



# Auditoria Web Goat

De

Daniel Shved



## OBJETIVOS:

Simulación de auditoria de web goat.

## HERRAMIENTAS USADAS:

- WAPPALYZER: Tecnologia y lenguajes de programación utilizados en la aplicación web
- NMAP: Sistema Operativo y puertos abiertos
- OWASP ZAP: Interceptar paquetes para resolver ejercicios.
- MODO DESARROLLADOR de Firefox: Ver como esta escrita la pagina para resolver ejercicios.

## INFORMACIÓN GATHERING:

El servidor en el que corre webgoat utiliza Linux como sistema operativo y tiene abiertos los puertos tcp 8080, 8081 y 9090. Hemos sacado esta información utilizando el programa NMAP

```
(kali㉿kali)-[~]
└─$ sudo nmap -O 127.0.0.1
[sudo] contraseña para kali:
Starting Nmap 7.92 ( https://nmap.org ) at 2022-07-10 15:25 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000069s latency).
Not shown: 997 closed tcp ports (reset)
PORT      STATE SERVICE
8080/tcp  open  http-proxy
8081/tcp  open  blackice-icecap
9090/tcp  open  zeus-admin
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6.32
OS details: Linux 2.6.32
Network Distance: 0 hops

OS detection performed. Please report any incorrect results at https://nmap.org/submit/.
Nmap done: 1 IP address (1 host up) scanned in 1.66 seconds

(kali㉿kali)-[~]
└─$
```

Utiliza los javascript frameworks: Backbone.js 1.4.0, RequireJS 2.3.6 .

Librerías javascript: jQuery 2.1.3. , jQuery UI 1.10.4, Underscore.js .

Font Scripts: Font Awesome

Ui frameworks: animate.css y Bootstrap

Lenguaje de Programacion: Java

The screenshot shows the Wappalyzer web application interface. At the top is a purple header with the Wappalyzer logo and navigation icons. Below the header are two tabs: 'TECHNOLOGIES' (active) and 'MORE INFO'. An 'Export' button is located in the top right. The main content area is divided into two columns. The left column lists technologies under four categories: 'JavaScript frameworks' (Backbone.js 1.4.0, RequireJS 2.3.6), 'Font scripts' (Font Awesome), and 'Programming languages' (Java). The right column lists technologies under two categories: 'JavaScript libraries' (jQuery 2.1.4, jQuery UI 1.10.4, Underscore.js) and 'UI frameworks' (animate.css, Bootstrap). At the bottom, there is a section titled 'Enrich your data with tech stacks' with a link to upload a list of websites for analysis.

**Wappalyzer**

TECHNOLOGIES MORE INFO Export

**JavaScript frameworks**

- Backbone.js 1.4.0
- RequireJS 2.3.6

**Font scripts**

- Font Awesome

**Programming languages**

- Java

[Something wrong or missing?](#)

**JavaScript libraries**

- jQuery 2.1.4
- jQuery UI 1.10.4
- Underscore.js

**UI frameworks**

- animate.css
- Bootstrap

**Enrich your data with tech stacks**

[Upload a list of websites to get a report of the technologies in](#)

## FALLOS DISPONIBLES EN LA SECCIÓN A1 SQL INJECTION:

Tras aprender SQL básico <http://www.sqlcourse.com/> utilizando el siguiente formato se puede sacar información del servidor usando el comando select:

The **select** statement is used to query the database and retrieve selected data that match the criteria that you specify. Here is the format of a simple select statement:

```
select "column1"  
[,"column2",etc]  
from "tablename"  
[where "condition"];  
[] = optional
```

### Resolucion del ejercicio A1(2):

Employees Table

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	LO9S2V
34477	Abraham	Holman	Development	\$50.000	UU2ALK
37648	John	Smith	Marketing	\$64.350	3SL99A

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 query

SQL query

Submit

You have succeeded!

SELECT department FROM Employees WHERE userid = 96134

DEPARTMENT

Marketing

Usando este formato a base de prueba y error se puede sobre escribir información en SQL:

The **update** statement is used to update or change records that match a specified criteria. This is accomplished by carefully constructing a where clause.

```
update "tablename"
set "columnname" =
    "newvalue"
[, "nextcolumn" =
    "newvalue2"...]
where "columnname"
    OPERATOR "value"
[and|or "column"
    OPERATOR "value"];

[] = optional
```

## Resolucion ejercicio A1(3)

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

### 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

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

Usando el comando ALTER TABLE se puede alterar la base de datos.  
Varchar(20) define la longitud del campo añadido a la tabla.

Resolucion ejercicio A1(9)



## Data Definition Language (DDL)

Data definition language includes commands for defining data structures. DDL commands are commonly used to define a database's schema. The schema refers to the overall structure or organization of the database and, in SQL databases, includes objects such as tables, indexes, views, relationships, triggers, and more.

