of Jaffna, Sri Lanka
[*] krsenthan@gmail.com

(Published 15 October 2021)

Abstract
SQL Injection Attacks (SQLIA) are among the most significant threats for Database Management Systems (DBMS) and Web applications. SQL Injection is a technique where an attacker attaches malicious SQL statements in one of many possible forms as input for a query in the DBMS. The DBMS is tricked into executing this malicious code while processing the original query. Insufficient validation of user input is the leading cause of SQL injection vulnerabilities. Detection of SQL injection using regular expression is one among many solutions for this problem. However, the effectiveness of regular expressions in detecting all types of SQL injection attacks has not yet been established, and this work attempts such a study. By analysing the literature on SQLIAs and a data set of 318 queries (293 malicious and 25 benign), four cases of patterns of malicious queries were identified. Furthermore, regular expressions created for the four cases could correctly identify 90% of SQLIA queries with low resources and execution time.

Keywords: Database management system, web application, SQL injection attack, regular expression

1. Introduction

Today, database management systems (DBMS) and Web applications have become a most valuable and unavoidable part of everyday life. In this information technology-dominated environment, all private and public institutions do their best to make their information assets accessible online. This objective can be achieved by the use of DBMS and Web applications. DBMS is a combination of data, hardware, software and users that helps enterprises to manage their operational data. Web applications provide access to the data on the databases for users from anywhere in the world. Databases are used to store all types of information, including details of persons, institutions, objects, activities and their relations. On many occasions, the stored data is highly confidential. The web applications and the DBMS should have a mechanism to secure the data and allow access for authorised persons. It is vital not to give opportunities to intruders to extract, modify or delete the data kept in the databases. Web applications and DBMSs face large varieties of attacks to breach security measures. SQL Injection Attacks (SQLIA) are one of the severe and widespread ones among many such threats. SQLIAs are on the rise incessantly in terms of quantity as well as sophistication. SQLIAs are a severe security risk because through this attack, an adversary can gain unrestricted access to the data and applications of an institution (Almutairi et al., 2012),(Saurabh et al., 2012). Methods proposed for this problem are able to handle a portion of the full spectrum of SQL injection attacks. Detection of SQLIAs using regular expression is one such solutions. This research work attempts to determine the effectiveness of regular expressions in detecting all types of SQL injection attacks.

2. Related work on detecting SQL Injection Attacks (SQLIA)

Structured Query Language (SQL) is a programming language for define and or manipulate relational Databases. Data Definition Language (DDL) commands are used for creating database objects. Data Manipulation Language (DML) commands are for manipulating the database contents. In an SQL Injection attack, an adversary attaches a series of SQL statements into a query and maliciously manipulate the input data into action in the database (Chandershekhar et al., 2016),(Saurabh et al., 2012). As a result of this attack, the integrity of the database could be compromised and confidential content could be copied, altered or deleted. Furthermore, as a result of this breach, the adversary may gain control of and corrupt the server systems hosting the Web application (Nithya et al., 2013).

Researchers for handling the SQLIA propose many solutions. However, due to the complexity and possibility for many types of malicious queries, current approaches cannot address the full spectrum of the SQLIA. Further, this problem becomes even more complex due to the wide range of techniques available for an attacker utilising these vulnerabilities. Hence, many proposed solutions in the past apply only to part of the full spectrum of SQL injection attacks (Halfond et al., 2006).

SQL injection is a type of code-injection attack where the data given by a user in the form of a set of malicious SQL statements is added into a regular SQL query (Chandershekhar et al., 2016). SQL code injection can generally be categorised into four types based on how the malicious code is attached. They are namely, Injection through User Input, Injection through cookies, Injection through Server Variables and Second-order injection. Further, SQL Injection attacks can be generally categorised into four types based on their operation: Code injection, SQL manipulation, Function call Injections, and Buffer overflows. Code injection attacks add extra SQL commands or statements to the existing SQL statement. In SQL manipulation, the existing SQL statements are modified by an adversary. The attack by adding database functions into SQL statements is known as function call injection. These functions can be used to make operating system calls or manipulate data in the database. Finally, buffer overflows are exploits against an operating system or applications. This attack overload the memory of a system by executing arbitrary computer programme statements on a target system. This type of attack would cause the system hosting the applications to fail (Halfond et al., 2006).

