IDENTIFICACIÓN DE VULNERABILIDADES BASADOS EN PROGRAMACIÓN: BUENAS PRÁCTICAS DE PROGRAMACIÓN PARA SISTEMAS INFORMÁTICOS MÁS SEGUROS

Programming-based vulnerability identification: best programming practices for more secure computer systems

ANGIE LORENA PUERTA CORREDOR[1], MICHAEL ALEXANDER BARRERA LASSO[2], EVELYN GARNICA[3]

Recibido:10 de diciembre de 2024. Aceptado:31 de diciembre de 2024

DOI: https://doi.org/10.21017/rimci.1132

Resumen

En este documento se analizarán las diversas vulnerabilidades basados en programación. Se presentará un análisis exhaustivo de la investigación llevada a cabo, junto con ejemplos concretos que ilustran estas vulnerabilidades. Además, se proporcionarán recomendaciones prácticas para la prevención y mitigación de ataques maliciosos perpetrados por agentes adversos. El objetivo de este trabajo es concienciar a las empresas sobre la crítica importancia de implementar robustas medidas de ciberseguridad.

Palabras clave: vulnerabilidades; ciberataques; hackers; investigación; Mitigacion.

Abstract

This document will analyze various programming-based vulnerabilities. It will present a comprehensive analysis of the research conducted, along with concrete examples illustrating these vulnerabilities. In addition, practical recommendations will be provided for the prevention and mitigation of malicious attacks carried out by adversarial agents. The objective of this work is to raise awareness among companies about the critical importance of implementing robust cybersecurity measures.

Keywords: vulnerabilities; cyber-attacks; hackers; research; Mitigation.

I. Introducción

DADA LA creciente dependencia de la tecnología en diversos contextos, incluidos los empresariales y personales, se ha generado un conjunto de vulnerabilidades que pueden comprometer la operatividad de las organizaciones afectadas. Los ciberataques dirigidos a sistemas informáticos han aumentado significativamente en los últimos años, con amenazas como inyecciones de código, desbordamientos de búfer, y errores en la gestión de sesiones, que representan riesgos críticos para la integridad de la información[1]. Un estudio de la *Cybersecurity and Infrastructure Security Agency* (CISA) revela que el 95 % de las vulnerabilidades explotadas en aplicaciones web

1 Estudiante de décimo semestre Ingeniería de Sistemas de la Corporación Universitaria Republicana. ORCID: https://orcid.org/0009-0001-8838-0401 Correo electrónico: al.puerta@urepublicana.edu.co

2 Estudiante décimo semestre Ingeniería de Sistemas de la Corporación Universitaria Republicana. ORCID: https://orcid.org/0009-0009-2999-4583 Correo electrónico: ma.barrera@urepublicana.edu.co

3 Doctora en Educación, Magister en Dirección de Proyectos, Ingeniera de Diseño y Automatización Electrónica. Docente Investigadora de la Corporación Universitaria Republicana. ORCID: https://orcid.org/0000-0002-6205-7817 Correo electrónico: egarnicae@urepublicana.edu.co

están relacionadas con errores de programación que podrían haberse evitado con buenas prácticas de desarrollo[2].

Este documento se enfocará en la investigación y análisis de las diversas vulnerabilidades presentes en la programación de los sistemas esenciales para el funcionamiento eficiente de estas entidades. Se analizarán casos específicos de vulnerabilidades detectadas en aplicaciones modernas, incluyendo fallas en la validación de entradas, inadecuada gestión de credenciales y configuraciones inseguras que podrían permitir el acceso no autorizado a datos sensibles[3]. Además, se proporcionarán recomendaciones específicas para la prevención y mitigación de las vulnerabilidades identificadas, siguiendo estándares de seguridad reconocidos como el OWASP Top 10 y las directrices del *National Institute of Standards and Technology* (NIST)[4], [5].

Al concluir, se busca fomentar una conciencia crítica sobre la importancia de implementar estas recomendaciones y reconocer que la ciberseguridad se ha convertido en un aspecto fundamental que requiere atención y conocimiento en la actualidad. La adopción de prácticas seguras durante el ciclo de desarrollo del software no solo protege la confidencialidad e integridad de los datos, sino que también mejora la resiliencia de las organizaciones frente a posibles ataques[6].

II. Problema a resolver

En la actualidad, el desarrollo de sistemas informáticos enfrenta numerosos desafíos relacionados con la seguridad, debido al incremento de ataques cibernéticos que explotan vulnerabilidades presentes en el código fuente de aplicaciones críticas[7]. Un informe del *National Institute of Standards and Technology* (NIST) destaca que más del 70 % de las vulnerabilidades detectadas en sistemas empresariales provienen de errores de programación que podrían haberse prevenido con prácticas adecuadas[4]. Estos errores incluyen la falta de validación de datos de entrada, inyecciones de código, gestión insegura de sesiones y accesos indebidos a recursos del sistema[1].

El informe *OWASP Top Ten* identifica que la mayoría de los ciberataques exitosos, como inyecciones SQL, ataques de scripts entre sitios (XSS) y configuraciones mal gestionadas, son consecuencia de una deficiente implementación de controles de seguridad en las etapas de desarrollo del software[3]. Esta situación es especialmente crítica en aplicaciones web y móviles, donde la exposición a usuarios externos aumenta el riesgo de explotación de vulnerabilidades[8]. De acuerdo con un estudio de Verizon, el 43 % de las brechas de seguridad en 2023 se originaron por vulnerabilidades conocidas que no fueron corregidas a tiempo[9].

Por esta razón, se han desarrollado métodos de seguridad para la programación. Sin embargo, una de las principales barreras, tanto para desarrolladores nuevos como experimentados, es la falta de conocimiento y comprensión de las vulnerabilidades más comunes, así como de aquellas menos conocidas que pueden comprometer la integridad, confiabilidad y disponibilidad de los datos. La falta de concientización sobre la importancia de adoptar buenas prácticas de programación segura y la ausencia de metodologías de desarrollo centradas en la seguridad (como DevSecOps) incrementan el riesgo de que las organizaciones sufran pérdidas financieras y de reputación[5]. Además, la ausencia de análisis estático y dinámico del código durante el ciclo de vida del desarrollo permite que vulnerabilidades críticas permanezcan sin ser detectadas hasta que son explotadas[6].

Los riesgos asociados con estas vulnerabilidades incluyen, por ejemplo, la inyección de código, que, aunque es un tipo de ataque frecuente, sigue siendo difícil de abordar adecuadamente. Además, las vulnerabilidades en las dependencias, como el uso de bibliotecas externas no seguras, representan otra amenaza significativa que puede poner en peligro la seguridad general del sistema. Estas deficiencias pueden tener consecuencias graves, como filtraciones de datos, pérdida de confidencialidad, daños a la reputación de las organizaciones y, en casos extremos, el colapso de servicios críticos.

Por tanto, es imperativo investigar las vulnerabilidades más frecuentes en la programación y proponer recomendaciones basadas en estándares reconocidos para prevenir y mitigar estos riesgos de manera efectiva.

III. Marco de referencia

Conceptos fundamentales de ciberseguridad y programación segura

La ciberseguridad es el conjunto de prácticas, tecnologías y procesos diseñados para proteger sistemas, redes y datos contra ataques cibernéticos[10]. En el contexto del desarrollo de software, la programación segura implica aplicar técnicas de codificación que minimicen la exposición del sistema a vulnerabilidades potenciales, reduciendo así el riesgo de ataques malintencionados[11]. Según el *National Institute of Standards and Technology* (NIST), el 80 % de las vulnerabilidades explotadas se originan durante la fase de desarrollo del software, lo que subraya la importancia de integrar controles de seguridad desde las primeras etapas del ciclo de vida del desarrollo de software (SDLC)[4].