If an attacker successfully "injects" DDL type SQL commands into a database, he can violate the integrity (using ALTER and DROP statements) and availability (using DROP statements) of a system.

- DDL commands are used for creating, modifying, and dropping the structure of database objects.
- CREATE - create database objects such as tables and views
- ALTER - alters the structure of the existing database
- DROP - delete objects from the database
- Example:
  - CREATE TABLE employees(  
    userid varchar(6) not null primary key,  
    first\_name varchar(20),  
    last\_name varchar(20),  
    department varchar(20),  
    salary varchar(10),  
    auth\_tan varchar(6)  
);
  - This statement creates the employees example table given on page 2.

Now try to modify the schema by adding the column "phone" (varchar(20)) to the table "employees". :

SQL  
query

Submit

**Sorry the solution is not correct, please try again.**

object name already exists in statement [ALTER TABLE employees ADD phone varchar(20)]

Usando el comando GRANT se le puede dar permisos de administrador a usuario:

Resolucion problema A1(5):

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

➕

1

2

3

4

5

6

7

8

9

10

11

12

13

➔

## Data Control Language (DCL)

Data control language is used to implement access control logic in a database. DCL can be used to revoke and grant user privileges on database objects such as tables, views, and functions.

If an attacker successfully "injects" DCL type SQL commands into a database, he can violate the confidentiality (using GRANT commands) and availability (using REVOKE commands) of a system. For example, the attacker could grant himself admin privileges on the database or revoke the privileges of the true administrator.

- DCL commands are used to implement access control on database objects.
- GRANT - give a user access privileges on database objects
- REVOKE - withdraw user privileges that were previously given using GRANT

Try to grant rights to the table `grant_rights` to user `unauthorized_user`:

✓

**SQL query**

grant select on grant\_rights to unauthorized\_user

[Submit](#)

**Congratulations. You have successfully completed the assignment.**



## A1 SQL INJECTION - APARTADO 10

El código lo que dice es selecciona todo de la tabla “user\_data” con los datos login count y User\_ID, excepto que si ponemos 1 or 1 = 1 (Booleano True) elimina la comprobación del Login\_Count y el User\_ID pues solo una de las condiciones de siente que cumplir; o cuadra el Login\_Count y el User\_Id con la info del servidor o 1 = 1.

Para aclarar es irrelevante lo que pongamos en Login\_Count por el “or 1 = 1 ” que es true.

12345678910111213

### Try It! Numeric SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = " + User_ID;
```

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

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

✓

Login\_Count: 100

User\_Id: 1 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	338434535333	AMEX	,	0
15837	Chaos	Monkey	32849386533	CM	,	0
19204	Mr	Goat	33812953533	VISA	,	0

Your query was: SELECT \* From user\_data WHERE Login\_Count = 100 and userid= 1 or 1 = 1

8

## A1 SQL INJECTION - APARTADO 11

Usamos SQL Injeccion para sacar datos de otros usuarios. De nuevo lo que pongamos en Authentication TAN es irrelevante al igual que antes lo era el Login\_Count. En este caso es por que el código Smith” or 1 = 1 – hace que el Authentication TAN sea texto.

Se cumple 1 = 1 ergo te doy todos los datos que cuadren con todo.

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; 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: Smith' or 1 = 1 --

Authentication TAN: 3SL99A

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

## A5 INSECURE DIRECT OBJECT REFERENCES - APARTADO 3

Filtrando los paquetes con owasp zap podemos ver que la respues del servidor mas de lo que presenta la pagina:

The screenshot shows the WebGoat application interface. The browser address bar displays `https://127.0.0.1:8080/WebGoat/start.mvc#lesson/IDOR.lesson/2`. The page title is "Insecure Direct Object References". A sidebar on the left lists various lessons, with "Insecure Direct Object References" highlighted. An "HTTP Message" window is open, showing the following content:

```
Content-Type: application/json
Date: Fri, 08 Jul 2022 17:11:51 GMT

{
  "role" : 3,
  "color" : "yellow",
  "size" : "small",
  "name" : "Tom Cat",
  "userId" : "2342384"
}
```

Below the HTTP message, there are three buttons: "Active Scan", "Replay in Console", and "Replay in Browser".

### Insecure Direct Object References

The screenshot shows the lesson content for "Insecure Direct Object References". It includes a "Show hints" button and a "Reset lesson" button. A progress bar shows 6 steps, with step 3 highlighted. The lesson title is "Observing Differences & Behaviors". The text explains that a consistent principle from the offensive side of AppSec is to view differences from the raw response to what is visible. It states that there is often data in the raw response that doesn't show up on the screen/page. Below this, there is a "View Profile" button and a text area showing the following profile data:

```
name:Tom Cat
color:yellow
size:small
```

Below the profile data, there is a checkbox and a text input field. The text input field contains the text "role, userId". A "Submit Diff" button is next to the text input field. Below the text input field, there is a message: "Correct, the two attributes not displayed are userId & role. Keep those in mind".

## A5 INSECURE DIRECT OBJECT REFERENCES - APARTADO 4

Usando el formato de la petición (que nos da el IDOR/profile) y los datos del ejercicio anterior (Numero de id) podemos a base de prueba y error sacar :

### Insecure Direct Object References



Show hints Reset lesson

➡ 1 2 3 4 5 6 ➡

#### Guessing & Predicting Patterns

##### View Your Own Profile Another Way

The application we are working with seems to follow a RESTful pattern so far as the profile goes. Many apps have roles in which an elevated user may access content of another. In that case, just /profile won't work since the own user's session/authentication data won't tell us whose profile they want view. So, what do you think is a likely pattern to view your own profile explicitly using a direct object reference?



Please input the alternate path to the Url to view your own profile. Please start with 'WebGoat' (i.e. disregard 'http://localhost:8080/')

WebGoat/IDOR/profile/2342382

Submit

**Congratulations, you have used the alternate Url/route to view your own profile.**

{role=3, color=yellow, size=small, name=Tom Cat, userId=2342384}

## A5 INSECURE DIRECT OBJECT REFERENCES - APARTADO 5

Podemos ver información de otro usuario utilizando la dirección que obtuvimos anteriormente y a base de prueba y error el numero de ID. Creo recordar que cambie a Get en vez de un post también. Me queda por aprender como hacer fuzzing con owasp, el numero de ID lo saque probando por que en las pistas ponía que estaba cerca del mio. También la segunda parte del ejercicio fui incapaz de hacerla.

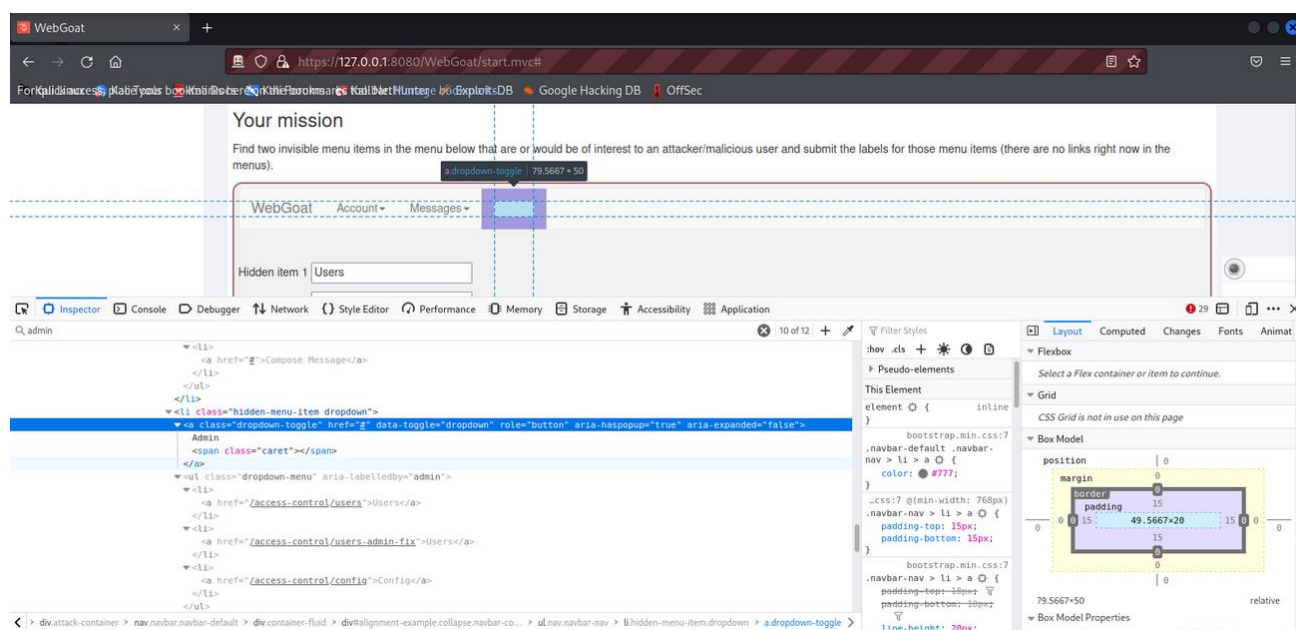
The screenshot shows the 'Insecure Direct Object References' lesson interface. On the left, a mobile app mockup displays a 'View Profile' button. On the right, an 'HTTP Message' window is open, showing a GET request to `http://127.0.0.1:8080/WebGoat/IDOR/profile/2342388`. The request headers include Host, User-Agent (Mozilla/5.0), Accept, Accept-Language, Content-Type, X-Requested-With, Connection, Referer, and Cookie. Below the message window are buttons for 'Active Scan', 'Replay in Console', and 'Replay in Browser'.

Respuesta con datos de otro Usuario.

The screenshot shows the same lesson interface, but the 'HTTP Message' window now displays a JSON response. The response contains fields for 'lessonCompleted', 'feedback', 'output' (which includes role, color, size, and name), 'userId', 'assignment', and 'attemptWasMade'. The buttons 'Active Scan', 'Replay in Console', and 'Replay in Browser' are still visible at the bottom.

## A5 MISSING FUNCTION LEVEL ACCESS CONTROL - APARTADO 2

En la programación de la pagina tras mucho mucho tiempo, usando búsquedas tipo admin, pass y hidden y utilizando la función de Firefox de señalarte en que parte del código esta a lo que apuntas con el cursor encontré dos elementos ocultos en la pagina, posiblemente para ser usados o que se muestran si entras como administrador:



1 2 3 4

### Relying on obscurity

One could rely on HTML, CSS, or javascript to hide links that users don't normally access. In the past, a network router tried to protect (hide) admin functionality with javascript in the UI: <https://www.wired.com/2009/10/routers-still-vulnerable>.

### Finding hidden items

There are usually hints to finding functionality the UI does not openly expose in:

- HTML or javascript comments
