Preventing SQL Injection Attack Based on Machine Learning

Eun Hong Cheon, Zhongyue Huang, Yon Sik Lee

Department of Computer Engineering, Woosuk University

Corresponding Author: ehcheon@woosuk.ac.kr, hitigon@gmail.com

Department of Computer Information Engineering, Kunsan National University

First Author: yslee@kunsan.ac.kr

Abstract

In this paper, we present the most critical security risk of vulnerable web applications, SQL injection attack. We design a system based on machine learning for preventing SQL injection attack, which utilizes pattern classifiers to detect injection attacks and protect web applications. The system captures parameters of HTTP requests, and converts them into numeric attributes. Numeric attributes include the length of parameters and the number of keywords of parameters. Using these attributes, the system classifies the parameters by Bayesian classifier for judging whether parameters are injection patterns. If any SQL injection pattern is found, the TCP connection between the attacker and server will be terminated immediately. As a learning-based method, it is necessary to have a training phase before the detection and prevention. We also present a tool that generates massive injection and legitimate patterns automatically by randomization and combination. We evaluated this method with various different types of injection patterns, and evaluated the actual effect with a popular SQL injection attack tool named Sqlmap. The results of evaluation show that proposed system was able to prevent SQL injection attack with a simple mechanism and high positive detection rate.

Keywords: SQL Injection Attack, Machine Learning, Web Security, Web Vulnerability

1. Introduction

With the enormous increase of quantity and quality, web applications are changing the living and work habits of people. Web applications are usually designed to interact with back-end databases and number of web applications were attacked due to web vulnerabilities [1]. According to OWASP's top Ten 2010, which lists the top ten vulnerabilities in web applications, the first security risk is SQL injection [2]. SQL injection attack (SQLIA) generates malicious queries in a web application that change the developer's intended functions when an attacker specifies crafted input. Using these inappropriate inputs, the attacker could retrieve important and sensitive data from databases, such as passwords of administrators or customers' personal information. If these data are published to the public, it may cause a large number of damage to these web application owners and users. SQLIA is usually caused by input contents from clients. Although we can use some methods to escape potentially harmful characters in client request messages, there may be a number of conditions that need to be concerned [3]. For instance, if an application considers some special symbols such as single-quotes are legal in name strings but are illegal in password strings, thus the application can’t just simply forbid these symbols from all input contents and need a more complicated method to refine it soundly.

In this paper, we design a system based on machine learning for preventing SQLIA attack. This system captures HTTP requests to obtain input contents and classifies them by Bayesian classifier, and then detects malicious contents and terminates attacks. In addition, we created a tool for generating training samples automatically through classifying and analyzing legitimate and injection patterns in the real world. We evaluated this learning-based method by various different types of injection patterns, and verified the actual effect with a SQL injection attack tool. Unlike previous approaches, our method is effective with a simple detection mechanism and independent of databases and web applications, in other hands, any web application with any database can be protected by the method well and need not be modified anything.

This paper makes the following contributions. In Section 2, we review the previous works of SQLIA. Section 3 presents the design of system based on Machine Learning for preventing SQL injection attack. Section 4 describes the experiments and the evaluation results, and Section 5 draws conclusions and future work.

2. Related Work

There are a mass of approaches to deal with SQLIA, such as static analysis, dynamic monitoring, parsing and so on. Each approach has its advantages, but also disadvantages. Some of detection
methods may be bypassed through particular characters and some could be affected by codes or values. Halfond et al. [4] had classified many techniques related to SQLI. We classify SQLI detection techniques based on it.

2.1. Dynamic Analysis

Sqlmap [5] is an open source automatic SQL tool. With multiple injection techniques, this tool detects and exploits SQL injection flaws with pre-defined attack codes. It compares contents of HTTP responses to determine if a SQL injection is successful. It can fetch data from the database and access the underlying file system and execute the commands on the operating system. We used this tool in the experiments for verifying the effectiveness of our method.

2.2. Dynamic query matching

Das et al. [6] give us a detection method for the SQL injection based on dynamic query matching. This approach is independent of the syntactical rules. It converts SQL queries into an XML form by an automatic parsing and then compares this XML file with another XML file which consists of the legitimate distinct SQL queries by a user defined threshold. An issue of this method is that the empirical threshold value would affect the accuracy of results.

2.3. Combined static and dynamic analysis

AMNESIA detects SQL injection attack with a model-based approach that combines the static and runtime monitoring [7][8]. In the static phase, it uses a static analysis to build a model of the legitimate SQL queries that could be generated by applications. In the dynamic phase, it checks generated queries against the static model. This method depends on the static phase, but certain types of obfuscation codes make it less precise and more false positives.