El modelo DevSecOps ha ganado popularidad como enfoque para integrar la seguridad en todo el ciclo de vida del desarrollo de software, garantizando que la seguridad no sea una etapa posterior, sino un componente integral[12]. Este modelo promueve la automatización de pruebas de seguridad y el análisis continuo de vulnerabilidades para detectar posibles fallas antes de que el software sea implementado[13].

Tipos comunes de vulnerabilidades en programación

Las vulnerabilidades en la programación pueden clasificarse en diversas categorías, de acuerdo con su origen y su impacto en el sistema. Los ataques más comunes incluyen:

La inyección de código, especialmente inyección SQL (SQLi) y Cross-Site Scripting (XSS), es una de las vulnerabilidades más críticas que afecta aplicaciones web. Un atacante puede manipular entradas de usuario no validadas para ejecutar comandos no autorizados en la base de datos o scripts maliciosos en el navegador del usuario[1]. De acuerdo con OWASP, la inyección SQL continúa siendo una de las principales amenazas para la seguridad de las aplicaciones web[15].

XSS permite a un atacante inyectar scripts maliciosos en páginas web vistas por otros usuarios. Estos scripts pueden robar información sensible o ejecutar acciones en nombre del usuario sin su conocimiento (Fig. 1).

Fig. 1. Imagen - XSS attacks.

Mitigación: sanear las entradas de los usuarios antes de mostrarlas en la página web. Utilizar cabeceras de seguridad como Content-Security-Policy (CSP) para mitigar la ejecución de scripts maliciosos. Aplicar políticas de escape de HTML en las respuestas del servidor[15].

El ataque de falsificación de petición en sitios cruzado "CSRF", es un ataque cibernético que engaña a los usuarios que se encuentran autenticados para que ejecuten peticiones maliciosas no deseadas sin que este se dé cuenta. Este método normalmente implica peticiones de cambio de estado ya que estos no reciben una respuesta, uno de los ejemplos son cambio de contraseña, borrar registros, envió de mensajes o compra de productos. Este ataque se utiliza la ingeniería social para que los usuarios por medio de correo electrónicos o chats de clic en los enlaces maliciosos[16].

Para que un ataque CSRF tenga un éxito se deben dar las siguientes condiciones:

- El usuario debe iniciar sesión en un aplicativo web que utilice las cookies o variables de sesión para sostener la autenticación.
- El atacante debe crear una solicitud HTTP similar que tenga los parámetros necesarios para que el servidor pueda procesarla.

- El aplicativo web debe utilizar las cookies para identificar al usuario que esta autenticado.
- La solicitud falsificada no debe requerir parámetros adicionales que el atacante no los pueda conseguir con facilidad.
- El sistema debe carecer de mecanismos para protegerse contra CSRF.

El CSRF permite a un atacante ejecutar acciones no deseadas en una aplicación web en la que el usuario está autenticado. Por ejemplo, puede realizar transferencias de dinero o cambiar la configuración de una cuenta sin el consentimiento del usuario (Fig. 2).

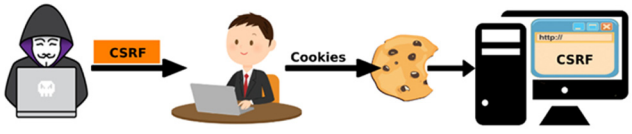

Fig. 2. Imagen de ejemplo- Explicación de CSRF - Security Souls.

Mitigación: Implementar tokens anti-CSRF en los formularios y asegurarse de que cada solicitud que modifique el estado del servidor esté autenticada[16].

El desbordamiento de búfer ocurre cuando un programa escribe más datos de los que un búfer puede manejar, lo que puede provocar la corrupción de memoria, fallos del sistema o la ejecución de código malicioso[8]. Este tipo de vulnerabilidad es más común en lenguajes como C y C++, debido a su manejo manual de memoria[17] (Fig. 3).

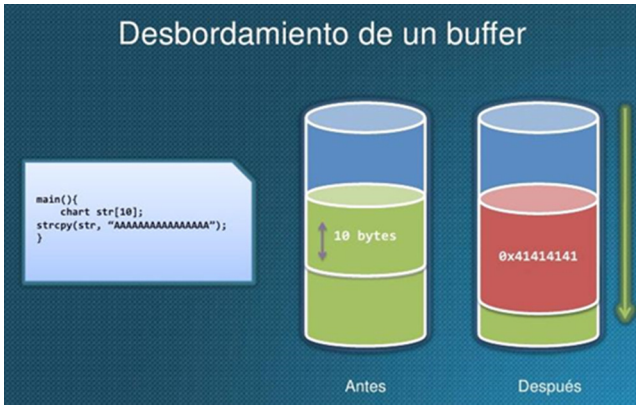

Fig. 3. Desbordamiento de un buffer.

***Mitigación*:** usar lenguajes de programación que gestionen automáticamente la memoria (como Java, Python) o asegurarse de que el manejo de memoria en C/C++ esté bien controlado. Además, implementar técnicas como la protección de pila (stack protection), y la ejecución de código en zonas no ejecutables (NX bit)[18].

Gestión insegura de sesiones y autenticación, puede llevar a la apropiación de cuentas y al acceso no autorizado. Los atacantes pueden explotar sesiones no invalidadas o utilizar cookies interceptadas para obtener acceso privilegiado[19]. La implementación incorrecta de tokens de autenticación, como JWT, también puede exponer sistemas a riesgos importantes si no se protegen adecuadamente[20].

Configuraciones por defecto inseguras, permisos excesivos y la exposición de interfaces administrativas, pueden permitir a los atacantes acceder a partes críticas del sistema sin restricciones[12]. Este tipo de errores, si no son corregidos a tiempo, pueden comprometer la integridad del sistema y exponer información sensible.

Las configuraciones incorrectas y errores de permisos son fallos comunes que surgen cuando los sistemas o servicios no se ajustan adecuadamente a estándares de seguridad, como dejar puertos abiertos innecesariamente o asignar privilegios de administrador a usuarios sin justificación. Estos descuidos facilitan accesos no autorizados, manipulación de datos o incluso el control total del sistema por parte de un atacante. Prevenirlos requiere auditorías periódicas, aplicar el principio de menor privilegio y asegurarse de que los servicios estén correctamente configurados según el entorno en el que operan[15].

El uso de librerías y dependencias, que contienen vulnerabilidades conocidas puede ser una fuente significativa de riesgo para el sistema. Las vulnerabilidades de las librerías pueden ser explotadas si no se actualizan a sus versiones seguras.

***Mitigación*:** mantener las dependencias y librerías siempre actualizadas, y utilizar herramientas como npm audit, pip audit, o OWASP Dependency-Check para identificar vulnerabilidades. Asi como, monitorear las actualizaciones de seguridad de las librerías y aplicar parches rápidamente[21].

IV. Buenas Prácticas de Programación para Sistemas Seguros

El desarrollo de sistemas seguros exige una atención rigurosa a las prácticas de programación que permitan mitigar el riesgo de explotación de vulnerabilidades. La seguridad debe integrarse desde las fases iniciales del ciclo de vida del software, adoptando una mentalidad proactiva y no reactiva. A continuación, se presentan algunas de las mejores prácticas ampliamente reconocidas en el ámbito de la ciberseguridad y el desarrollo seguro[22].

a. Validación y saneamiento de entradas

Una de las fuentes más comunes de vulnerabilidades es la falta de validación adecuada de los datos proporcionados por el usuario. Las técnicas de inyección de código, como SQL injection o Cross-Site Scripting (XSS), explotan precisamente esta debilidad. OWASP[23] recomienda aplicar una validación robusta del lado del servidor y realizar un saneamiento (sanitization) que elimine caracteres o patrones maliciosos antes de procesar los datos.

