IMPLEMENTATION OF SMART FILTER TO AVOID SQL INJECTIONS WITH SIGNATURE BASED INTRUSION DETECTION

Reshma Rai
Department of CSE
RKDF Institute of Science & Technology, Bhopal
E-mail: reshma.rai7@gmail.com

Jitendra Jadhav
Assistant Professor, Department of CSE
RKDF Institute of Science & Technology, Bhopal

ABSTRACT
In this paper, we present a detailed discussion on different SQL injection attacks and their prevention technique. In addition, we proposed a new technique for prevention of SQL injection attack for web application. To address the problems of SQL injections, we proposed a technique that uses a concept of filter called “Smart Filter”, that avoid the SQL injections with static matching and dynamic signature based intrusion detection mechanism with MS SQL database web application. This smart filter actually works in between the web application & database server. Therefore, before sending SQL queries to the database, the smart filter will analyse the query to check the vulnerability. If found any, it reported else it forwards the query to database server. Apart from the checking the SQL query by smart filter, it also reports the new vulnerabilities found in SQL queries. The proposed scheme is efficient and overhead is negligible. Therefore, we strongly suggest the proper use of anti-vulnerable techniques in the e-commerce website development.

Keywords:
SQL Injections, Web Security, Database Security, Vulnerabilities, Database Intrusion

1. INTRODUCTION
In recent years, a large number of software systems are being ported towards the Web, and platforms providing new kinds of services over the Internet are becoming more and more popular: e-health, e-commerce, e-government. At the same time, however, such Web applications are subject to attacks by hackers, with the objective of gaining unauthorized access to the system, accessing to private information, or simply causing a denial of service. A very common vulnerability for Web application is SQL Injection. As we can see in following figure 1.

In SQL injection, attacker provides user inputs those results in the different database requests. This means; user input in SQL statements in the different form then original intended. In short, this attack is happened when user input is parsed as SQL tokens then were changing the semantics of underlying query. The aim of SQL injection is to query the database a manner that was not the intent of the application programmer.

Figure 1: Break down of website vulnerabilities [24]

There are several techniques used in SQL injection. Most of them use SQL statement in different SQL injection techniques. Increased dependence on web applications significantly and use in the activities of our daily lives grow in the number and level of attacks that target them.

With SQL injections, cyber-criminals can take complete remote control of the database, with the consequence that they can become able to manipulate the database to do anything they wish, including:
- Insert a command to get access to all account details in a system, including user names and retrieve VNC passwords from registry.
- Shut down a database.
- Upload files.
• Through reverse lookup, gather IP addresses and attack those computers with an injection attack.
• Corrupting, deleting or changing files and interact with the OS, reading and writing files.
• Online shoplifting e.g. changing the price of a product or service, so that the cost is negligible or free.
• Insert a bogus name and credit card in to a system to scam it at a later date.
• Delete the database and its all contents.

91% of database attacks lead to financial loss [1], but the financial impact can be dwarfed be the long-term damage to an organizations reputation. In fact, research by Ipsos MORI[1] revealed that 58% of consumers would stop using an organization’s services following a security breach involving their personal data.

SQL injections attack techniques are the specific means by which a threat agent carries out attacks using malicious code. Threat agents may use many different methods to achieve their goals, often combining several of these sequentially or employing them in different varieties.

**Tautology:** The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The consequences of this attack depend on how the results of the query are used within the application. The most common usages are to bypass authentication pages and extract data. In this type of injection, an attacker exploits an inject-able field that is used in a query’s WHERE condition. Transforming the conditional into a tautology causes all of the rows in the database table targeted by the query to be returned. In general, for a tautology-based attack to work, an attacker must consider not only the inject-able/vulnerable parameters, but also the coding constructs that evaluate the query results. Typically, the attack is successful when the code either displays all of the returned records or performs some action if at least one record is returned.

**End of Line Comment:** After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments. An example would be to add [ - - ] after inputs so that remaining queries are not treated as executable code, but comments. This is useful since threat agents may not always know the syntax or fields in the server. Like, an attacker submits [admin'--] for the user name input field (the input submitted for the other fields is irrelevant). The resulting query is:

```
SELECT * FROM user WHERE username = 'admin'-- AND password = ''
```

The code injected in the condition [admin'--] transform the WHERE clause in that way it is going to log the attacker as admin user if there is a user name called “admin”, because rest of the SQL query will be ignored.

**Illegal/Logically Incorrect Query:** This attack lets an attacker gather important information about the type and structure of the back-end database of a Web application. The attack is considered a preliminary, information gathering step for other attack. The vulnerability leveraged by this attack is that the default error page returned by application servers is often overly descriptive. In fact, the simple fact that error messages is generated can often reveal vulnerable/inject-able parameters to an attacker. Additional error information, originally intended to help programmers debug their applications, further helps attackers gain information about the schema of the back-end database. When performing this attack, an attacker tries to inject statements that cause a syntax, type conversion, or logical error into the database. Syntax errors can be used to identify inject-able parameters. Type errors can be used to deduce the data types of certain columns or to extract data. Logic errors often reveal the names of the tables and columns that caused the error.

**Union Query:** In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer. Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query>. Because the attackers completely control the second/injected query, they can use that query to retrieve information from a specified table. The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query.

Example: Referring to the running example, an attacker could inject the text [' UNUnion SELECT username, password from user_info where user_name=’abc’--'] into the login field, which produces the following query:

```
SELECT * FROM usersaccounts WHERE account='' UNION SELECT password from user_info where user_name=’abc’--AND pass=''```

Assuming that there is no login equal to ”, the original first query returns the null set, whereas the second query returns data from the “user_info” table. In this case, the database would return column “password” for username “abc”. The database takes the results of these two queries, unions them, and returns them to the application. In many
applications, the effect of this operation is that the value for "password" is displayed along with the user information.

**Piggy-backed Query:** In this attack type, an attacker tries to inject additional queries into the original query. We distinguish this type from others because, in this case, attackers are not trying to modify the original intended query; instead, they are trying to include new and distinct queries that "piggy-back" on the original query. As a result, the database receives multiple SQL queries. The first is the intended query which is executed as normal; the subsequent ones are injected queries, which are executed in addition to the first. This type of attack can be extremely harmful. If successful, attackers can insert virtually any type of SQL command, including stored procedures, into the additional queries and have them executed along with the original query. Vulnerability to this type of attack is often dependent on having a database configuration that allows multiple statements to be contained in a single string. For example, If the attacker inputs ["; drop table users - -] into the password field, the application generates the query:

```
SELECT accounts FROM users WHERE login='reshma' AND pass=''; drop table usersaccounts --
```

After completing the first query, the database would recognize the query delimiter (";") and execute the injected second query. The result of executing the second query would be to drop table users, which would likely destroy valuable information. Other types of queries could insert new users into the database or execute stored procedures. Many databases do not require a special character to separate distinct queries, so simply scanning for a query separator is not an effective way to prevent this type of attack.

**System Stored Procedure:** SQLIAs of this try to execute stored procedures present in the database. Today, most database vendors ship databases with a standard set of stored procedures that extend the functionality of the database and allow for interaction with the operating system. Therefore, once an attacker determines which backend database is in use, SQLIAs can be crafted to execute stored procedures provided by that specific database, including procedures that interact with the operating system.

2. LITERATURE REVIEW

After studying many researches on the SQL injection detection and prevention, it is found that not a single technique is strong enough to handle all the problems and vulnerabilities to websites due to SQL injections. Some of the techniques proposed the changes at the development stages. As in [16], the proposed technique works for defending the SQL injections against the stored procedure. Similarly, various researches provides defending techniques against SQL injection.

In [1] SQL injection is one most important web security threat that needs attention so as to improve security for the users and their data. This paper deals with an application specific randomized encryption algorithm to detect and prevent it further its effectiveness was compared with other existing techniques and its performance was quantified. Hence we took up this web security vulnerability and analysed its attack types. The security threat posed SQLIA is really high and it is very necessary to protect users’ data in a web application, since it is very confidential and sensitive.

The mechanism to keep track of the positive taints and negative taints is proposed by William G.J Halfond, Alessandro Orso, Panagiotis Manolios [17]. Defensive Programming [11] [5] has given an approach for Programmers by which they can implement their own input filters or use the existing safe API so that prevent malicious input or that convert malicious input into a safer input. Vulnerability pattern approach is used by Livshits and Lam, they propose a static analysis approach for finding the SQL injection attack. The main issues with this method, is that it cannot detect the SQL injection attacks patterns that are not known beforehand. Vulnerability patterns are described here in this approach.

AMNESIA mechanisms to prevent SQL injection at run time is proposed by William G.J. Halfond and Alessandro Orso [9]. It uses a model based approach to detect illegal queries before it is sent for execution to the database. Static analysis framework (called SAFEI) has been proposed by Xiang Fu et al [12], for identifying SIA (SQL Injection attacks) vulnerabilities at compile time.. The source code information can be analysed by SAFEI and will be able to identify very delicate vulnerabilities that cannot be discovered by black-box vulnerability scanners.

Scott and Sharp Proxy filter [23] [21], this technique can be effective against SQLIA; they used a proxy to filter input data and output data streams for a web application, although correctly specify filtering rules for each application is required by the developers to input.

The mechanism which filters the SQL Injection in a static manner is proposed by Buehrer et al [18]. The SQL statements by comparing the parse tree of a SQL statement before and after input and only
allowing to SQL statements to execute if the parse
trees match.

Novel-specification based methodology proposed by
Konstantinos et al [10], they are given a mechanism
to detect SQL injection with. This approach utilizes
specifications that define the intended syntactic
structure of SQL queries that are produced and
executed by the web-application. Instruction-Set
Randomization [23] [19] defined a framework that
allows developers to create SQL queries using
randomized keywords instead of the normal SQL
keywords.

Marco Cova et al [11], proposed a Swaddler which,
analyses the internal state of a web application and
learns the relationships between the application's
critical execution points and the application's
internal state.

NTAGW ABIRA Lambert and KANG Song Lin [5]
propose a string tokenizer which, creates tokens of
original query and SQL-injected query, and creates
an array of tokens of both the original and injected
query, if the length of arrays of both query is found
equal, that means no SQL-injection. Otherwise
there is injection.

