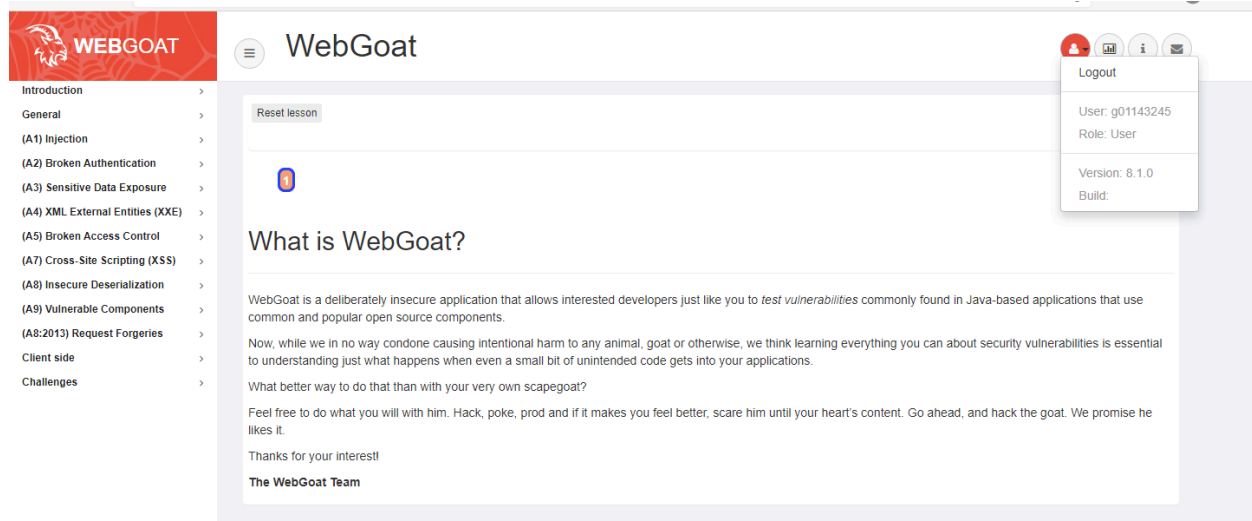


Roozah Khan
Lab #1

a. Screenshot of WebGoat running with your user id



b. Screenshot of Burp Suite running with interception on.

⚡ Burp Project Intruder Repeater Window Help Burp Suite Community Edition v2021.8.2 - Temporary Project

Dashboard Target **Proxy** Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn

Intercept HTTP history WebSockets history Options

✎ Request to http://localhost:8080 [127.0.0.1]

Forward Drop **Intercept is on** Action Open Browser Comment this

Pretty **Raw** Hex \n ≡

```
1 POST /WebGoat/login HTTP/1.1
2 Host: localhost:8080
3 Content-Length: 36
4 Cache-Control: max-age=0
5 sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="92"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://localhost:8080
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36
11 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://localhost:8080/WebGoat/login
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Cookie: JSESSIONID=VS0U2RU8sdFuY843v4rY_K90D6L80y_iMp2i0una
20 Connection: close
21
22 username=g01143245&password=roozahl8
```

c. Screenshot of Lesson 11 results, and the SQL statement you used.

[Show hints](#) [Reset lesson](#)

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

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 to read sensitive data like credit card numbers from a database.

What is String SQL injection?

If queries are built dynamically in the application by concatenating strings to it, this makes it very susceptible to String SQL injection.

If the input takes a string that gets inserted into a query as a string parameter, then you can easily manipulate the build query using quotation marks to form the string to your specific needs. 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 - like 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 earning employee, you want to exploit the system and 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
32147	Paulina	Travers	Accounting	46000	P45JSI
34477	Abraham	Holman	Development	50000	UU2ALK
37648	John	Smith	Marketing	64350	3SL99A
89762	Tobi	Barnett	Development	77000	TA9LL1
96134	Bob	Franco	Marketing	83700	LO9S2V

d. Screenshot of Lesson 12 results, and the SQL statement you used.



Introduction >
General >
(A1) Injection >
SQL Injection (intro) >
SQL Injection (advanced) >
SQL Injection (mitigation) >
Path traversal >
(A2) Broken Authentication >
(A3) Sensitive Data Exposure >
(A4) XML External Entities (XXE) >
(A5) Broken Access Control >
(A7) Cross-Site Scripting (XSS) >
(A8) Insecure Deserialization >
(A9) Vulnerable Components >
(A8-2013) Request Forgeries >
Client side >
Challenges >

SQL Injection (intro)

Show hintsReset lesson

12345678910111213

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.

