A SQL Injection : Internal Investigation of Injection, Detection and Prevention of SQL Injection Attacks

Abhay K.Kolhe
Faculty, Dept. Of Computer Engineering
MPSTME, NMIMS
Mumbai, India

Pratik Adhikari
Student, Dept. Of Computer Engineering
MPSTME, NMIMS
Mumbai, India

Abstract

SQL Injection has been always as the top threat in any web site and web application. In this paper we are making a dummy web site and injecting some SQL queries, detecting the SQL injection using the IP tracking method, preventing SQL injection using different types of defense mechanism. We have made the dummy website to inject, detect and prevent the SQL injection attacks. We are also giving the internal view where it is required to explain these attacks, the detection and defense mechanism through the explanation of the source codes.

Keywords: SQL injection, SQL injection vulnerability, web security.

1. Introduction

The SQL injection is the code injection technique, in which malicious SQL statements are inserted into an entry field for execution and used to attack the database and performs different types of the database interaction, operations and functions without sanitizing the inputs in the entry field [1]. SQL injection is always a top priority for security of any web applications as injection can be done in any website or web application made in any language like PHP, JSP, ASP with the database like Mysql, Oracle, Microsoft SQL server [2] [3].

2. SQL injection for Authentication, Insert, drop, Update and delete.

We have made the simple website where the user can register, login and get authenticated. We make the web vulnerable as we didn't use any protection mechanism in any form. In the below codes in **2.1** we are explaining the simple mechanism for extraction of login information for the registered user through the MySql database.

2.1. Extracting login information

```
    //adding values
mySQL_select_db("pratik", $con);
    $name1 = mySQL_query("select * from user where
username='$username' and password = '$password'");
    $row1 = mySQL_fetch_array($name1);
    if($row1['username'] == '')
    {
    echo 'Wrong username/password!';
    echo '</br><a href="loginform.html">Retry</a>';
    }
    else
    {
        $_SESSION['logedinuser']=$row1['username'];
            echo 'Login successful';
            echo'</br></br>Hello,
'.$row1['username'].'    &nbs
p; <a href="logout.php">Logout</a>';
            echo '</br>';
            echo '<h1>Welcome to our site!!</h1>';
```

Steps:
1) Select the database.
2) Select Username and password and store in variable name1.
3) From the variable name1 fetch the values to variable row1.
4) If variable row1 is empty or doesn't match return "wrong username and password".
5) Else make login successful when values match with the database.

6) Return “user name” and “welcome to our site message

2.2 SQL injection for authentication

We are able to get a login into our website successfully by giving username: anything' OR 'x'='x and password: anything' OR 'x'='x as the following shown in the **Figure 2.1.1(a).**This type of SQL injections is for incorrectly filtered escape character. Later in the prevention phase we will discuss how it can be prevented. We have made the table name as the “user” where the values of all registered user are stored.

Injection query can be explained as:

"SELECT * FROM user WHERE name = '" + userName + "';"

If the "userName" is replaced by SQL string, like anything' or 'x'='x during authentication, the database responds to the code in same manner as the first code and displays the records. This is because evaluation of 'x'='x' is always true. So we get authenticated without any proper valid user name and password in the database. Here in this case single quote always allows the inside quote to get executed [4].

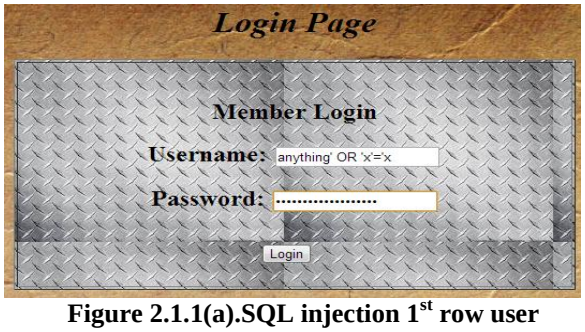

Figure 2.1.1(a).SQL injection 1st row user

We can see in the **Figure 2.1.1(b)** that we are able to login to the website without any username and password. In this case the row for the first user is always getting selected for the login on the website. This form of the SQL injection is easy to check on any website .Most of the coders and developers forget to filter the escape character and this attack is always vulnerable in PHP as well as asp.net for both MySQL and MS SQL server 2005, 2008.

