SEGURIDAD WEB

FRONTEND

¿QUE ES?

La seguridad en el Frontend se refiere a las prácticas y medidas que se implementan en la interfaz de usuario de una aplicación web para proteger los datos, prevenir ataques y garantizar una experiencia segura para los usuarios.

DIFERENCIAS ENTRE FRONTEND Y BACKEND

El frontend debe enfocarse en proteger la interfaz de usuario y prevenir ataques del lado del cliente, mientras que el backend debe garantizar la seguridad de los datos, la autenticación y el acceso a los recursos.

El frontend es la primera línea de defensa, pero no puede confiar en el cliente
La validación en frontend mejora la experiencia de usuario, pero la validación final debe realizarse en el backend

La combinación de ambas estrategias es clave para una aplicación web segura.

PRINCIPALES AMENAZAS

- ***Cross-Site Scripting (XSS):*** Inyección de código malicioso en la aplicación a través de entradas del usuario.
- ***Cross-Site Request Forgery (CSRF):*** Ataques que obligan a un usuario autenticado a ejecutar acciones no deseadas.
- ***Exposición de datos sensibles:*** Evitar almacenar tokens, credenciales o información privada en el frontend.
- ***Validación de datos:*** Evitar confiar solo en la validación del frontend y reforzarla en el backend.

CROSS-SITE SCRIPTING (XSS):

Ej: Un formulario de comentarios en tu aplicación, donde los usuarios pueden ingresar texto y enviarlo.

```
<input type="text" id="userInput" placeholder="Escribe algo">
<button onclick="submitComment()">Enviar</button>
<div id="output"></div>

<script>
  function submitComment() {
      const input = document.getElementById("userInput").value;
      document.getElementById("output").innerHTML = "Comentario: " + input;
  }
</script>
```

Un atacante podría ingresar el siguiente script en el campo de entrada:

```
<script>alert('¡Has sido hackeado!');</script>
```

Esto se inyecta directamente en el innerHTML, ejecutándose en el navegador de cualquier usuario que vea el comentario.

Consecuencias: Robo de cookies, ejecución de malware, redirección a sitios maliciosos, etc.

CROSS-SITE REQUEST FORGERY (CSRF):

Ejemplo de un sitio web donde los usuarios pueden cambiar su contraseña enviando un formulario POST a:

```
https://victima.com/cambiar-password
```

El código HTML de ese formulario en el sitio legítimo podría verse así:

```html
<form action="https://victima.com/cambiar-password" method="POST">
    <input type="password" name="newPassword" placeholder="Nueva contraseña">
    <button type="submit">Cambiar contraseña</button>
</form>
```

CROSS-SITE REQUEST FORGERY (CSRF):

Si un usuario está autenticado en **victima.com** y un atacante lo engaña para visitar un sitio malicioso, el atacante puede hacer que el navegador de la víctima envíe una solicitud maliciosa sin que el usuario se dé cuenta.

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Hackeando...</title>
</head>
<body>
    <h1>¡Oferta especial! Haz clic aquí para ganar un premio</h1>
    <img src="https://victima.com/cambiar-password?newPassword=hacked123" style="display:none
</body>
</html>
```

CROSS-SITE REQUEST FORGERY (CSRF):

Soluciones para Prevenir CSRF

Usar Tokens CSRF

El backend debe generar un token único por sesión y verificarlo en cada solicitud POST.

Usar Encabezados Origin y Referer en el Backend

El backend debe validar que la solicitud provenga del dominio correcto:

```
@app.route("/cambiar-password", methods=["POST"])
def cambiar_password():
    if request.headers.get("Origin") != "https://victima.com":
        return "CSRF detectado", 403
```

CROSS-SITE REQUEST FORGERY (CSRF):

Soluciones para Prevenir CSRF

Usar Tokens CSRF

El backend debe generar un token único por sesión y verificarlo en cada solicitud POST.

Usar Encabezados Origin y Referer en el Backend

El backend debe validar que la solicitud provenga del dominio correcto:

```
@app.route("/cambiar-password", methods=["POST"])
def cambiar_password():
    if request.headers.get("Origin") != "https://victima.com":
        return "CSRF detectado", 403
```

EXPOSICIÓN DE DATOS SENSIBLES:

```
<script>
    const apiKey = "12345-SECRET-API-KEY";  // ❌ ¡Nunca guardes claves en el frontend!

    fetch(`https://api.ejemplo.com/data?key=${apiKey}`)
        .then(response => response.json())
        .then(data => console.log(data));
</script>
```

¿Por qué es un problema?

Cualquier usuario puede abrir la consola del navegador (F12) y ver la clave API expuesta.
Un atacante puede usar esa clave para acceder a datos privados de la API.

VALIDACIÓN DE DATOS

Tenemos dos formas de validar los datos:

CON HTML5:

proyectos → frontend_vulnerable → validacion de datos → validacionHTML.html

VALIDACION COMPLETA CON JAVASCRIPT:

proyectos → frontend_vulnerable → validacion de datos → validacionJS.html