Nunca se debe confiar en la entrada del usuario, independientemente de su origen, ya que incluso sistemas internos pueden ser manipulados si no existen controles adecuados. Por ello, es esencial aplicar listas blancas (whitelisting) de valores permitidos y evitar el uso exclusivo de listas negras (blacklisting), que pueden ser evadidas con facilidad[24].

b. Gestión segura de credenciales y autenticación

El manejo de contraseñas y tokens de autenticación es otro componente crítico para la seguridad del software. Las credenciales nunca deben almacenarse en texto plano, sino utilizando algoritmos de hash seguros como bcrypt, Argon2 o PBKDF2, con sal (salt) aleatoria para evitar ataques de diccionario y rainbow tables[4].

También se recomienda implementar autenticación multifactor (MFA) y mecanismos de bloqueo tras múltiples intentos fallidos. La falta de estas medidas puede facilitar ataques de fuerza bruta y comprometer la seguridad del sistema.

Uso de controles de acceso basados en Roles (RBAC)

El modelo de Control de Acceso Basado en Roles (RBAC) permite asignar permisos en función del rol que un usuario desempeña en el sistema, reduciendo así el principio de privilegios mínimos. Este enfoque, ampliamente adoptado en entornos empresariales, mejora la trazabilidad y disminuye la exposición a riesgos por accesos indebidos[25]

Implementar RBAC permite limitar las operaciones que un usuario puede ejecutar, asegurando que cada actor tenga acceso únicamente a los recursos necesarios para su función. Esto contribuye a la prevención de escalamiento de privilegios y reduce la superficie de ataque.

Programación defensiva

La **programación defensiva** consiste en escribir código anticipándose a posibles errores, mal uso o comportamientos inesperados. Se basa en suposiciones mínimas sobre el entorno de ejecución y favorece el manejo explícito de errores, validaciones internas y el uso de condiciones de falla segura (fail-safe conditions)[11].

Esta estrategia busca fortalecer la resiliencia del software, asegurando que aún en situaciones anómalas, el sistema no se comporte de forma peligrosa o inestable. La programación defensiva no solo es una práctica técnica, sino una postura mental orientada a la seguridad[26].

Adoptar estas buenas prácticas de programación no solo contribuye a mitigar vulnerabilidades comunes, sino que constituye un componente esencial en la construcción de sistemas informáticos resilientes. La validación de entradas, la gestión segura de credenciales, el uso de RBAC y la programación defensiva deben ser pilares en cualquier estrategia de desarrollo seguro.

V. Herramientas para la Detección de Vulnerabilidades

La identificación temprana de vulnerabilidades es un componente esencial en el ciclo de desarrollo seguro de software. Existen múltiples herramientas que permiten evaluar el estado de seguridad

de aplicaciones web, sistemas operativos y componentes de código, ya sea a través de análisis estático o dinámico. A continuación, se describen algunas de las más utilizadas en entornos profesionales y académicos:

- **OWASP ZAP (Zed Attack Proxy)**: Es una herramienta gratuita y de código abierto desarrollada por el proyecto OWASP, destinada a pruebas de seguridad en aplicaciones web. Permite identificar vulnerabilidades como inyecciones SQL, fallos de autenticación, y exposición de datos sensibles mediante técnicas de análisis activo y pasivo[23].

- **Burp Suite**: Plataforma integrada para pruebas de seguridad en aplicaciones web, muy utilizada en auditorías de seguridad por profesionales de pentesting. Su versión profesional incorpora funcionalidades avanzadas como fuzzing, análisis de tráfico HTTPS y detección automatizada de fallas en la lógica de negocio[27].

- **Nessus**: Escáner de vulnerabilidades ampliamente adoptado que permite identificar configuraciones inseguras, servicios obsoletos, errores de software, y fallas de seguridad en sistemas operativos, dispositivos de red y servidores de aplicaciones. Proporciona informes detallados con recomendaciones para la mitigación de riesgos[28].

- **SonarQube**: Herramienta de análisis estático de código fuente que permite detectar errores de calidad y problemas de seguridad, tales como inyecciones de comandos, bucles inseguros o funciones no controladas. Facilita la adopción de buenas prácticas de codificación segura, y es compatible con múltiples lenguajes de programación[29].

- **Nikto**: Escáner de servidores web que analiza configuraciones inseguras, archivos vulnerables, directorios expuestos, y versiones desactualizadas de software. Es útil para realizar evaluaciones rápidas y amplias sobre la superficie de ataque de un servidor web[30].

- **Snyk**: Esta herramienta se enfoca en entornos de desarrollo moderno, particularmente aquellos que dependen de paquetes y bibliotecas de código abierto. Detecta vulnerabilidades en dependencias, sugiere actualizaciones seguras, y permite integraciones CI/CD para asegurar el ciclo de vida del software desde el desarrollo[31].

El uso combinado de estas herramientas puede fortalecer significativamente la postura de seguridad de un sistema, al ofrecer diferentes niveles de análisis que abarcan desde la infraestructura hasta el código fuente.

VI. Normativas y estándares de seguridad

El cumplimiento de normativas y estándares internacionales es un pilar fundamental en la construcción de sistemas seguros y resilientes. Estas normativas proporcionan marcos de referencia que permiten establecer políticas, procesos y controles enfocados en la protección de la información y la gestión de riesgos.

- **ISO/IEC 27001**: Es el estándar internacional más reconocido para la gestión de la seguridad de la información. Define un marco sistemático para establecer, implementar, mantener y mejorar un Sistema de Gestión de Seguridad de la Información (SGSI), con el fin de proteger la confidencialidad, integridad y disponibilidad de los activos informacionales. Este estándar se complementa con **ISO/IEC 27002**, que proporciona un conjunto detallado de controles aplicables a distintos contextos organizacionales[32].

- **NIST Cybersecurity Framework (CSF)**: Desarrollado por el Instituto Nacional de Estándares y Tecnología de los Estados Unidos, el CSF propone un enfoque estructurado para la gestión de riesgos cibernéticos. Está compuesto por cinco funciones clave: *identificar, proteger, detectar, responder* y *recuperar*, lo cual facilita a las organizaciones establecer medidas adaptadas a su nivel de madurez y objetivos estratégicos[4].

- **PCI DSS (Payment Card Industry Data Security Standard)**: Este estándar es obli-

gatorio para todas las entidades que procesan, almacenan o transmiten información de tarjetas de pago. Sus requisitos incluyen la implementación de cifrado, segmentación de redes, monitoreo continuo, pruebas de penetración y políticas de control de acceso. Su cumplimiento ayuda a reducir el riesgo de fraudes financieros y violaciones de datos[3].

- **Reglamento General de Protección de Datos (GDPR)**: Vigente en la Unión Europea, el **GDPR** establece las obligaciones legales para el tratamiento de datos personales, incluyendo la obtención de consentimiento explícito, el derecho de acceso y supresión, así como la notificación de incidentes de seguridad. Este reglamento tiene impacto extraterritorial, por lo que aplica también a empresas fuera de la UE que traten datos de ciudadanos europeos[34].

- **COBIT (Control Objectives for Information and Related Technologies)**: Es un marco de gobierno y gestión de tecnologías de la información que permite alinear los objetivos empresariales con la gestión de TI. COBIT proporciona principios y modelos para la evaluación de riesgos, la planificación de controles, y el aseguramiento de la disponibilidad, integridad y confidencialidad de la información[35].

VII. Desarrollo

Se realizaron pruebas a un aplicativo con el fin de mostrar las principales vulnerabilidades al momento de programar, con esto sse abarcara las siguientes pruebas (Fig. 4 y 5):

- Inyección SQL
- Subida de archivos (Uploads)
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

La primera vulnerabilidad que evidenciamos son las contraseñas de los usuarios ya que no están encriptadas y esto hace que en sean que cualquier persona con accesos o en caso de robo de información pueda ingresar a la cuenta del usuario.

security_problems.users: 3 filas en total (aproximadamente)

id	user	password	email
1	pedro	456	pedro@test.com
4	maikol	1234	maikol@test.com
5	maikol1	12341	maikol1@test.com

Fig. 4. Datos de usuario en la tabla 'security_problems.users'.

Fig. 5. Tabla de usuarios y contraseñas sin encriptar.

Inyección SQL

Estas pruebas se realizaron al Login del sistema donde se ingresa con normalidad al sistema y después se introduce la inyección SQL para vulnerar el sistema (Fig. 6).

En la imagen se evidencia el código del Login realizado en PHP donde se remarcan las principales vulnerabilidades.

- **Rojo (las credenciales de la base de datos están en texto plano):** Esto genera que si en un ataque tienen acceso al código fuente del aplicativo puedan ver los accesos a la base de datos.

- **Azul (datos del formulario no están sanitizados): Al** no limpiar ni filtrar los datos de entrada no se ayudan a prevenir vulnerabilidades de seguridad ya que con estos se puede hacer inyección SQL o XSS.

- **Verde (Concatenación directa de variables en la consulta SQL):** Al construir la consulta directamente con las variables que ingreso el usuario ya que el atacante puede manipular los valores y pasar con éxito el login sin tener las credenciales.

```php
<?php
session_start();

$servername = "127.0.0.1";
$username = "root";
$password = "";
$dbname = "security_problems";

$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) die("Conexión fallida: " . $conn->connect_error);

$user = $_POST['user'];
$password = $_POST['password'];

$result = $conn->query("SELECT * FROM users WHERE user = '$user' AND password = '$password'");

if ($result->num_rows > 0)
{
    $_SESSION["user"] = $result->fetch_assoc();
    echo "El usuario ".$user." ha sido iniciado session con exito.";
}
else echo "Usuario o contraseña incorrectos";
```

Fig. 6. Código pruebas inyección SQL.

Pruebas de código *(Fig. 7)*

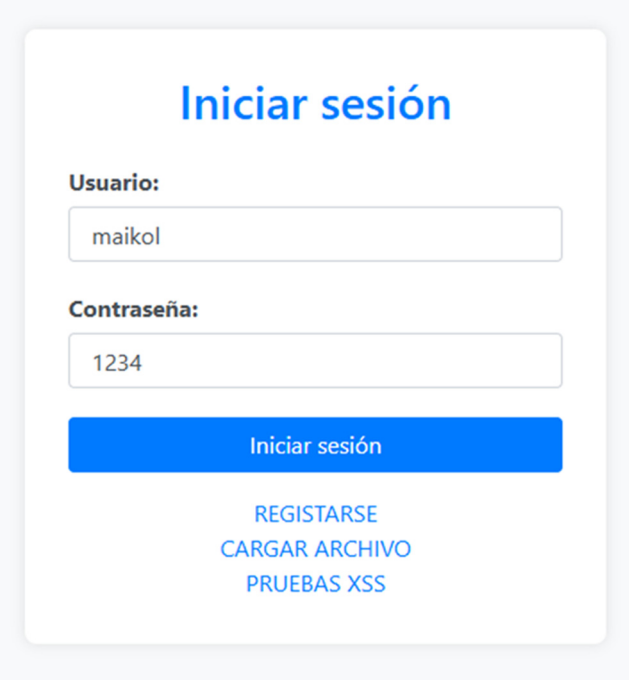

Fig. 7. Frontend del login.

Comenzando las pruebas desde el frontend se ingresa datos que existen en la base de datos para confirmar que haga login en el sistema (Fig. 8).

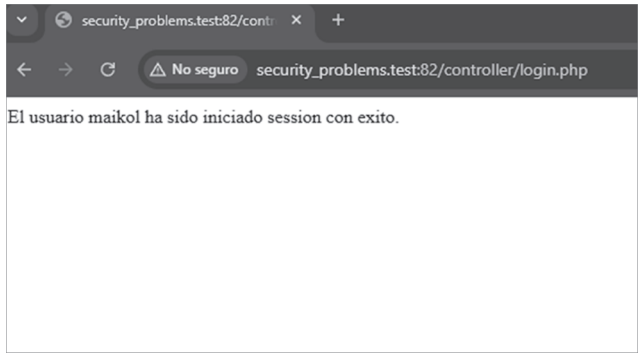

Fig. 8. Resultado de la petición del Frontend.

Ya que los datos son correctos el usuario inicio sesión satisfacción (Fig. 9).

En este caso se ingresan datos erróneos y no debe dejar iniciar sesión (Fig. 10).

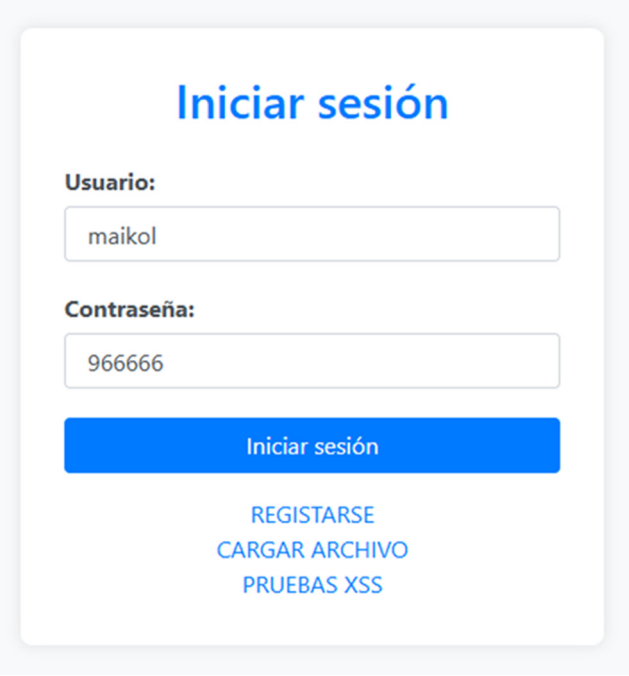

Fig. 9. Frontend del login con datos diferentes.

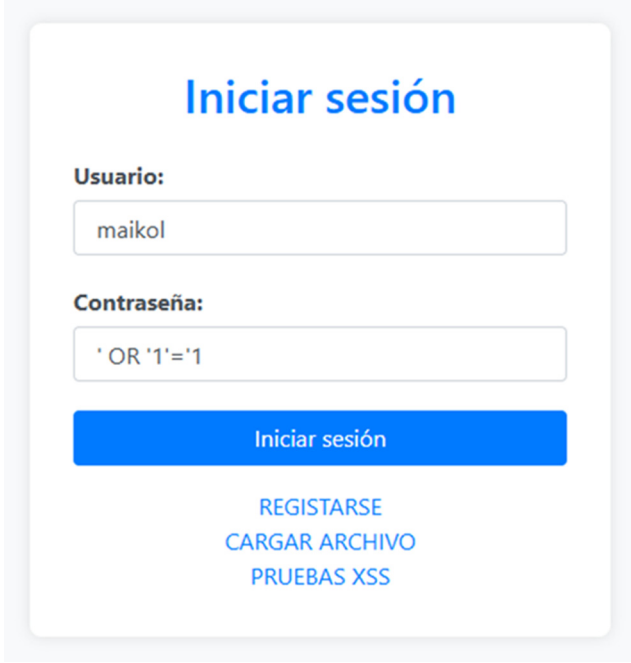

Fig. 11. Insertando inyección SQL login.