ROMIL RAWAT and SHAILENDRA KUMAR
SHRIVASTAVA [2] In this paper we will use SVM
(Support Vector Machine) for classification and
prediction of SQL-Injection attack. In our propose
algorithm, SQL-Injection attack detection accuracy is
(96.47%) and which is the highest among the
existing SQL-Injection detection Techniques.

The system proposed by MEHDI KIANI, ANDREW
CLARK AND GEORGE MOHAY [8] uses an anomaly
based approach which utilizes the character
distribution of certain sections of HTTP requests to
detect previously unseen SQL injection attacks. The
advantage of the system proposed by Mehdi Kiani
et.al is that it does not require any user interaction,
or no modification of, or access to the backend
database or the source code of the web application.
The problem faced is the high rate of false alerts
which had to be taken care while implementing the
system in real time environment. This is because of
less information available on attacks to the
administrator, thus making it difficult to differentiate
between false alerts and the real attacks.

V. SHANMUGHANNEETHI, C. EMILIN SHYNI
AND DR.S.SWAMYNATHAN uses a methodology
[6] which make use of an independent web service
which is intended to generalize the syntactic
structure of the SQL query and validate user inputs.
The SQL query inputs submitted by the user are
parsed through an independent service and the
correctness of the syntactic structure of the query are
checked. The main advantage of this paper is that the
error message generated doesn’t contain any Meta
data information about the database which could
help the attacker. Since the web service is not
integrated with the web application, any
modification that should be done to the system
should be done in such a way that it should be
supported by the web service.

R. EZUMALAI, G. AGHILA, proposed [7] a
combinatorial approach for shielding web
applications against SQL injection attacks. This
combined approach incorporates signature based
method, used to address security problems related to
input validation and auditing based method which
analyse the transactions to find out the malicious
access. This approach requires no modification of
the runtime system, and imposes a low execution
overhead. It can be inferred from this approach that
the public interface exposed by an application
becomes the only source of attack.

YUJI KOSUGA, KENJI KONO, MIYUKI
HANAOKA, et.al proposed [13] a technique called
Sania for detecting SQL injection vulnerabilities
during the development and debugging phases of a
web application. It identifies the vulnerable spots by
analysing the SQL queries issued in response to the
HTTP requests in which an attacker can insert
arbitrary strings. The main feature of Sania is the
generation of attacks using the knowledge of this
model, thus checking if the SQL injection
vulnerabilities lie in the web application.

KE WEI, M. MUTHUPRASANNA, SURAJ
KOTHARI in [16] proposed a technique to defend
attacks against the stored procedures. This technique
combines a static application code analysis with a
runtime validation to eliminate injection attacks. In
the static part, a stored procedure parser is designed,
and for any SQL statement that depends on user
inputs, and use this parser to instrument the
necessary statements in order to compare the
original SQL statement structure to that including
user input. The underlying idea of this technique is
that any SQLIA will alter the structure of the
original SQL statement and by detecting the
difference in the structures, a SQLIA can be
identified.

KAI-XIANG ZHANG, CHIA-JUN LIN, et.al
proposed a translation and validation (TransSQL)
based approach for detecting and preventing SQL
Injection attacks [4]. The basic idea of this approach
relies on how different databases interpret SQL
queries and those SQL queries with an injection.
After detailed analysis on how different databases
interpret SQL queries, Kai-Xiang Zhang, et.al
proposed an effective solution TransSQL, using
which the SQL requests are executed in two
different databases to evaluate the responses generated.

3. PROPOSED WORK
Wassermann and Su propose an approach that uses static analysis combined with automated reasoning to verify that the SQL queries generated in the application layer cannot contain a tautology [11][12]. The primary drawback of this technique is that its scope is limited to detecting and preventing tautologies and cannot detect other types of attacks. To handle the SQL injection types vulnerabilities from the websites, we need a strong technique that ensures us to avoid the execution of malicious code in website scripts.

In the proposed technique, we handle the detection and prevention of the SQL injections, which is one of the major threat related with websites today. The proposed technique uses “SMART FILTER” to detect and record the SQL Injections. For this functioning, we will implement our “smart filter” in three modules:
- Injection Parser Module
- Signature Based Detection Module
- Threat Recorder Module