Saner [9] combines static and dynamic analysis techniques to identify faulty sanitation procedures that can be bypassed by an attacker. With the static analysis technique, Saner characterizes the sanitization process by modeling the way in which an application processes input values. The dynamic analysis technique is able to reconstruct the code that is responsible for the sanitization of application input.

2.4. Machine learning approach

Valeur et al. [10] proposed the use of an intrusion detection system based on a machine learning technique. This method identifies queries that do not match multiple models of typical queries at runtime, including string model and data type-independent model. It is trained by a set of typical application queries, and the quality depends on the quality of the training set.

We presented a new method to prevent SQLI attack based on machine learning [11]. This method identifies SQL injection codes by HTTP parameters’ attributes and the Bayesian classifier. This technique depends on the choices of patterns’ attributes and the quality of the training set. We choose two values as attributes of patterns, and invent a way to generate the real-world patterns automatically. Unlike some dynamic approaches such as parsing, it handles input strings with simple and quick processes. It is also independent because it supports all of databases and web applications. Unlike the static approaches, the source codes of web applications need not to be modified or used for creating basic models.

3. The design of system

We have developed a SQL injection detection system that utilizes pattern classifiers to detect injection attacks and protect the web applications and back-end systems. In this section, we describe the details about this detection system.
3.1. The Architecture of System

The detection module is in front of the web server, and it captures HTTP requests from clients and reviews these requests to prevent any malicious SQL query. Figure 1 shows the architecture of the detection system. Monitor is responsible for capturing and decoding the parameters of HTTP requests from clients. After capturing parameters of each HTTP request, these parameters are converted into numeric attributes by Converter. Classifier estimates the class of each parameter and makes further appropriate processes.

![Figure 1. The architecture of detection system](image)

3.2. Monitor

Monitor captures each request that sends to a specified port of the server, and the default port is 80. There are two kinds of request methods, GET and POST, which are usually used in web interactions. These requests can contain every parameter requested by clients. After capturing HTTP requests, Monitor decodes characters that have been encoded by URL encoding in each parameter and turns each letter into upper case. This step is not only to unify symbols and letters but also to prevent attackers from bypassing the detection method through URL encoding and mixed case.

3.3. Converter

Converter is used to convert HTTP parameters into numeric attributes. These attributes are supplied for Classifier as pattern attributes. We choose the length of parameters and the number of keywords of parameters as pattern attributes.

The keywords contain words and symbols in SQL statements, like commas, equal signs, quotation marks, “SELECT”, “UNION” and so on. There are three types of keywords according to SQL statements. The first type must be a blank space on the right side of the word when it is used in any SQL statement, like “SELECT ”; the second one, each side of the word must have a blank space, like “UNION”; and the last, such as commas, need not any blank space aside. Figure 2 shows an example of Converter, the length of this injection code is 24 and the number of keywords is 10.

![Figure 2. An example of Converter](image)
3.4. Classifier

Classifier is to determine each pattern's class with Bayesian classifier [12][13]. Bayesian classifier is based on Bayes’ decision theory. It uses Bayes’ equation to calculate the probability of each class of an object, and then choose the one that has larger posterior probability as the object class. The principle of Bayesian classifier is given in Figure 3.

In this figure, \(X\) is a collection of patterns. If there are \(n\) patterns, \(X\) is showed as \(X = \{(X_1, t_1), (X_2, t_2), (X_3, t_3), \ldots (X_n, t_n)\}\). \(X_i\) is the collection of each pattern's attributes, and showed as \(X_i = (x_1, x_2, \ldots)\). \(t_i\) is the corresponding class name of \(X_i\). Assume that there are \(m\) classes, and \(w\) means class names, so \(t_i \in \{w_1, w_2, \ldots, w_m\}\). In practice, \(X_i\) has two attributes and \(m\) has two classes.

![Figure 3. The principle of Bayesian classifier](image)

The attributes of each sample are served as the parameters of discriminate function \(g(X)\). Discriminate functions calculate the probabilities based on Bayes’ equation, which can be express as \(P(X|w_i)P(w_i)\). Through comparing the results of discriminate functions, Classifier calculates the maximum value, and further gets the most possible class of a pattern and calls the corresponding processes.

The attributes of each sample are served as the parameters of discriminate function \(g(X)\). Discriminate functions calculate the probabilities based on Bayes’ equation, which can be express as \(P(X|w_i)P(w_i)\). Through comparing the results of discriminate functions, Classifier calculates the maximum value, and further gets the most possible class of a pattern and calls the corresponding processes.

The algorithm of Bayesian classifier is presented below:

```python
input: X
output: None
[1] begin
[2] Get \(g_1(X) = P(X | w_1)P(w_1)\);
[3] Get \(g_2(X) = P(X | w_2)P(w_2)\);
[4] Get the class index \(k\) of max value between \(g_1(X)\) and \(g_2(X)\);
[5] if \((w_k == \text{Injection Pattern})\)
[7] Write the pattern into injection log;
[8] Execute ‘Block process’;
[9] else
[10] Declare ‘Legitimate String’;
[11] Write the string into legal log;
[12] Execute ‘Pass process’
[13] end
```

In this algorithm, Classifier calculates two probabilities of a pattern by discriminate functions, including the probability of the injection pattern and the probability of the legitimate string, and then gets the most probable class of this pattern. Classifier executes next processes according to this result. If the pattern is a SQL injection code, it will sound a warning and write this pattern into the injection log file and execute the block process to terminate the connection. If the pattern is a legitimate string, it will write this string into the legal log and let it pass through.

3.5. Training and test

As the fundamental and crucial steps of machine learning, the training phase and test phase are performed prior to detection. The training phase processes the pre-defined training samples to learn
what the injection and legitimate patterns look like. It obtains the classifier-related values and stores the values into Classifier. The test phase processes test samples by Classifier. It calculates the positive rate of test samples for checking the effectiveness of the method. For these procedures, we generated two sample files, one file is named “training.samples” for the training process, and another is named “test.samples” for the test process.

We developed a script program to generate controllable legitimate and injection patterns. The legitimate patterns are based on real-world web applications. The program simulates numbers, user names, passwords, email addresses and so on. They are generated randomly, and some of them contain one or more keywords of SQL statements. For example, “pGIWAzQ@mGvrN.com” is a legal string for simulating an Email address.

The injection patterns are more complex. A SQL injection attack may include several attack techniques. According to the features of SQLIA templates in J. Clarke’s book [14] and some technical reports [15], the program simulates multiple injection techniques and generates different injection patterns by randomization and combination. Randomization means that templates could replace strings, numbers with random strings and numbers. And combination means that the script combines different injection patterns or keywords together. For example, in the blind SQL injection test, one of templates is ‘1) or (1=1’, the program replaces the number ‘1’ with random integers, and changes the number of parentheses if the whole statement conforms to the SQL syntax. As a result, the generated pattern may be ‘15))) or (((1878=1878’. And then the script converts each pattern into numeric attributes and stores these data.

4. Experiments and evaluation

We implemented a version of the detection module in C++. The components of this module were described in the previous sections. In addition, we developed a sample-generator in Python that generated a training sample file with 2,142 lines of SQL injection patterns and legitimate strings, and a test sample file with 4,070 lines of patterns.

The experiments had four phases. Training and test were the first and second phases. The third phase used various different SQL injection patterns to attack the web application. The results of these phases are showed in Table 1. The final phase was to use a popular SQL injection attack tool named Sqlmap for verifying the actual effect of the system in the real environment. The details of the third and fourth phases are described in next sections.

4.1. Attacks

The attack patterns in the third phase are mixed with several kinds of common SQLIA techniques. The injection patterns for attacking are generated by the script program and tested in the test phase of the module.

4.1.1. Attack 1

Attack 1 contains two kinds of injection techniques and total 30 patterns in the attack file. The stacked injection combines multiple SQL statements to execute some SQL codes on the back-end database. For example, it is a SELECT * FROM users WHERE user='admin'. It does not only query data from the database like the example, but also can manipulate the database by using update or drop SQL statements. The time-based injection is a kind of blind injection for inferring the information of tables and databases. Using the delay time generator like sleep() and benchmark() in MySQL, attackers can infer the details about the database. In the example, the statement makes the MySQL database to run hash function sha1() for a million times.

4.1.2. Attack 2

Attack 2 contains two kinds of injection techniques and total 160 patterns in the attack file. This technique is not only for injection but also can be used to identify the number of columns in a query. The ‘order by’ statement is useful because it is always hard to be found in the database logs and the attackers can get the results quickly by using binary search. As the same as ‘order by’ statement, this union-based injection technique is also used for finding the number of columns in a query. But it is more complex and powerful than ‘order by’ statement. Because when an attacker identifies the number of columns, he or she can enumerate some data from tables, like passwords of administrators, or even can load files into the file system.
4.1.3. Attack 3

Attack 3 contains two kinds of injection techniques and combines these techniques together to simulate the complexity of SQL injection attack and total 1,940 patterns in the attack file. The inline injection is a basic way to test and confirm the vulnerabilities in a web application. By a logical sequence, an attacker can estimate whether the web application is injectable. The fingerprint injection can retrieve the information of the database, including the database version, because there are different features between different database management systems and even different versions in the same DBMS. An attacker cannot inject a web application without knowing the ‘fingerprint’ of the back-end database. There are many methods to get the fingerprint of MySQL, and we used parts of them and combined the inline injection to generate different and random patterns.

These attacks are common and typical, and always used by automatic attack tools. The results of training, test and attacks are presented in Table 1.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Patterns</th>
<th>Correct Detection</th>
<th>Positive rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>2,142</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Test</td>
<td>4,070</td>
<td>4,054</td>
<td>99.61%</td>
</tr>
<tr>
<td>Attack 1</td>
<td>30</td>
<td>30</td>
<td>100%</td>
</tr>
<tr>
<td>Attack 2</td>
<td>160</td>
<td>160</td>
<td>100%</td>
</tr>
<tr>
<td>Attack 3</td>
<td>1,940</td>
<td>1,932</td>
<td>99.59%</td>
</tr>
</tbody>
</table>

4.2. Evaluation with Sqlmap

Sqlmap supports almost all of SQL injection techniques and databases. For evaluating the effectiveness of our system in the real environment, we used injection techniques in section 4.1 that also supported by Sqlmap to attack the web application. The attack procedures are simple by controlling the commands of Sqlmap, such as "python sqlmap.py -u "http://192.168.0.1/sqli/index.php?id=1" --dbs -v 3". A counter inside the module was set up to record the numbers of total patterns and detected patterns. And then we compared the records in the Sqlmap’s output file with detection results. Figure 4 shows the procedure of an attack by Sqlmap.

![Figure 4. The procedure of an attack by Sqlmap](image)

The results of these experiments are showed in Table 2. The column in the table, ‘Injection method’, means the attack methods used in this test, and most of them have been described in the section 4.1; ‘Patterns’ is the number of injection and legitimate patterns used by Sqlmap; ‘Correct Detection’ is the number of patterns detected correctly by the module. From these data and positive rates, it is proved that our method can prevent the majority of SQL injection attacks, and most of attack patterns can be detected correctly, but some inline or blind injection codes, which are used for testing SQLI spots, would be missed or ignored.
Table 2. The results of Sqlmap attacks

<table>
<thead>
<tr>
<th>Injection method</th>
<th>Patterns</th>
<th>Correct Detection</th>
<th>Positive rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stacked injection</td>
<td>51</td>
<td>51</td>
<td>100%</td>
</tr>
<tr>
<td>Time-based injection</td>
<td>51</td>
<td>51</td>
<td>100%</td>
</tr>
<tr>
<td>Union-based injection</td>
<td>5</td>
<td>5</td>
<td>100%</td>
</tr>
<tr>
<td>Enumerate</td>
<td>239</td>
<td>239</td>
<td>100%</td>
</tr>
<tr>
<td>Fingerprint</td>
<td>321</td>
<td>321</td>
<td>100%</td>
</tr>
<tr>
<td>Inline injection</td>
<td>23</td>
<td>17</td>
<td>73.91%</td>
</tr>
</tbody>
</table>

5. Conclusion

This paper presents a detection system for preventing SQL injection attack against web applications based on machine learning. This system can capture HTTP requests from clients and converts the parameters of HTTP requests into numeric attributes. Using these attributes, it can classify the parameters by Bayesian classifier to detect SQL injection attacks and further to terminate these attacks. This approach relies on numeric attributes and training samples. We use the length of parameters and the number of keywords of parameters as numeric attributes, and generate a mass of training samples by randomization and combination for simulating SQL injection patterns and legitimate strings. This system has a simple mechanism and can be used for different web applications with different databases. Through some experiments and attacking by a SQL injection attack tool, the results of study show that the system was able to detect the majority of SQL injection techniques and was effective for preventing SQL injection attack.

Our future work will be focus on improving the method of generating samples and the algorithm of classifier. The better training samples and algorithm would raise the positive rate of the module and correct potential problem, such as the low detection rate of ‘Inline injection’. We can compare different machine learning methods with Bayesian classifier to get the balance between the detection rate and efficiency. The security of log files also should be concerned. These files should be encrypted with hash functions and set access privileges. We also will make more experiments with other databases and attack tools, and test the overhead of operating system.

6. Acknowledgement

This research was supported by the Woosuk University Research Fund in 2013 and the Business for Cooperative R&D between Industry, Academy, and Research Institute funded Korea Small and Medium Business Administration in 2012(Grants No. C0028654)

7. References