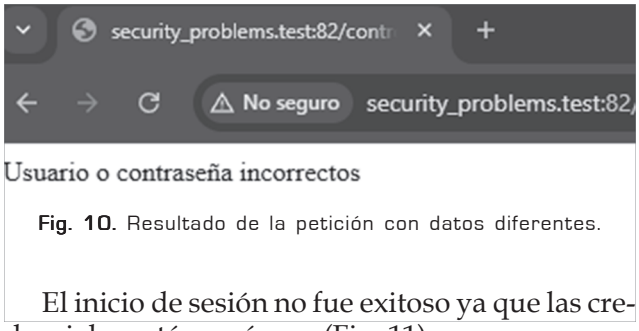

Fig. 10. Resultado de la petición con datos diferentes.

El inicio de sesión no fue exitoso ya que las credenciales están erróneas (Fig. 11).

En este caso se va a insertar inyección SQL poniendo la condición en la contraseña que se cumple, en la consulta se vería de la siguiente forma (Fig. 12 y 13).

Como se indica la condición de que 1 = 1 y esto es verdadero hace que el Login sea exitosa, haciendo que el sistema sea vulnerable a cualquier atacante SQL.

Subida de archivos (Uploads)

Estas pruebas se realizan en la carga de archivos del sistema donde se ingresa un archivo malicioso para que afecte a este en el momento que estén en el servidor (Fig. 14).

```
SELECT * FROM usuarios WHERE usuario = '' OR '1'='1' AND contrasena = '';
```

Fig. 12. Consulta realizada con inyección SQL.

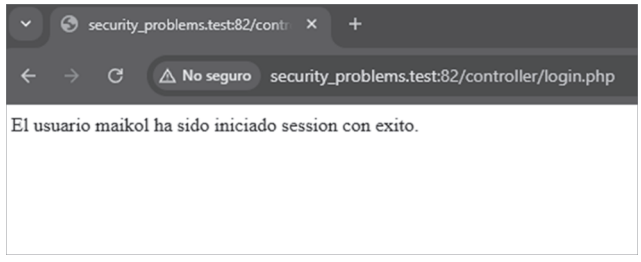

Fig. 13. Resultado de peticion con inyección SQL

```php
<?php

$directory = "../public/files/";
$root = $directory.basename($_FILES["file"]["name"]);

if (move_uploaded_file($_FILES["file"]["tmp_name"], $root))
    echo "El Archivo ".basename($_FILES["file"]["name"]). " ha sido subido.";

else echo "Error subiendo el archivo";
```

Fig. 14. Código uploads.

En el código de subida de archivos se identifican errores críticos (señalados en rojo, amarillo), además de una notable ausencia de mecanismos de validación.

- **Rojo (Subida sin verificar extensión):** Esta falta de validación permite la carga indiscriminada de cualquier tipo de archivo al servidor, incluyendo aquellos con contenido malicioso.
- **Amarillo (Limpieza y nombre de archivo):** La falta de validación en los nombres de archivo abre la puerta a la inclusión de caracteres especiales o código malicioso. Esto podría llevar a interpretaciones erróneas por parte del sistema, el reemplazo accidental de archivos existentes o incluso su corrupción.
- **Verde (Acceso vía navegador):** Si el directorio de almacenamiento es accesible vía web, cualquier persona con la URL podría visualizar los archivos o en el caso de ser archivo malicioso, ejecutarlo.

Pruebas al código *(Fig. 15).*

Fig. 15. Carga de archivo de prueba.

Se cargará un archivo de prueba desde el Frontend para verificar que el archivo exitosamente al servidor (Fig. 16 y 17).

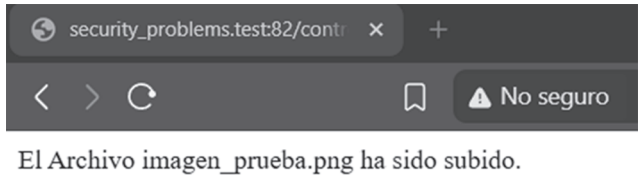

Fig. 16. Respueta subida de archivo.

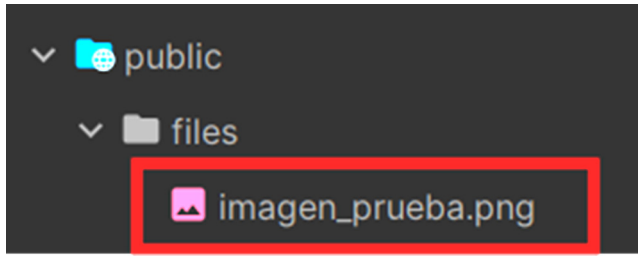

Fig. 17. Evidencia archivo en el servidor.

Con las anteriores imágenes se evidencia la subida del archivo de prueba al servidor (Fig. 18).

Se sube un archivo llamado **malicious.php** que contiene este código donde elimina todos los archivos donde se encuentre alojado este (Fig. 19).

Se evidencia en el Frontend el archivo a cargar en el servidor (Fig. 20 y 21).

El archivo fue cargado con el servidor de manera exitosa, al no tener una validación de extensión de archivos dejo ingresar un archivo malicioso sin ningún problema.

Al ingresar en la ruta http://security_problems. test/public/files/malicious.php "Ruta de prueba donde se encuentra el archivo" y lo ejecutara (Fig. 22 y 23).

Al ejecutar el software malicioso, borro todos los archivos en el servidor que se encontraban en el directorio.

Cross-Site Scripting (XSS)

Estas pruebas consisten en inyectar scripts en el envío de mensajes para evaluar si el servidor los ejecuta, revelando posibles vulnerabilidades (Fig. 24).

```php
$folder = __DIR__;

$files = scandir($folder);

foreach ($files as $file) {
    if ($file != "." && $file != "..") {
        $root = $folder . "/" . $file;
        if (is_file($root)) {
            if (unlink($root))
                echo "Archivo borrado: " . $root . "<br>";
            else
                echo "Error al borrar el archivo: " . $root . "<br>";
        }
    }
}

echo "Archivos eliminados";
```

Fig. 18. Código de archivo malicioso.

Fig. 19. Subida de archivo malicioso desde el Frontend.

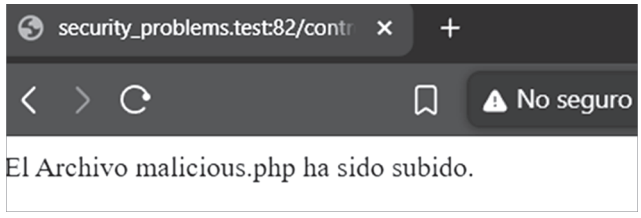

Fig. 20. Respuesta archivo malicioso cargado.

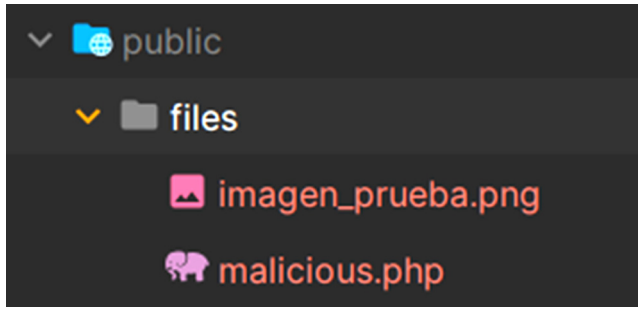

Fig. 21. Evidencia de archivo subido al servidor.

