Reporte de Vulnerabilidad

Archivo: usuario.php

Código Analizado:

```
<?php
    require_once("ClassUsuario.php");
    $objUsuario = new Usuario("Andres Cardona","andres@info.com","Admin");
    $objUsuario1 = new Usuario("Andresa","andresa@info.com","Cliente");
    echo Usuario::$strEstado;


    echo $objUsuario->gerPerfil();
    echo $objUsuario1->gerPerfil();
    echo $objUsuario1->setCambiarClave("123456");

    echo $objUsuario1->gerPerfil();

?>
```

Análisis: ```html
Vulnerabilidades de Seguridad

Ausencia de Manejo de Datos Sensibles y Claves Hardcodeadas

Tipo: Exposición de información sensible y riesgo de acceso no autorizado.
Línea aproximada: `$objUsuario1->setCambiarClave("123456");`
Descripción: La clave "123456" está codificada directamente en el código. Esto representa un riesgo de seguridad significativo. Si el código se compromete, la clave se revelará fácilmente. Además, la ausencia de un mecanismo robusto para manejar las contraseñas (ej: hasheo) es una vulnerabilidad crítica.
Mitigación:

- Nunca almacenes contraseñas en texto plano. Utiliza funciones de hashing seguras como `password_hash()` para almacenar las contraseñas hasheadas.
- Implementa un sistema de gestión de contraseñas que requiera contraseñas fuertes y políticas de cambio de contraseña regulares.
- Evita hardcodear cualquier dato sensible en el código fuente. Configura estos datos en archivos de configuración protegidos.

Ausencia de Validación de Entrada

Tipo: Inyección (potencialmente SQL Injection o Cross-Site Scripting si `gerPerfil` usa la entrada del usuario sin validación).
Línea aproximada: `echo $objUsuario->gerPerfil();` y `echo $objUsuario1->gerPerfil();` (si la función `gerPerfil` usa datos del usuario).
Descripción: Si la función `gerPerfil()` utiliza datos del usuario (nombre, email, etc.) sin validarlos, existe el riesgo de inyección SQL o XSS. Por ejemplo, si el nombre del usuario se inserta directamente en una consulta SQL o se muestra en la página web sin ser escapado.
Mitigación:

- Valida y escapa todas las entradas del usuario antes de usarlas en consultas SQL o mostrarlas en la página web.
- Utiliza sentencias preparadas (prepared statements) para evitar la inyección SQL.
- Escapa la salida HTML para evitar la inyección XSS. Utiliza funciones como `htmlspecialchars()` o `strip_tags()`.

Métricas de Calidad del Código

Legibilidad

El código es relativamente legible dado su tamaño. Sin embargo, podría mejorar con:

- Comentarios que expliquen la funcionalidad de las funciones y clases.
- Nombres de variables y funciones más descriptivos.
- Una estructura de directorios más organizada (especialmente para un proyecto más grande).

Acoplamiento

El código presenta acoplamiento a la clase `Usuario`. Si la clase `Usuario` cambia, el código que la utiliza también debe cambiar. Esto se puede mitigar con:

- Utilizando interfaces para definir el comportamiento de la clase `Usuario`.
- Aplicando el principio de inversión de dependencias.

Complejidad

La complejidad ciclomática del código es baja, dado su pequeño tamaño. Sin embargo, la complejidad podría aumentar significativamente en la clase `Usuario`, dependiendo de la lógica interna de sus métodos (especialmente `gerPerfil` y `setCambiarClave`). Para reducir la complejidad:

- Divide los métodos complejos en métodos más pequeños y manejables.
- Utiliza patrones de diseño para simplificar la lógica.

Duplicación

No hay duplicación evidente en el fragmento de código proporcionado. Sin embargo, la duplicación podría existir en la clase `Usuario`, si la lógica se repite en diferentes métodos.

Solución Propuesta

La siguiente solución incluye un ejemplo de hasheo de contraseñas usando `password_hash()` y una estructura básica. La clase `Usuario` (que no se incluyó en el código original) necesita implementación para tener validación de datos de entrada y sanitización.

```
<?php
require_once("ClassUsuario.php");

// Clase Usuario (ejemplo simplificado - implementar validación y sanitización)
class Usuario {
    private $nombre;
    private $email;
    private $rol;
    private $password; // Almacenar el hash

    public static $strEstado = "Activo";

    public function __construct($nombre, $email, $rol, $password) {
        $this->nombre = $nombre;
        $this->email = $email;
        $this->rol = $rol;
        $this->password = password_hash($password, PASSWORD_DEFAULT); // Hashear la contraseña
    }

    public function gerPerfil() {
        // Implementar validación y sanitización de datos antes de mostrar
        return "Nombre: " . htmlspecialchars($this->nombre) . "<br>Email: " .
htmlspecialchars($this->email) . "<br>Rol: " . htmlspecialchars($this->rol) . "<br>";
    }

    public function setCambiarClave($nuevaClave) {
        $this->password = password_hash($nuevaClave, PASSWORD_DEFAULT);
```

```
        return "Contraseña actualizada correctamente.<br>";
    }

    public function verificarPassword($password) {
        return password_verify($password, $this->password);
    }

    //Getters y setters (opcional)
}

// Crear instancias de Usuario (ejemplo)
$objUsuario = new Usuario("Andres Cardona", "andres@info.com", "Admin", "UnaClaveSegura");
$objUsuario1 = new Usuario("Andresa", "andresa@info.com", "Cliente", "OtraClaveSegura");

echo Usuario::$strEstado . "<br>";
echo $objUsuario->gerPerfil();
echo $objUsuario1->gerPerfil();

// Cambiar contraseña (ejemplo)
echo $objUsuario1->setCambiarClave("NuevaClaveSegura") . "<br>";

echo $objUsuario1->gerPerfil();

// Verificar la contraseña (ejemplo)
if ($objUsuario1->verificarPassword("NuevaClaveSegura")) {
    echo "Contraseña verificada correctamente.<br>";
} else {
    echo "Contraseña incorrecta.<br>";
}

?>
```

```
```