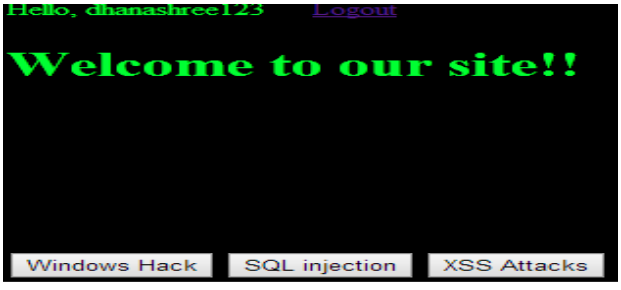

Figure 2.1.1(b).Results for injection 1st row user

2.1.2 Injection for particular user such as admin

For the case, such as admin here we use username as “admin” we can login as x' OR username LIKE '%admin%. We supply both query to our login details such as username: x' OR username LIKE '%admin% and Password: x' OR username LIKE '%admin%.

When user enters a “LIKE” clause with the username then the database will return the matching criteria to the user immediately. SELECT username, password, email, full_name FROM user WHERE username=‘x‘OR full_name LIKE _%admin%‘; Here, the database will return information of any user where the name starts with “admin”. **Figure 2.1.2(a)** shows the injection for the particular user such as “admin” and **Figure 2.1.2(b)** shows the login results.“%” is the wild character used to select particular user from the database.

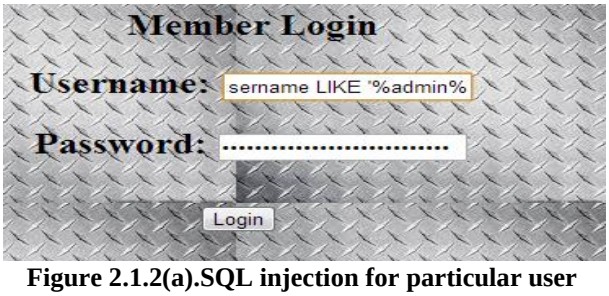

Figure 2.1.2(a).SQL injection for particular user

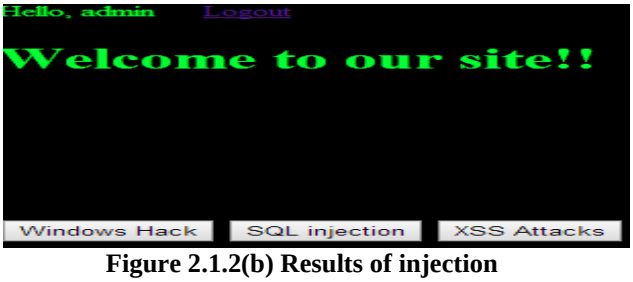

Figure 2.1.2(b) Results of injection

For the case, such as admin here we use the username as “admin” we can login as x' OR username LIKE '%admin%. We supply both query to our login details such as username: x' OR username LIKE '%admin% and Password: x' OR username LIKE '%admin%.

When a user enters a “LIKE” clause with the username then the database will return the matching criteria to the user immediately. SELECT username, password, email, full_name FROM user WHERE username=‘x‘OR full_name LIKE _%admin%‘; Here, the database will return information of any user where the name starts with “admin”. **Figure 2.1.2(a)** shows the injection for the particular user such as “admin” and **Figure 2.1.2(b)** shows the login results.“%” is the wild character used to select a particular user from the database**.**

2.1.3 Injection for insert operation.

Here we are injecting the insert query to the database as username, password, email, and the name for the particular user. **Figure 2.1.3(a) shows** the injection for insert query.