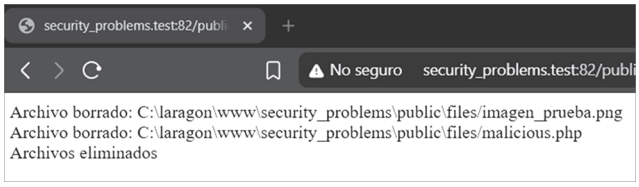

Fig. 22. Respueta servidor al ejecutar archivo malicoso.

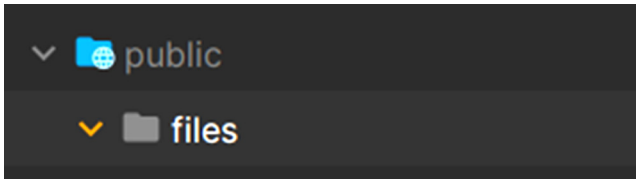

Fig. 23. Evidencia de archivos en el servidor.

```php
if (isset($_POST['message']))
{
    $message = $_POST['message'];
    echo "<p>Tu mensaje fue: " . $message . "</p>";

}
else
    echo "<p>No se recibió ningún mensaje.</p>";

echo "<p><a href='../view/xss.html'>Volver al formulario</a></p>";
```

Fig. 24. Código pruebas XSS.

El código muestra en rojo errores que evidencian la falta de limpieza de los datos ingresados, permitiendo la inyección de scripts por parte del usuario sin restricciones.

Rojo (Datos entrantes sin validar): La falta de validación y sanitización de los datos del usuario al insertarlos directamente en el HTML permite a un atacante redirigir a los usuarios a sitios web maliciosos para robar sus credenciales o inyectar código JavaScript para acceder y robar sus cookies.

Pruebas al código *(Fig. 25).*

Se envía un mensaje de prueba para confirmar la ejecución exitosa del proceso (Fig. 26).

Se constata que el mensaje fue enviado exitosamente (Fig. 27).

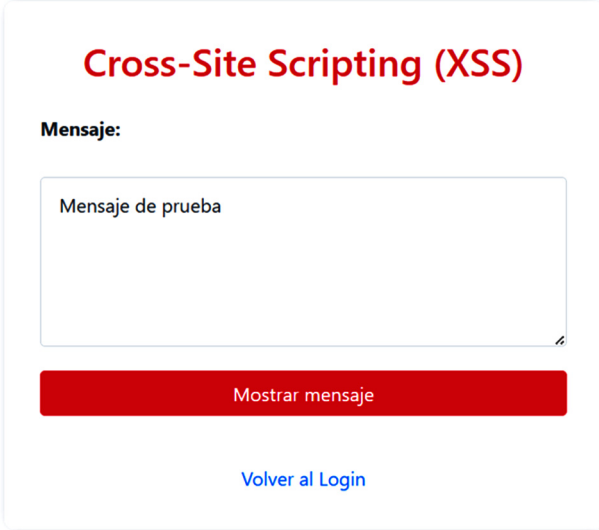

Fig. 25. Envío de mensaje de prueba.

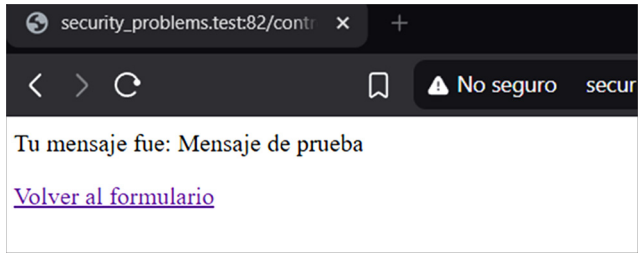

Fig. 26. Respuesta de mensaje

Fig. 27. Envio de mensaje con XSS

Se inyecta un script malicioso en el campo de mensaje para analizar el comportamiento y las acciones del servidor ante esta entrada (Fig. 28 y 29).

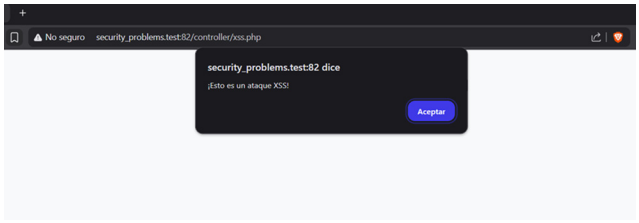

Fig. 28. Ejecucion del Script XSS.

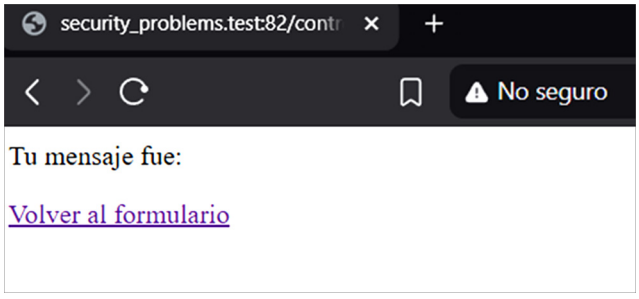

Fig. 29. Respueta de mensaje malicioso..

La ejecución del script en el servidor genera una ventana emergente con el mensaje inyectado, pero simultáneamente el sistema reporta que el mensaje enviado está vacío.

Cross-Site Request Forgery (CSRF)

Estas pruebas consisten en enviar una petición de registro simulando ser el Frontend del sistema a través de Postman, con el fin de validar que el servidor carece de un token anti-CSRF (Fig. 30).

El código de registro evidencia vulnerabilidades previamente mencionadas, como la inyección SQL (resaltada en rojo), y la ausencia de un token CSRF para mitigar ataques de este tipo.

- Falta de token CSRF: La ausencia de este token CSRF expone al sistema a peticiones no originadas en el Frontend, permitiendo que atacantes creen registros no autorizados.

```php
<?php

$servername = "127.0.0.1";
$username = "root";
$password = "";
$dbname = "security_problems";

$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) die("Conexión fallida: " . $conn->connect_error);

$user = $_POST['user'];
$password = $_POST['password'];
$email = $_POST['email'];

$sql = "INSERT INTO users (user, password, email) VALUES ('$user', '$password', '$email')";

if ($conn->multi_query($sql) === TRUE) echo "Registro exitoso";

else echo "Error: " . $sql . "<br>" . $conn->error;
```

Fig. 30. Código prueba sin token CSRF..

Pruebas al código *(Fig. 31)*

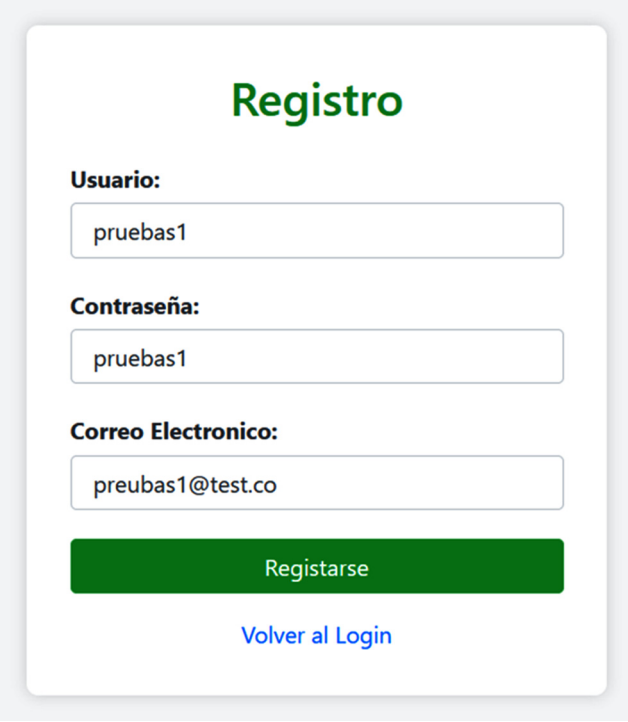

Fig. 31. Frontend de registro con datos.