- Commented out elements
- Items hidden via CSS/classes

### Your mission

Find two invisible menu items in the menu below that are or would be of interest to an attacker/malicious user and submit the labels for those menu items (there are no links right now in the menus).

WebGoat Account Messages

✓

Hidden item 1

Hidden item 2

Submit

Correct! And not hard to find are they?!? One of these urls will be helpful in the next lab.

## A5 MISSING FUNCTION LEVEL ACCESS CONTROL - APARTADO 3

Modificando la petición anterior con Owasp Zap a Get (era un POST) y modificando el Acces Control y El Content Type puedes sacar el Hash de otros usuarios.

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

Modify the GET request to '/access-control/users' to include 'Content-Type: application/json'

1 2 3 4

### Try it

As the previous page described, sometimes applications rely on client-side controls to control access (obscurity). If you can find invisible items, try them and see what happens. Yes, it can be that simple!

#### Gathering User Info

Often data dumps originate from vulnerabilities such as SQL injection, but they can also come from poor or lacking access control.

It will likely take multiple steps and multiple attempts to get this one:

- Pay attention to the comments and leaked info.
- You'll need to do some guessing too.
- You may need to use another browser/account along the way.

Start with the information you already gathered (hidden menu items) to see if you can pull the list of users and then provide the 'hash' for Jerry's account.

Your Hash:

Submit

Sorry the solution is not correct, please try again.

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

Modify the GET request to '/access-control/users' to include 'Content-Type: application/json'

1 2 3 4

### Try it

As the previous page described, sometimes applications rely on client-side controls to control access (obscurity). If you can find invisible items, try them and see what happens. Yes, it can be that simple!

#### Gathering User Info

Often data dumps originate from vulnerabilities such as SQL injection, but they can also come from poor or lacking access control.

It will likely take multiple steps and multiple attempts to get this one:

- Pay attention to the comments and leaked info.
- You'll need to do some guessing too.
- You may need to use another browser/account along the way.

Start with the information you already gathered (hidden menu items) to see if you can pull the list of users and then provide the 'hash' for Jerry's account.

✓  
Your Hash:

Submit

Congrats! You really succeeded when you added the user.

Usando estos comandos:

While there is a simple well-known defense for this attack, there are still many instances on the web. Coverage of fixes also tends to be a problem in terms of fixing it. We will talk more about the defense in a little bit.

Especially as 'Rich Internet Applications' are more and more commonplace, privileged function calls linked to via JavaScript may be compromised. And if not adequately protected, sensitive data (such as your authentication cookies) can be stolen and used for someone else's purpose.

- From the JavaScript console in the developer tools of the browser (Chrome, Firefox)

- Any data field returned to the client is potentially injectable

Podemos averiguar que este campo es vulnerable:

An easy way to find out if a field is vulnerable to an XSS attack is to use the `alert()` or `console.log()` methods. Use one of them to find out which field is vulnerable.

15



## RESUMEN:

En resumen de cara a al ejercicio esta pagina es vulnerable a SQL Injection , tiene Insecure Direct Object References y es vulnerable a Cross Site Scripting.