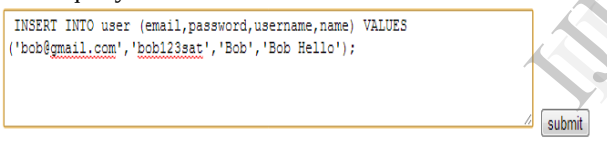

Figure 2.1.3(a) SQL Injection for Insert Query

Table 2.1.3(b) shows the insert operation results. We can see the email, password, username and name is added in the table name user.

Table 2.1.3(b) Insert operation

Id	Username	Password	Email	Name
33	dhanashree123	dhan123	dhanashree@gmail.com	dhanashree abcd
36	admin	9920999801	admin@gmail.com	admin test
37	naresh	naresh123	naresh@gmail.com	Naresh Stambamkabi
38	Bob	bob123sat	bob@gmail.com	Bob Hello

Inserted new row

2.1.4 Injection for update

Here we are injecting the update query to the database we are updating the username for the particular email address. **Figure 2.1.4(a)** shows the injection for the update query.

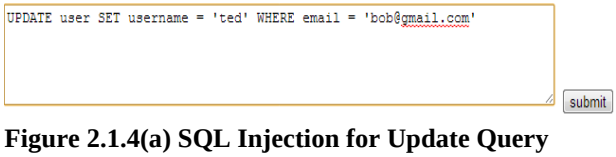

Figure 2.1.4(a) SQL Injection for Update Query

Table 2.1.4(b) shows the update results we can see results clearly that the username for the id 38 have been updated from “bob” to “ted”.

Table 2.1.4(b) update operation

Id	Username	Password	Email	Name
33	dhanashree123	dhan123	dhanashree@gmail.com	dhanashree abcd
36	admin	9920999801	admin@gmail.com	admin test
37	naresh	naresh123	naresh@gmail.com	Naresh Stambamkabi
38	ted	bob123sat	bob@gmail.com	Bob Hello

Update operation done

2.4 Injection for delete

Here we are injecting the delete query to the database we are deleting the particular row in the database based on the username. **Figure 2.1.5(a)** shows injection for the delete query.

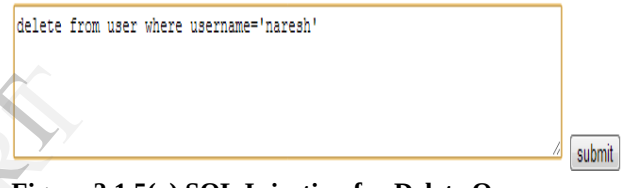

Figure 2.1.5(a) SQL Injection for Delete Query

Table 2.1.5(b) shows the delete results .we can see clearly that the username “naresh” with id “37” has been deleted successfully.

Table 2.1.5(b) Delete operation

Id	Username	Password	Email	Name
33	dhanashree123	dhan123	dhanashree@gmail.com	dhanashree abcd
36	admin	9920999801	admin@gmail.com	admin test
38	ted	bob123sat	bob@gmail.com	Bob Hello

Delete operations done for user id 37 “naresh”

2.5 Injection for Drop

To show the Drop operation in our website we are making additional table for the drop operation.

```
CREATE TABLE IF NOT EXISTS `student` (
  `fname` varchar(100) NOT NULL,
  `lname` varchar(100) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
--
-- Dumping data for table `student`
--
INSERT INTO `student` (`fname`, `lname`) VALUES
('Pratik', 'Adhikari'),
('Nikhil', 'Kamath'),
```

('cc', 'dd'),
('Pratik', 'www.Hackingweb.com');

To inject the drop query we have made extra table named "student" for the first name and last name so our main table "user" doesn't get altered. **Table 2.1.6(a)** shows the table name "student".

Table 2.1.6(a) Student Table

First name	Last Name
Pratik	Adhikari
Nikhil	Kamath
Jayesh	Kabra

Here we are injecting the drop query. We are dropping the table name "student". **Figure 2.1.6(b)** show the injection for the drop query.

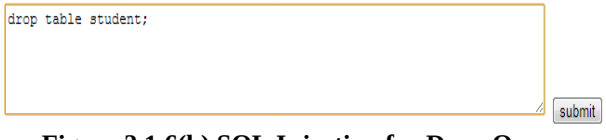

Figure 2.1.6(b) SQL Injection for Drop Query

Figure 2.1.6(c) shows the results after dropping the table.