Utilizando estos datos, se realizará un registro exitoso en el sistema desde el Frontend (Fig. 32 y 33).

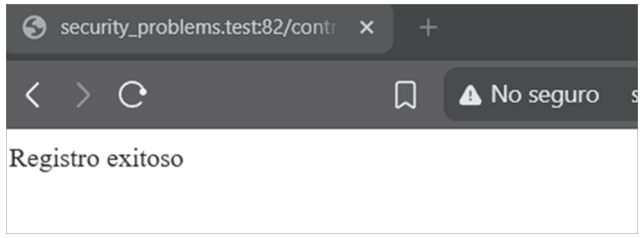

Fig. 32. Respuesta de la petición de registro.

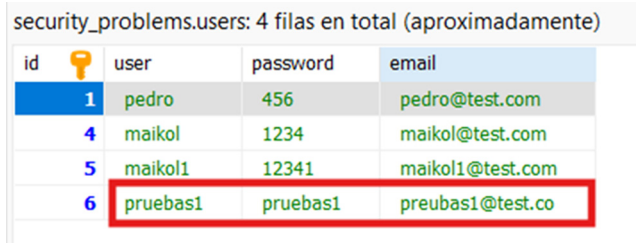

Fig. 33. Registro nuevo en la tabla.

Se evidencia el registro exitoso del usuario, el cual ha sido guardado en la base de datos 'users' (Fig. 34).

A continuación, se realizan pruebas de registro al sistema directamente desde Postman para verificar la ausencia de un token CSRF (Fig. 35 y 36).

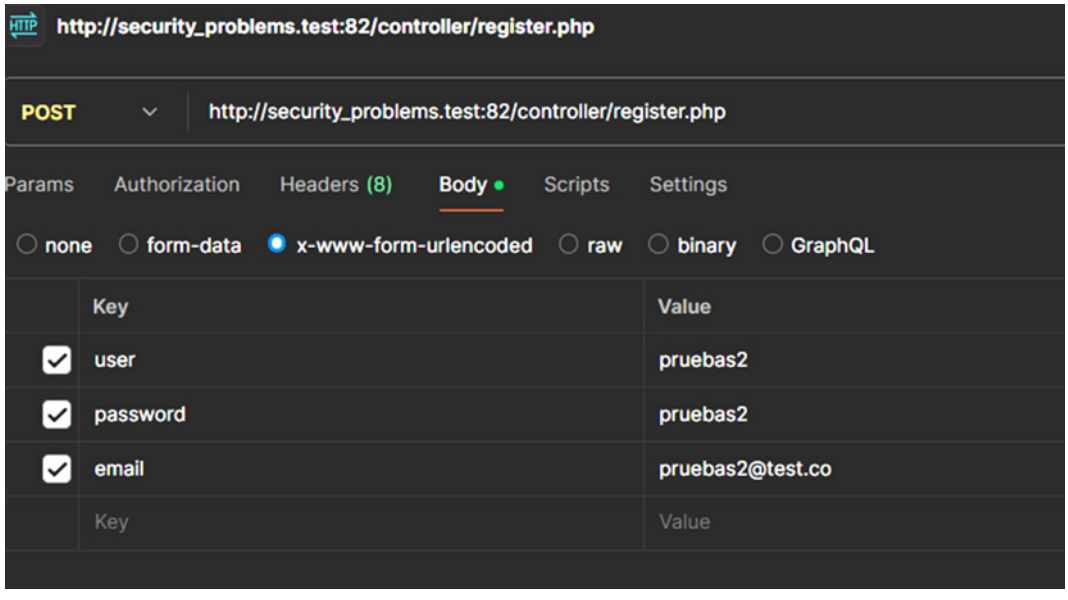

Fig. 34. Envío de datos desde postman

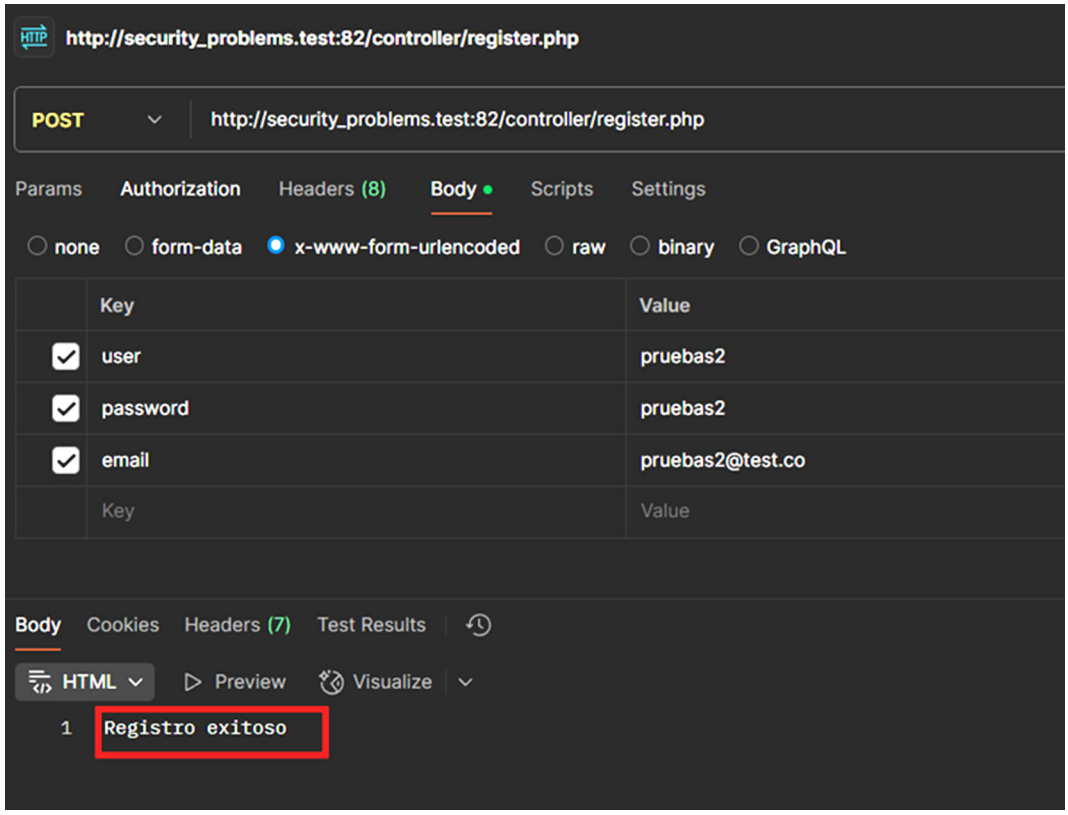

Fig. 35. Respuesta de petición desde postman.

security_problems.users: 5 filas en total (aproximadamente)

id	user	password	email
1	pedro	456	pedro@test.com
4	maikol	1234	maikol@test.com
5	maikol1	12341	maikol1@test.com
6	pruebas1	pruebas1	preubas1@test.co
7	pruebas2	pruebas2	pruebas2@test.co

Fig. 36. Registro nuevo en tabla user.

Se evidencia que el registro fue exitoso incluso desde Postman, y el usuario también fue guardado en la base de datos 'users'.