It is widely accepted that the SQLIAs are due to inadequate input validation. Here the data provided by a user is not correctly validated and is accepted as an input straight away. Many studies have been carried out on detecting SQL Injection attacks, and many solutions have been proposed. These can be grouped as Detection and Prevention methods, Instruction Set Randomization, Intrusion Detection System and Proxy Server implementations, and Analysing using a threat Model (Chandershekhar et al., 2016). Several approaches and methods have been proposed to address the SQL injection attack problem. These approaches either fail to address the full scope of the problem or are able detect only a subset of the SQLIA types due to their limitations (Halfond et al., 2006). Detection of SQLIAs using regular expression is one study area among them. However, the effectiveness of regular expressions in detecting all types of SQL injection attacks has not yet been established and this work attempts such a study. Furthermore, this research work considers code injection type attacks only and provides a way to validate the SQL query by identifying either benign or malignant.

3. Methodology

Regular expressions provide a flexible and compact way to match strings of text to a common underlying pattern of the text. For example, regular expressions are used for identifying and separating various types of items (tokens) in a programme text such as keywords, variables, numbers and strings. Hence they are used in the lexical analysis phase of compilers to identify the tokens in a programme text to be passed to the syntax analysis phase. Several studies have reported the use of Regular expressions for detecting SQLIAs (Sandeep et al., 2016),(Monali et al., 2017),(Gowthami et al., 2016). However, its effectiveness in detecting all types of attacks has not been thoroughly researched (Sandeep et al., 2016). This paper reports such a study on establishing the efficiency of regular expressions for detecting SQLIA queries. By analysing the literature on SQLIAs and a data set (Stuart et al., 2017) of 318 queries (293 malicious and 25 benign), four patterns on malicious queries were identified. The four cases are:

Case 1: A tautology s=s follows an or. For example, or s=s – here s can be any character or string.
Case 2: A malicious code follows the select keyword. For example, select @@version;, select ascii('A');, select /*comment*/1;.
Case 3: The keyword or is followed by a malicious code. For example, or 'whatever' in ('whatever');, or pg_sleep(__TIME__);.
Case 4: Certain keywords are followed by a malicious code. For example, admin'*;, admin'#;, admin';, declare @q nvarchar (200) 0x730065006c00650063 ...);.

For the above four cases, regular expressions were created to detect the common patterns in each case. The regular expressions for each of the four cases are given in Table 1. Each regular expression is proposed to detect SQLIA from each of the four identified types. To analyze a SQL query, it is parsed with the four regular expressions to detect whether it matches with any of them. If a regular expression matches with a query, then it is an SQL Injection. If it is not matched with any of the four, then it is a benign SQL query.

Table 1. Proposed regular expressions for identifying SQLIA

RE 1:

```
"^.*?([[a-z&&[or]][\\W[a-zA-Z0-9]\\W]]+)(|\\s)=.*?((\\1#)|(\\1\\s\\#)|(\\1--)|(\\1\\s\\--
)|(\\1\\/\\*)|(\\')|(\\'\\1)|(\")|(\"\\1)|(\\s\\1\\s)|(\\1)$)"
```

RE 2:

```
".*?(SELECT (@@[a-zA-Z]+;)|(\\d\\;\\s\\#[a-zA-Z]+)|(\\/\\*[a-zA-Z]+\\*\\/\\d;)|\"\n" +
    "        + \"([a-zA-Z]+\\(\\);)|([a-z0-9A-Z]+\\smysql.user;)|([a-zA-Z]+\\(\\))|([a-z0-9A-Z]+\\smysql.db|\\;)\"\n"
    +
    "+                                        \"|([a-z0-9A-Z]+\\sinformation_schema.schemata;)|([a-z0-9A-
    Z]+\\sinformation_schema.columns)|(\\d+\\&\\d+)|(ascii\\(\\W+[a-zA-Z0-9]+\\W+\\);)|\"\n" +
    "              + \"(cast\\(\\'\\d+\\?[a-zA-Z0-9]\\);)|(concat\\(\\'[a-zA-Z0-9]+\\'\\,\\'[a-zA-Z0-9]+\\'\\,\\'[a-zA-Z0-
    9]+\\'\\);)|\"\n"  +"                    +  \"(if\\(\\d+\\=\\d+\\,\\'[a-zA-Z0-9]+\\'\\,\\'[a-zA-Z0-9]+\\'\\);)|([a-zA-Z0-
    9]+\\s\\(\\d+\\=\\d+\\)\\s[w+\\a-zA-Z0-9\\w+]+\\;))"
```