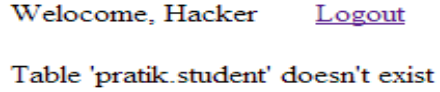

Figure 2.1.6(c) Results for dropped table.

3. Detection based on the IP Tracking mechanism

We are taking the array which takes the injection by the keywords and we match the same keyword using preg_match function. After that we track the clients/users IP address and insert the injection type as well as IP address in the database table called iptracker[5].

```
//Logic for finding out injecion
$injection['inject']=array("LIKE","--
","insert","update","delete","drop","'=");
$querytofire=$query;
$regex=implode("|",$injection['inject']);
if(preg_match("/(\b{$regex}\b)/i",$querytofire,$matche
s)){
$ip=$_SERVER['REMOTE_ADDR'];
$injection_type=$matches[0];
$ipupdate="insert into iptracker
(ip_addr,injection_type) values
('$ip','$injection_type')";
$ipinsert=mysql_query($ipupdate);
}
```

3.1 Database for IP detection

Here we have created the table name "iptracker".

```
CREATE TABLE IF NOT EXISTS `iptracker` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `ip_addr` varchar(20) NOT NULL,
  `injection_type` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
AUTO_INCREMENT=19 ;
--
-- Dumping data for table `iptracker`
--
INSERT INTO `iptracker` (`id`, `ip_addr`,
`injection_type`) VALUES
(1, '127.0.0.1', ''),(2, '127.0.0.1', ''),(3, '127.0.0.1', ''),(4,
'127.0.0.1', ''),(5, '127.0.0.1', ''),(6, '127.0.0.1', ''),(7,
'127.0.0.1', ''),(8, '127.0.0.1', 'INSERT'),(9, '127.0.0.1',
'UPDATE'),(10, '127.0.0.1', 'delete'),
(11, '127.0.0.1', 'drop'),(12, '127.0.0.1', 'or'),(13,
'127.0.0.1', 'or'),(14, '127.0.0.1', '1=1'),(15, '127.0.0.1',
'or'),(16,'127.0.0.1', 'or'),(17, '127.0.0.1', 'INSERT'),(18,
'127.0.0.1', 'drop');
```

In PHP we extract the IP address using $_SERVER['REMOTE_ADDRS'] which is the server variable and added to the database.

4. Prevention technique for SQL injection

4.1 mySQL_real_escape_string()

Escapes special characters in the unescaped_string, taking into account the current character set of the connection so that it is safe to place it in a mySQL_query()[6].

mySQL_real_escape_string() calls MySQL's library function mySQL_real_escape_string, which adds backslashes to the following characters: \x00, \n, \r, \, ', " and \x1a.We are filtering out the query before it is

passed to the database we can see the that we are unable to inject the query after using this mechanism[6].

```
if(isset($_POST['submit'])&&
$_POST['prevention_param_escape']==1){
$query=mysql_real_escape_string($_POST['query']);
mysql_connect("localhost", "root", "") or
die(mysql_error());
mysql_select_db("pratik") or die(mysql_error());
$result=mysql_query($query);
if (!$result) {
   echo 'Access Unauthorised';
}
$message='Access Unauthorised';
header("Location:sqlinjection.php?message=$message"
);
}
```

Here in the code we are using **mySQL_real_escape_string()** for the prevention mechanism , in the above code first we are checking prevention method is set or not by using isset function and prevention_param_escape parameter. Then by using the function "mysql_real_escape_string" we check is it injected or not. If it's injected the message access unauthorized is generated.