Enlace de repositorio de GitHub del proyecto de las pruebas.

https://github.com/MaikolBarrera2303/security_problems

VIII. Conclusión

El análisis de las vulnerabilidades más críticas en la programación evidencia que muchas de ellas, como la falta de validación de entradas, inyecciones de código, desbordamientos de búfer, errores en la gestión de sesiones y configuraciones inseguras, continúan representando riesgos significativos para la seguridad de los sistemas informáticos. Estas debilidades, en su mayoría se pueden prevenir con las buenas prácticas de programación y adoptando mejores procesos de gestión.

La adopción de buenas prácticas de programación, respaldadas por estándares reconocidos como OWASP y NIST, ha demostrado tener un impacto positivo en la reducción de vulnerabilidades. Implementar medidas como la codificación segura, la gestión adecuada de credenciales, el principio de menor privilegio y el monitoreo constante de configuraciones, no solo fortalece la postura de seguridad de las organizaciones, sino que también

mejora la eficiencia operativa y la confianza de los usuarios.

En este contexto, la formación continua de los desarrolladores en materia de seguridad se vuelve indispensable. La naturaleza dinámica de las amenazas requiere que los profesionales del software se mantengan actualizados con las últimas técnicas, herramientas y normativas. Fomentar una cultura de seguridad desde las etapas iniciales del desarrollo no solo mitiga riesgos, sino que constituye una inversión estratégica para la sostenibilidad y resiliencia de las organizaciones frente a los desafíos del entorno digital actual.

Las verificaciones de seguridad realizadas al sistema muestran? claramente los problemas al momento de? programar como, por ejemplo:

- Cross-Site Request Forgery (CSRF) pueden mostrar cómo la falta de validaciones de entrada de las peticiones permite que los atacantes realicen acciones no deseadas haciéndose pasar por un usuario autenticado.
- Cross-Site Scripting (XSS) muestran el peligro de no sanear la entrada del usuario, abriendo puerta a inyectar código malicioso en el sistema, dañando la información sensible y la experiencia de? los usuarios.
- Subida de archivos sin revisar cuidadoso de la extensión puede permitir la ejecución de código malicioso en el servidor.
- Inyección SQL muestran cómo una validación no exhaustiva de la entrada puede dejar a atacantes modificar las consultas a base de datos accediendo, modificando o eliminando información sensible.

Referencias

[1] M. Howard, D. LeBlanc y J. Viega, *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. New York, NY, USA: McGraw-Hill, 2021.

[2] Cybersecurity and Infrastructure Security Agency (CISA), "Securing Web Applications: Mitigating Common Vulnerabilities," U.S. Department of Homeland Security, Washington, D.C., USA, Tech. Rep. 2023.

[3] OWASP Foundation, "OWASP Top Ten 2021," Open Worldwide Application Security Project, 2021.[Online]. Available: https://owasp.org/www-project-top-ten/

[4] National Institute of Standards and Technology (NIST), *NIST Special Publication 800-53: Security and Privacy Controls for Federal Information Systems and Organizations*. Gaithersburg, MD, USA: NIST, 2020.

[5] M. Christakis and C. Bird, "What Developers Want and Need from Program Analysis: An Empirical Study," *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.*, May 2021.

[6] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Hoboken, NJ, USA: Wiley, 2021.

[7] A. J. Menezes, P. C. van Oorschot, y S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 2021.

[8] S. Garfinkel y G. Spafford, *Web Security, Privacy and Commerce*. Sebastopol, CA, USA: O'Reilly Media, 2023.

[9] Verizon, "2023 Data Breach Investigations Report", Verizon Communications, 2023.[Online]. Available: https://www.verizon.com/dbir/

[10] M. Bishop, *Computer Security: Art and Science*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2019.

[11] G. McGraw, *Software Security: Building Security In*. Boston, MA, USA: Addison-Wesley, 2021.

[12] C. Bird et al., "DevSecOps: Integrating Security in Agile Development," *IEEE Security & Privacy*, vol. 19, no. 3, pp. 45–53, May 2021.

[13] S. J. Vaughan-Nichols, "DevSecOps: Building Security into CI/CD Pipelines," *IEEE Comput.*, vol. 53, no. 9, pp. 14–19, Sep. 2022.

[14] J. Williams, "SQL Injection Attack Detection Techniques," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, no. 5, pp. 423–432, 2023.

[15] OWASP Foundation, «Cross-Site Scripting (XSS),» OWASP, 2021.[Enlace]. Available: https://owasp.org/www-community/attacks/xss.

[16] OWASP Foundation, «Cross-Site Request Forgery (CSRF),» OWASP, 2021.[Enlace]. Available: https://owasp.org/www-community/attacks/csrf.

[17] A. K. Ghosh, "Mitigating Buffer Overflow Vulnerabilities," *IEEE Softw.*, vol. 38, no. 6, pp. 60–66, Nov. 2021.

[18] S. Murtaza, «Buffer overflow attacks,» Geeksfor Geeks, 2020.[Enlace]. Available: https://www.geeksforgeeks.org/buffer-overflow-attack/.

[19] OWASP Foundation, «Using components with known vulnerabilities,» OWASP, 2021.[Enlace]. Available: https://owasp.org/www-community/Vulnerabilities/Using_Components_with_Known_Vulnerabilities.

[20] F. Wu, "Security Concerns in JWT Implementa tions," *IEEE Comput.*, vol. 55, no. 1, pp. 28–35, Jan. 2023.

[21] Kinsta. «Ataque CSRF: qué es, cómo funciona y cómo prevenirlo.» Kinsta Blog, https://kinsta.com/es/blog/ataque-csrf/. Consultado el 12 de octubre de 2024.

[22] J. Viega & G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley. 2001.

[23] OWASP Foundation, "OWASP Zed Attack Proxy (ZAP),"[Online]. Available: https://owasp.org/www-project-zap/

[24] M. Howard & D. LeBlanc, *Writing Secure Code* (2nd ed.). Microsoft Press. 2003.

[25] R. Sandhu, E. J. Coyne, H. L. Feinstein & C. E. Youman, Role-Based Access Control Models. IEEE Computer, 29(2), 38–47. https://doi.org/10.1109/2.485845 1996.

[26] B. Chess & J. West, *Secure Programming with Static Analysis*. Addison-Wesley. 2007.

[27] PortSwigger Ltd., "Burp Suite Professional," 2023.[Online]. Available: https://portswigger.net/burp

[28] Tenable, Inc., "Nessus Vulnerability Scanner," 2023.[Online]. Available: https://www.tenable.com/products/nessus

[29] SonarSource S.A., "SonarQube Documentation," 2023.[Online]. Available: https://www.sonarsource.com/products/sonarqube/

[30] CIRT.net, "Nikto Web Scanner," 2023.[Online]. Available: https://cirt.net/Nikto2

[31] Snyk Ltd., "Snyk Open Source Security," 2023. [Online]. Available: https://snyk.io/

[32] ISO/IEC, *ISO/IEC 27001:2022 - Information security, cybersecurity and privacy protection – Information security management systems – Requirements*, International Organization for Standardization, Geneva, 2022.

[33] PCI Security Standards Council, Payment Card Industry Data Security Standard – Requirements and Security Assessment Procedures, Version 4.0, Mar. 2022.[Online]. Available: https://www.pcisecuritystandards.org/

[34] European Union, *General Data Protection Regulation (GDPR)*, Regulation (EU) 2016/679, Official Journal of the European Union, Apr. 2016.

[35] ISACA, *COBIT 2019 Framework: Governance and Management Objectives*, ISACA, Schaumburg, IL, USA, 2019.