RE 3:

```
".*?(or (([0-9]+\\s\\--\\')|([a-zA-Z0-9]+[like]+\\'\\%)|(EXISTS))|([0-9]+\\s\\>\\s[0-9]+)|"
          +  "([0-9]+\\>[0-9]+)|(benchmark\\([0-9]+,MD5\\([0-9]+\\)\\)\\W)|(\\W[a-zA-Z]+\\W  \\>  \\W[a-zA-
    Z]+\\W)|" + "(\\W[a-zA-Z0-9]+\\W in \\(\\W[a-zA-Z0-9]+\\W\\))|([0-9]+ in \\(select @@version\\)\\W)|"
    +       "([0-9]+        [between]        [0-9]+\\sand\\s[0-9]+)|(isNULL\\([0-9]+\\/[0-9]+\\)\\W)|([a-zA-Z0-
    9]+\\Wlike\\Wchar\\([a-zA-Z0-9]+\\);)|" + "(sleep\\(__TIME__\\)\\W+))"
```

RE 4:

```
 ".*?(([admin]+\\\\'(\\\\--|\\\\#|\\\\/\\\\*))|(UNION SELECT)|(UNION ALL SELECT)|"
          +  "(\\WSHUTDOWN\\W)|(\\Wpassword\\W)|(\\Wsp_password)|(\\Wvariable)|(\\Wselect  top  [0-
    9]+)|" + "(ORDER BY [0-9]+\\;--)|(@var select @var as var into temp end \\--)|(\\Wmail=\\W)|(\\|\\|\\'[0-
    9]+)|"   +   "(admin\\W   or   \\W)|([0-9]+\\WSELECT\\W[0-9]+\\W)|([0-9]+\\W+)|(union   all   select
    \\W+version\\W+)|(UNION\\W+SELECT)|"
    +          "(\\/\\/\\*)|(\\WDR\\W+OP          tempTable\\W)|(sqlattempt1)|(sqlvuln;)|(select\\W+from
    information_schema.tables\\--)|"
    +     "(\\W+union     select\\W+from     information_schema.tables\\;)|(UNION     ALL     SELECT
    LOAD_FILE\\W+etc\\Wpasswd\\W+)|"
+ "(exec (sp)|(\\(\\W[a-zA-Z]+\\))|(\\(\\WSEL\\W+ECT [0-
9]+\\W\\))|(xp_regread)|(master\\W+xp_cmdshell)|(master\\W+xp_cmdshell\\W+ping [0-9]+\\W[0-9]+\\W[0-
9]+\\W[0-9]+\\W+)))"
```

4. Results, Discussion and Conclusion

The proposed method was tested by using java programme utilising the java.util.regex.Matcher and java.util.regex.Pattern classes. All the SQL queries in the data set were given as input and the output of the programme was obtained. Out of the 293 injection queries, the regular expressions detected 265 correctly, and the remaining 28 did not match any. Out of the 25 benign queries, all were detected as benign. The proposed regular expressions were applied for NoSQL queries (Cr0hn, 2021) and found to be producing promising results. By fine-tuning the regular expressions, the results can be improved further. In addition, this method was found to be utilising meagre resources for its operation. Hence this method can be included into any DBMS query processor easily. The study can be further extended to find the typical injection attack patterns in SQL queries using a suitable machine learning approach and automatically or manually update the regular expressions.

References

Almutairi, A. H., & Alruwaili, A. H. (2012). Security in database systems. *Global Journal of Computer Science and Technology Network, Web & Security* 12(17): 9–14.

Sharma, C., Jain, S. C., & Sharma, A. K. (2016). Explorative study of SQL injection attacks and mechanisms to secure web application database-A. *Int J Adv Comput Sci Appl* 7(3): 79–87.

Gowthami, S., & Kumar, K. P. Detecting SQL Injection Attacks in Web Application Using REGEX and Query Result Size. *International Journal of Innovative Research in Computer Science and Engineering, ISSN*, 2394-6364.

Halfond, W. G., Viegas, J., & Orso, A. (2006, March). A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE international symposium on secure software engineering* (Vol. 1, pp. 13-15). IEEE.

Monali, S., K., & Butey, P., K. (2017). An Approach for Detecting and Preventing SQL Injection and Cross Site Scripting Attacks using Query sanitization with regular expression, *International Journal of Computer Trends and Technology* 49: 237-245.

Nithya, V., Regan, R., & Vijayaraghavan, J. (2013). A survey on SQL injection attacks, their detection and prevention techniques. *Int. J. Eng. Comput. Sci* 2(4): 886-905.

Sukhdeve, S. D., & Channe, H. (2016). The Code Sanitizer: Regular Expression Based Prevention of Content Injection Attacks. *International Journal of Computer Trends and Technology (IJCTT)* 35(1): 21-28.

Stuart, M. (2017). SQL Injection [Data set]. GitHub. https://github.com/stu17682/sql-injection-filter/tree/master/dataset