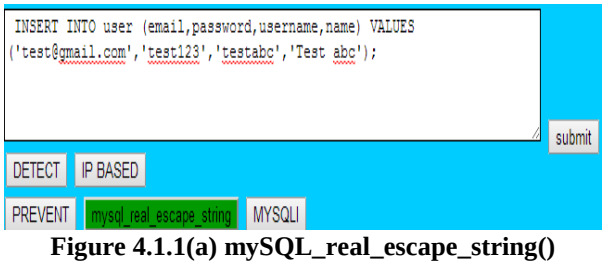

Figure 4.1.1(a) mySQL_real_escape_string()

4.1.2 MySQLi

MySQLi optionally allows having multiple statements in one statement string [7].

Multiple statements or multi queries must be executed with mySQLi_multi_query(). The individual statements of the statement string are separated by semicolon. Then; all result sets returned by the executed statements must be fetched. The MySQL server allows having statements that do return result sets and statements that do not return result sets in one multiple statements [7].

An extra API call is used for multiple statements to reduce the likeliness of SQL injection attacks [7].

```
if(isset($_POST['submit'])&&$_POST['prevention_par
am']==0&&
$_POST['prevention_param_escape']==0){
$query=$_POST['query'];
$mysqli = new mysqli("localhost", "root", "", "pratik");
if (!$mysqli->multi_query($query)) {
echo "Query failed: (" . $mysqli->errno . ") " . $mysqli-
>error;
}
```

Here in the code we are first checking the prevention method is set or not using isset function and 'prevention_param' parameter. Then by using the function "mysqli" we check is it injected or not and if it is injected then we will generate message access unauthorized.

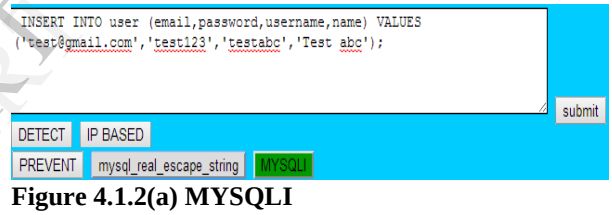

Figure 4.1.2(a) MYSQLI

Results after **mySQL_real_escape_string()** and **MYSQLI**

Table 4.1.1(b) Using **MYSQLI** and **mySQL_real_escape_string()**

Id	Username	Password	Email	Name
33	dhanashree123	dhan123	dhanashree@gmail.com	dhanashree abcd
36	admin	9920999801	admin@gmail.com	admin test
38	ted	bob123sat	bob@gmail.com	Bob Hello

Access Unauthorised

5. Conclusion and further work:

In this paper we have shown different types of injection of SQL attacks like authentication, insert, delete, drop and update operations. We also have show the detection of these attacks by using IP-tracking method, where we store the IP address of the user for the particular type of attacks that they are injecting.

We also have shown the internal coding for the injection, detection as well as prevention for these types of attacks.

In future we would like to explore more on SQL injection attacks and analyse these attacks based on the various parameters.

6. References

[1] http://technet.microsoft.com/enus/library/ms161953(v=SQL.105).aspx 25th Nov 2013.

[2] https://www.owasp.org/index.php/Top_10_2013-Top_10 25th Nov 2013.

[3] Zeinab Raveshi, Sonali R.Idate "Investigation and Analysis of SQL Injection Attacks on Web Applications: Survey" International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-2, Issue-3 February 2013.

[4] Ramakanth Dorai,Vinod Kannan "SQL Injection-Database Attack Revolution and Prevention" Journal of International Commercial Law and Technology Vol. 6, Issue 4 (2011).

[5] Perumalsamy Ramasamy, Dr.SunithaAbburu, "SQL INJECTION ATTACK DETECTION AND PREVENTION" International Journal of Science and Technology (IJEST) ISSN: 0975-5462 Vol. 4 No.04 April 2012.

[6] http://php.net/manual/en/function.mySQL-real-escape-string.php 26 Nov 2013.

[7] http://www.php.net/manual/en/mySQLi.quickstart.multiple-statement.php 26 Nov 2013.

[8] Sid Ansari,Edward R. Sykes ,"SQL Injection in Oracle: An exploration of vulnerabilities", International Journal on Computer Science and Engineering (IJCSE) ISSN :0975-3397 Vol. 4 No. 04 April 2012 522.