Reporte de Vulnerabilidad

Archivo: modal.php

Código Analizado:

```
<div id="modalmantenimiento" class="modal fade" tabindex="-1" aria-labelledby="myModalLabel"
aria-hidden="true" style="display: none;">
    <div class="modal-dialog modal-lg">
        <div class="modal-content">
            <div class="modal-header bg-primary text-white">
                <h5 class="modal-title" id="lbltitulo">Detalle de Factura</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>


            </div>

            <div class="modal-body">
                <div class="table-responsive">
                    <table id="detalleTabla" class="table table-bordered table-striped">
                        <thead class="table-light">
                            <tr>
                                <th>ID Producto</th>
                                <th>Cliente</th>
                                <th>Producto</th>
                                <th>Precio</th>
                                <th>Cantidad</th>
                                <th>Iva</th>
                                <th>Saldo Iva</th>
                                <th>SubTotal</th>
                                <th>Total</th>
                            </tr>
                        </thead>
                        <tbody>
                            <!-- Se rellenará dinámicamente -->
                        </tbody>
                    </table>
                </div>
            </div>
            <div class="modal-footer">


                <button type="reset" class="btn btn-light" data-bs-
dismiss="modal">Cerrar</button>
                <button type="submit" name="action" value="add" class="btn btn-
primary">Guardar</button>
            </div>
        </div>
    </div>
</div>
```

Análisis: ```html
Vulnerabilidades de Seguridad

Posible Vulnerabilidad XSS

Tipo: Cross-Site Scripting (XSS)
Línea Aproximada: Rellenado dinámico de la tabla `#detalleTabla` (`tbody`).
Descripción: Si los datos que se utilizan para rellenar la tabla (ID Producto, Cliente, Producto, etc.) provienen de una fuente no confiable (por ejemplo, entrada del usuario, base de datos vulnerable), y no se sanitizan correctamente antes de insertarlos en el DOM, un atacante podría inyectar código JavaScript malicioso. Este código se ejecutaría en el navegador del usuario cuando la página se renderiza.
Mitigación:

1. **Sanitización de Entrada:** Antes de mostrar cualquier dato en la tabla, especialmente los campos `Cliente` y `Producto`, utilizar funciones de escape HTML. En JavaScript, se puede usar una función para reemplazar caracteres especiales como &, , >, ", y ' con sus correspondientes entidades HTML (&, <, >, ", ').
2. **Validación de Entrada:** Validar los datos antes de guardarlos en la base de datos o usarlos en la aplicación. Asegurarse de que cumplan con el formato y tipo esperados.
3. **Content Security Policy (CSP):** Implementar una política CSP para controlar qué recursos puede cargar el navegador. Esto puede ayudar a prevenir la ejecución de código malicioso inyectado.

Posible Vulnerabilidad CSRF

Tipo: Cross-Site Request Forgery (CSRF)
Línea Aproximada: `<button type="submit" name="action" value="add" class="btn btn-primary">Guardar</button>`
Descripción: El formulario que envía la factura, tiene un botón submit que puede ser manipulado por un atacante, haciendo que el usuario ejecute acciones no deseadas sin su conocimiento.
Mitigación:

1. **Tokens CSRF:** Generar un token único y secreto para cada sesión de usuario. Incluir este token como un campo oculto en el formulario. En el servidor, verificar que el token enviado con la solicitud coincida con el token almacenado para la sesión del usuario antes de procesar la solicitud.
2. **Misma política de origen (SameSite Cookies):** Configurar las cookies de sesión con el atributo `SameSite` establecido en `Strict` o `Lax` (dependiendo de las necesidades de la aplicación). Esto ayuda a prevenir que las cookies se envíen con solicitudes entre sitios.

Métricas de Calidad del Código

Legibilidad

El código HTML en sí mismo es bastante legible debido al uso de clases de Bootstrap y nombres de atributos descriptivos. Sin embargo, la legibilidad podría mejorarse aún más con:

- Comentarios más detallados, especialmente en la parte donde se rellena dinámicamente la tabla.
- Consistencia en el uso de espacios y sangrías.

Acoplamiento

El código HTML tiene un acoplamiento moderado con Bootstrap. Si bien Bootstrap facilita la creación de interfaces de usuario, el código se vuelve dependiente de esta biblioteca. Si se quisiera cambiar el framework CSS, sería necesario modificar el HTML.

Complejidad

La complejidad del HTML es baja en este fragmento. La lógica principal (rellenar la tabla dinámicamente) reside probablemente en el JavaScript asociado.

Duplicación

En este fragmento no se observa duplicación significativa.

Solución Propuesta

1. **Implementar Sanitización de Datos:** Antes de insertar los datos en la tabla, usar una función de escape HTML para prevenir ataques XSS. Ejemplo: ```javascript function escapeHtml(text) { var map = { '&': '&', '': '<', '>': '>', '"': '"', "'": ''' }; return text.replace(/[&>"']/g, function(m) { return map[m]; }); } // Ejemplo de uso al rellenar la tabla: var producto = escapeHtml(data.producto); // Suponiendo que 'data.producto' viene de la base de datos. ```

2. **Añadir Token CSRF:** Implementar un token CSRF para proteger el formulario contra ataques CSRF. ```html ``` Y verificar el token en el servidor antes de procesar la solicitud.
3. **Validación de Entrada:** Validar todos los datos que se reciben del cliente antes de utilizarlos en la aplicación o guardarlos en la base de datos.

```
```