Algorithm:
It is known to all that, once the user entered the data in the server control, it first goes to the sever end. Then,
1- We first checked the inputted text using the static method called “Static method” Parser. This checks the string by using the well-known parsing method called “Recursive Descent Parsing”.
   A. If parser return TRUE. (String Parse successfully) Goto step 3.
   B. If parser return FALSE. (String Parse Un-successful) Goto step 2.
2- Now, we will apply the adaptive method for detection called “Signature based detection”. This method uses the database to check the existing signatures to compare with.

A. If the signature found, Generate “error” message, call recorder and quit.
B. If the signature not found, Goto step 3.
3- Make transaction with database, the inputted data doesn’t contain any vulnerable character.
4- Call event recorder, to record every event in a log file.
5- exit

3.1 SMART FILTER
The “Smart filter” works as the interface between website (web Pages) and database server. Therefore, to detect SQL Injections the smart filter must check every user query or inputs and properly analyze all of them to report the SQL INJECTION. The key idea is to save time and make an efficient tool to handle the problems of SQL injections.

We just need to embed the smart filter in database layer of any website. With the help of proposed technique, we will able to detect the following types of SQL Injections as listed in table 1.

Table 1 : Different types of SQL injections

<table>
<thead>
<tr>
<th>Type of Attack</th>
<th>Attack Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tautologies</td>
<td>Authentication Identifying Injectable Parameters</td>
</tr>
<tr>
<td></td>
<td>Extracting Data</td>
</tr>
<tr>
<td>Logically Incorrect</td>
<td>Identifying Injectable Parameters</td>
</tr>
<tr>
<td>Queries</td>
<td>Performing Database Finger-Printing</td>
</tr>
<tr>
<td></td>
<td>Extracting Data</td>
</tr>
<tr>
<td>Union Query</td>
<td>Bypassing Authentication</td>
</tr>
<tr>
<td></td>
<td>Extracting Data</td>
</tr>
<tr>
<td>PiggyBacked Queries</td>
<td>Executing Data</td>
</tr>
<tr>
<td></td>
<td>Adding or Modifying Data</td>
</tr>
<tr>
<td></td>
<td>Performing Denial of Service</td>
</tr>
<tr>
<td></td>
<td>Executing Remote Commands</td>
</tr>
<tr>
<td>Stored Procedures</td>
<td>Performing Privilege Escalation</td>
</tr>
<tr>
<td></td>
<td>Performing Denial of Service</td>
</tr>
<tr>
<td></td>
<td>Executing Remote Commands</td>
</tr>
</tbody>
</table>
### 3.1 Injection Parser Module:
This module works like a static checker in the Smart Filter for the SQL injection detection. In this module, we are using a recursive descent parser based on CFG, the CFG is designed here as per the requisite for the safe queries. Therefore, this recursive descent parser ensures the administrator that the query does not contain any vulnerable character. Therefore, it protects the system from injected query; the only problem with this module is that it cannot fully protect the system. Therefore, we use this concept as a module in our proposed methodology.

#### 3.1.1 Injection Parser Module:
- Blind Injection
  - Identifying Injectable Parameters
  - Extracting Data
  - Determining Database Schema

- Timing Attacks
  - Identifying Injectable Parameters
  - Extracting Data
  - Determining Database Schema

- Alternate Encodings
  - Evading detection

#### 3.1.2 Signature Based Detection Module:
This module is the core part of the proposed technique as it provides a robust technique to detect and prevent the SQL injections, this module works when:
1. Query may have special characters.
2. Injected query cannot detect by the Injection Parser Module.

This module can upgrade the knowledge using supervised-learning, the administrator can update the knowledge of the system periodically. This will result the accuracy in detection of malicious entries.

#### 3.1.3 Threat Recorder Module:
This module is developed for the auditing purpose, as it generates the reports that help the administrator to identify the errors, choose the signatures to upgrade the system knowledge. The point of a threat recorder module is to keep track of what is happening with the server. If something should malfunction within a system, there may be no other way of identifying the problem.

When a problem does occur, it is easy to pinpoint and fix. This module and log file recording are also important to keeping track of applications that have little to no human interaction, such as server applications. Therefore, we do so. We can also use this module for the learning purpose, as it contains all the malicious code and signatures.

4. RESULT ANALYSIS
The complete implementation of the proposed work carried out using the Microsoft .Net framework 4.0 with Visual Studio 2008, SQL Server 2008, IIS Server and a vulnerability scanner tool. Through which we scan the demonstrated developed website for any vulnerability threat and SQL attacks. We also demonstrate the whole project by a web application developed by VB.Net & ASP.Net along with the database in SQL Server 2008. The demonstrated website works in two modes at every instance. Whether its login or search, inserting or updating, etc.

Smart Protected Mode:- In this mode the website works under the guidance of Smart Filter, which sanitizes the inputs and always looking for any intrusion or malicious code in the input. The development of smart filter leads to a mechanism that helps to decide whether the input data is legitimate and does not contain any malicious code. It also contains a supervised-learning mechanism that help the system for the further improvement. Therefore, the more knowledge, patterns inputted in the system, the more accurate results will be obtained.

At server side, the proposed smart filter analyses the input data and sends to the database stored procedure, where execution of SQL commands performs and the desired data retrieved. At data access layer, the website received the data from the internal web pages, and then it uses the sanitization process to make decision on input data before sending to the database program.

The most important feature of our proposed plan for detecting & preventing the SQL injections is that it never overhead the data in the database. It can be applied to any existing web model without performing any alteration in database schema (if considering a separate database schema for smart filter.).

The table 2 shown below are the results of the different testing sessions on the proposed implemented system. It shows the actual number of requests made to the web application, out of
which, the proposed system successfully detects the injections.

Table 2: Result obtained by using the vulnerability scanner

<table>
<thead>
<tr>
<th>TOTAL DATABASE REQUESTS</th>
<th>INJECTED DATA</th>
<th>VALID DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2373</td>
<td>1497</td>
<td>876</td>
</tr>
<tr>
<td>5013</td>
<td>3452</td>
<td>1561</td>
</tr>
<tr>
<td>3588</td>
<td>2453</td>
<td>1135</td>
</tr>
<tr>
<td>1993</td>
<td>1209</td>
<td>784</td>
</tr>
</tbody>
</table>

Table is represented as:

<table>
<thead>
<tr>
<th>Test</th>
<th>Database Requests</th>
<th>Injected Data</th>
<th>Valid Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>2373</td>
<td>1497</td>
<td>876</td>
</tr>
<tr>
<td>Test2</td>
<td>5013</td>
<td>3452</td>
<td>1561</td>
</tr>
<tr>
<td>Test3</td>
<td>3588</td>
<td>2453</td>
<td>1135</td>
</tr>
<tr>
<td>Test4</td>
<td>1993</td>
<td>1209</td>
<td>784</td>
</tr>
</tbody>
</table>

While working with demonstrating website, by using the vulnerability scanners and concurrent users, we have successfully generated the above number of requests in four different iterations. The proposed implemented system gives the appropriate results by analyzing the inputs provided to them and finally output the counts of the Valid & Injected queries.

Advantages and Limitations
The proposed methodology for evaluation of security & protection tool against SQL injection attacks called “SMART FILTER”, we presents several important points of interest related to the SQL injection prevention and avoidance. First of all the methodology is abstracted enough to be utilized for the evaluation of different types of tools, not only the proposed method. In fact, it provides a standard and common guideline for the evaluation process of detection and prevention of SQL injection attack tools in general without any restriction or limitations. In proposed technique all the phases are needed for the complete evaluation procedure, and must be adapted and stetted up for the specific tool you are testing. This is the biggest weakness, but it is compulsory in order to keep a large level of abstraction. For example in our case, ASP.Net vulnerable web applications and MS SQL Server. However proposed Smart Filter is database and operating system independent, so for them we were free of choice. Another important advantage is that, our proposed methodology provides a complete evaluation by analysing different aspects of the tool. The only limitation with the proposed research is that, it is language dependent; one has to migrate the logic to other language.

5. CONCLUSION
Finally, after analysing the existing techniques, we propose a supervised-learning methodology for the SQL injections detection & preventions. Which consist of three important modules called Parser, Smart Filter & recordkeeping module. One of our goals in this paper was to increase the level of security awareness among organizations regarding web applications, especially towards SQL injection threats. We hope that further surveys in this area and in related web application subjects will help achieving that goal, so that hopefully security standards will be implemented and countermeasures built into applications during development. Ultimately, organizations will use a proactive approach towards application layer security, which will then be an indispensable part of web applications.

6. REFERENCES