The integrity of any data can be compromised, if an attacker per example changes information that he should not even be able to access.

What is SQL query chaining?

Query chaining is exactly what it sounds like. When 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 which marks the end of a query and that way allows to start another one right after it within the same 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:

Get department

Logout

User: g01143245
Role: User

Version: 8.1.0
Build:



Introduction >
General >
(A1) Injection >
SQL Injection (intro) >
SQL Injection (advanced) >
SQL Injection (mitigation) >
Path traversal >
(A2) Broken Authentication >
(A3) Sensitive Data Exposure >
(A4) XML External Entities (XXE) >
(A5) Broken Access Control >
(A7) Cross-Site Scripting (XSS) >
(A8) Insecure Deserialization >
(A9) Vulnerable Components >
(A8-2013) Request Forgeries >
Client side >
Challenges >

SQL Injection (intro)

Show hintsReset lesson

12345678910111213

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.

The integrity of any data can be compromised, if an attacker per example changes information that he should not even be able to access.

What is SQL query chaining?

Query chaining is exactly what it sounds like. When 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 which marks the end of a query and that way allows to start another one right after it within the same 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:

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
37648	John	Smith	Marketing	1000000	3SL99A
96134	Bob	Franco	Marketing	83700	LO9S2V
89762	Tobi	Barnett	Development	77000	TASLL1
34477	Abraham	Holman	Development	50000	UU2ALK
32147	Paulina	Travers	Accounting	46000	P4SJSI

Logout

User: g01143245
Role: User

Version: 8.1.0
Build:

e. Screenshot of Lesson 13 results, and the SQL statement you used.

WEBGOAT

Introduction

General

(A1) Injection

SQL Injection (intro)

SQL Injection (advanced)

SQL Injection (mitigation)

Path traversal

(A2) Broken Authentication

(A3) Sensitive Data Exposure

(A4) XML External Entities (XXE)

(A5) Broken Access Control

(A7) Cross-Site Scripting (XSS)

(A8) Insecure Deserialization

(A8) Vulnerable Components

(A8-2013) Request Forgeries

Client side

Challenges

SQL Injection (intro)

Show hintsReset lesson

12345678910111213

Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we now are going to compromise the third element of the CIA triad: **availability**.

There are many different ways to violate availability. If an account is deleted or the password gets changed, the actual owner cannot access it anymore. Attackers could also try to delete parts of the database making it useless or even dropping the whole database. Another way to compromise availability would be to per example revoke access-rights from admins or any other users, so that nobody gets access to (specific parts of) the database.

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:

Search logs

Logout

User: g01143245

Role: User

Version: 8.1.0

Build:

WEBGOAT

Introduction

General

(A1) Injection

SQL Injection (intro)

SQL Injection (advanced)

SQL Injection (mitigation)

Path traversal

(A2) Broken Authentication

(A3) Sensitive Data Exposure

(A4) XML External Entities (XXE)

(A5) Broken Access Control

(A7) Cross-Site Scripting (XSS)

(A8) Insecure Deserialization

(A8) Vulnerable Components

(A8-2013) Request Forgeries

Client side

Challenges

SQL Injection (intro)

Show hintsReset lesson

12345678910111213

Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we now are going to compromise the third element of the CIA triad: **availability**.

There are many different ways to violate availability. If an account is deleted or the password gets changed, the actual owner cannot access it anymore. Attackers could also try to delete parts of the database making it useless or even dropping the whole database. Another way to compromise availability would be to per example revoke access-rights from admins or any other users, so that nobody gets access to (specific parts of) the database.

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:

Search logs

Logout

User: g01143245

Role: User

Version: 8.1.0

Build:

WEBGOAT

Introduction

General

(A1) Injection

SQL Injection (intro)

SQL Injection (advanced)

SQL Injection (mitigation)

Path traversal

(A2) Broken Authentication

(A3) Sensitive Data Exposure

(A4) XML External Entities (XXE)

(A5) Broken Access Control

(A7) Cross-Site Scripting (XSS)

(A8) Insecure Deserialization

(A8) Vulnerable Components

(A8-2013) Request Forgeries

Client side

Challenges

SQL Injection (intro)

Show hintsReset lesson

12345678910111213

Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we now are going to compromise the third element of the CIA triad: **availability**.

There are many different ways to violate availability. If an account is deleted or the password gets changed, the actual owner cannot access it anymore. Attackers could also try to delete parts of the database making it useless or even dropping the whole database. Another way to compromise availability would be to per example revoke access-rights from admins or any other users, so that nobody gets access to (specific parts of) the database.

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:

Search logs

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

Logout

User: g01143245

Role: User

Version: 8.1.0

Build: