PROYECTO FINAL CRIPTOGRAFIA:

AGENDA ELECTRÓNICA PARA PACIENTES EN EL ÁREA DE LA SALUD

ENRIQUE LÓPEZ
JAVIER IBARRIA

INDICE:

ENRIQUE LÓPEZ
JAVIER IBARRIA

1.1 INTRODUCCIÓN

Importancia de la Seguridad en el Manejo de Datos Sensibles en el Área de Salud

En el mundo moderno, la información es uno de los activos más valiosos que posee una organización. En el sector de la salud, los datos sensibles de los pacientes son de suma importancia, no solo por su valor intrínseco, sino también por el impacto que puede tener su protección o divulgación indebida en la vida de las personas. La digitalización de los registros médicos y la creciente dependencia de sistemas electrónicos para gestionar esta información han traído consigo múltiples beneficios, tales como una mejor coordinación del cuidado del paciente y una mayor eficiencia operativa. Sin embargo, también ha incrementado significativamente los riesgos de seguridad.

Vulnerabilidades en los Sistemas de Salud

Los sistemas de información en salud son objetivos atractivos para los atacantes debido a la riqueza de datos personales y médicos que contienen. Las amenazas incluyen, pero no se limitan a, acceso no autorizado, robo de identidad, fraude médico, y manipulación de registros. Las violaciones de datos pueden tener consecuencias devastadoras, desde la exposición de información personal hasta el compromiso de la calidad del cuidado médico. Además, los ataques cibernéticos en este sector pueden derivar en pérdidas financieras y daños a la reputación de las instituciones involucradas.

Necesidad de Criptografía en la Protección de Datos

La criptografía es una herramienta esencial para proteger la información sensible en los sistemas de salud. Mediante técnicas criptográficas, es posible asegurar que los datos sean accesibles únicamente por personas autorizadas y que cualquier intento de acceso no autorizado resulte en información ininteligible. Dos de los métodos más importantes en criptografía son el cifrado y el hasheo.

- Cifrado: El cifrado convierte la información legible (texto plano) en una forma codificada (texto cifrado), que solo puede ser revertida a su forma original mediante una clave secreta. Esto garantiza que los datos permanezcan privados incluso si son interceptados.
- Hasheo: El hasheo transforma una entrada (como una contraseña) en un valor fijo de longitud mediante una función hash. Es particularmente útil para la verificación de contraseñas, ya que permite almacenar una representación segura de la contraseña sin tener que guardar la contraseña en sí.

Análisis Estático y Dinámico del Código Fuente

Además de la protección directa de los datos, es crucial asegurar que el código fuente del software utilizado en sistemas de salud esté libre de vulnerabilidades y errores que puedan ser explotados por atacantes. Aquí es donde entran en juego el análisis estático y el análisis dinámico del código:

- Análisis Estático: Esta técnica implica examinar el código fuente sin ejecutarlo, identificando posibles fallos, vulnerabilidades de seguridad y adherencia a las mejores prácticas de programación. Herramientas de análisis estático pueden detectar problemas como inyecciones de SQL, fallos de control de acceso y errores de lógica.
- Análisis Dinámico: A diferencia del análisis estático, el análisis dinámico evalúa el comportamiento del software en tiempo de ejecución. Este método es esencial para descubrir problemas que solo se manifiestan cuando el software se ejecuta, como problemas de memoria y condiciones de carrera.

Objetivo del Proyecto

El objetivo de este proyecto es desarrollar una agenda electrónica para la gestión de datos de pacientes en el área de salud que incorpore medidas avanzadas de seguridad mediante el uso de criptografía y análisis de código. En concreto, se implementará un sistema de autenticación seguro usando el algoritmo de hash Argon2 y se protegerán los datos de los pacientes almacenados en la

base de datos utilizando el cifrado AES en modo CBC (Cipher Block Chaining). Además, se realizarán análisis estáticos y dinámicos del código fuente para garantizar su seguridad y calidad.

Este proyecto no solo busca proteger la información sensible de los pacientes, sino también establecer un estándar de seguridad y buenas prácticas que puedan ser adoptadas en el desarrollo de sistemas de salud electrónicos.

1.2 OBJETIVOS

El objetivo principal de este proyecto es diseñar y desarrollar una agenda electrónica para la gestión segura de los datos de los pacientes en el área de salud. Para lograrlo, se emplearán técnicas avanzadas de criptografía y se realizarán análisis exhaustivos del código fuente para asegurar la calidad y la seguridad del sistema. A continuación, se detallan los objetivos específicos que guiarán el desarrollo de este proyecto:

1.2.1 Implementación de un Sistema Seguro de Autenticación usando Argon2

Descripción: La autenticación de usuarios es el primer y más crucial paso para garantizar la seguridad de un sistema. En este proyecto, se utilizará el algoritmo de hash Argon2 para la gestión segura de las contraseñas de los usuarios.

- **Hashing Seguro de Contraseñas**: Argon2 es un algoritmo de derivación de claves diseñado para ser seguro contra ataques de fuerza bruta y ataques de canal lateral. Se seleccionará Argon2id, que combina las características de Argon2d (resistente a ataques de GPU) y Argon2i (resistente a ataques de canal lateral).
- **Almacenamiento Seguro de Hashes**: Los hashes de las contraseñas generados por Argon2 se almacenarán de manera segura en la base de datos. Cada hash incluirá información sobre los parámetros utilizados (como el costo de tiempo y la memoria), permitiendo una futura verificación adecuada.
- **Verificación de Contraseñas**: Al momento del inicio de sesión, el sistema verificará las contraseñas ingresadas comparándolas con los hashes almacenados, utilizando el mismo algoritmo y parámetros.

Metas Específicas:

- Asegurar que las contraseñas de los usuarios estén protegidas contra intentos de acceso no autorizado.
- Implementar medidas para evitar ataques de fuerza bruta y ataques de diccionario.
- Garantizar que el sistema de autenticación sea eficiente y escalable.

1.2.2 Protección de Datos de Pacientes Usando AES en Modo CBC

Descripción: La protección de los datos sensibles de los pacientes es fundamental para cumplir con las regulaciones de privacidad y seguridad, como el Reglamento General de Protección de Datos (GDPR) y la Ley de Portabilidad y Responsabilidad de Seguros de Salud (HIPAA). En este proyecto, se utilizará el cifrado simétrico AES en modo CBC para asegurar los datos almacenados en la base de datos.

- **Cifrado de Datos**: Los datos de los pacientes se cifrarán utilizando el algoritmo AES (Advanced Encryption Standard) en modo CBC (Cipher Block Chaining). Este modo de operación proporciona confidencialidad adicional al encadenar bloques cifrados, asegurando que cada bloque de texto cifrado depende de todos los bloques anteriores.

- **Gestión de Claves**: Se implementará una gestión segura de las claves de cifrado, asegurando que las claves se generen, almacenen y distribuyan de manera segura. Se explorarán opciones como el uso de HSMs (Hardware Security Modules) o servicios de gestión de claves en la nube.
- **Descifrado de Datos**: Solo los usuarios autorizados podrán descifrar y acceder a los datos de los pacientes, utilizando las claves correspondientes y los vectores de inicialización (IV) asociados.

Metas Específicas:

- Asegurar que los datos de los pacientes estén protegidos contra acceso no autorizado.
- Garantizar la integridad y confidencialidad de los datos durante el almacenamiento y la transmisión.
- Cumplir con las normativas y estándares de seguridad aplicables en el sector de salud.

1.2.3 Análisis Estático y Dinámico del Código Fuente

Descripción: Para asegurar la calidad y seguridad del código fuente del proyecto, se realizarán análisis estáticos y dinámicos exhaustivos. Estos análisis identificarán posibles vulnerabilidades y errores en el código, permitiendo su corrección antes de la implementación final.

- **Análisis Estático**: Se utilizarán herramientas como SonarQube para analizar el código fuente sin ejecutarlo. Este análisis identificará problemas como vulnerabilidades de seguridad, errores de lógica, y desviaciones de las mejores prácticas de programación.
- **Análisis Dinámico**: Se emplearán herramientas para evaluar el comportamiento del sistema en tiempo de ejecución. Esto incluye la detección de problemas de memoria, condiciones de carrera, y otros errores que solo se manifiestan durante la ejecución.
- **Pruebas de Penetración**: Se realizarán pruebas de penetración para simular ataques reales y evaluar la resistencia del sistema contra intrusiones y vulnerabilidades.

Metas Específicas:

- Identificar y corregir vulnerabilidades de seguridad en el código fuente.
- Mejorar la calidad del código mediante la adopción de las mejores prácticas de programación.
- Asegurar que el sistema sea robusto y resistente a ataques y fallos.

Resultados Esperados

Al finalizar este proyecto, se espera lograr un sistema de gestión de datos de pacientes que cumpla con los más altos estándares de seguridad. Específicamente, los resultados esperados incluyen:

- Un sistema de autenticación robusto que proteja las cuentas de los usuarios utilizando el algoritmo Argon2.
- Datos de pacientes cifrados de manera segura con AES en modo CBC, asegurando su confidencialidad e integridad.
- Un código fuente de alta calidad, libre de vulnerabilidades críticas, gracias a un análisis estático y dinámico exhaustivo.

Conclusión

El desarrollo de una agenda electrónica segura para la gestión de datos de pacientes en el área de salud es un proyecto de gran relevancia y complejidad. La implementación de técnicas avanzadas de criptografía y la realización de análisis detallados del código son esenciales para garantizar que el sistema cumpla con las normativas de seguridad y proteja la información sensible de los pacientes. Este proyecto no solo tiene como objetivo crear un producto funcional, sino también establecer un modelo de buenas prácticas y estándares de seguridad que puedan ser adoptados en el desarrollo de futuros sistemas de información en salud.

ENRIQUE LÓPEZ
JAVIER IBARRIA

2. TECNOLOGÍAS Y HERRAMIENTAS

2.1 Tecnologías

En este apartado, se detallan las tecnologías que se utilizarán en el desarrollo de la agenda electrónica segura para la gestión de datos de pacientes. La selección de tecnologías adecuadas es crucial para garantizar la seguridad, escalabilidad y eficiencia del sistema.

2.1.1 Lenguajes de Programación

PYTHON:

- **Razones para elegir Python**: Python es un lenguaje de programación de alto nivel conocido por su simplicidad y legibilidad, lo que facilita el desarrollo rápido y la mantenibilidad del código. Además, cuenta con una amplia gama de bibliotecas y frameworks que soportan el desarrollo de aplicaciones seguras.
- **Bibliotecas relevantes**:
  - argon2-cffi: Para implementar el algoritmo de hash Argon2.
  - pycryptodome: Para operaciones criptográficas, incluyendo AES en modo CBC.
  - flask o django: Para el desarrollo del backend del sistema, manejando las solicitudes HTTP y la lógica del servidor.

JAVA:

- **Razones para elegir Java**: Java es un lenguaje de programación robusto y multiplataforma, ampliamente utilizado en aplicaciones empresariales. Ofrece una sólida seguridad y rendimiento.
- **Bibliotecas relevantes**:
  - Bouncy Castle: Un proveedor de servicios criptográficos que incluye implementaciones de Argon2 y AES.
  - Spring Framework: Para desarrollar aplicaciones web seguras y escalables.

C#:

- **Razones para elegir C#**: C# es un lenguaje desarrollado por Microsoft, ideal para aplicaciones en entornos Windows. Es conocido por su potencia y soporte para el desarrollo de aplicaciones empresariales.
- **Bibliotecas relevantes**:
  - BouncyCastle.Crypto: Una implementación de Bouncy Castle para C#.
  - ASP.NET Core: Para el desarrollo de aplicaciones web seguras y eficientes.

2.1.2 Frameworks y Bibliotecas Criptográficas

PyCryptodome (Python):

- Proporciona herramientas para realizar operaciones criptográficas básicas y avanzadas, incluyendo el cifrado AES en modo CBC.
- Ejemplo de uso:

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
```

Bouncy Castle (Java y C#):

- Una biblioteca de criptografía que proporciona una amplia gama de algoritmos y funcionalidades criptográficas.
- Ejemplo de uso en Java:

```java
import org.bouncycastle.crypto.generators.Argon2BytesGenerator;
import org.bouncycastle.crypto.params.Argon2Parameters;
```

Argon2-cffi (Python):

- Implementación del algoritmo Argon2 para el hasheo de contraseñas.
- Ejemplo de uso:

```python
from argon2 import PasswordHasher
ph = PasswordHasher()
hash = ph.hash("contraseña_secreta")
```

2.1.3 Bases de Datos

MySQL:

- **Características**: MySQL es un sistema de gestión de bases de datos relacional muy popular, conocido por su rendimiento y fiabilidad. Es adecuado para almacenar grandes volúmenes de datos de pacientes de manera segura.
- **Seguridad**: Soporta cifrado nativo de tablas y conexiones seguras (SSL/TLS).

PostgreSQL:

- **Características**: PostgreSQL es un sistema de gestión de bases de datos objeto-relacional avanzado, conocido por su extensibilidad y soporte para transacciones ACID.
- **Seguridad**: Ofrece características avanzadas de seguridad, incluyendo cifrado a nivel de columna y autenticación mediante certificados SSL.

MongoDB:

- **Características**: MongoDB es una base de datos NoSQL que almacena datos en formato de documento (JSON). Es altamente escalable y flexible.
- **Seguridad**: Proporciona cifrado en reposo y en tránsito, y soporte para autenticación y autorización avanzadas.

2.1.4 Infraestructura y Entorno de Desarrollo

Docker:

- **Razones para usar Docker**: Docker permite la creación de entornos de desarrollo aislados y replicables mediante contenedores. Facilita la gestión de dependencias y asegura que el software se ejecute de manera consistente en diferentes entornos.
- **Ejemplo**: Crear un contenedor para el backend de la agenda electrónica con todas las dependencias necesarias.

Kubernetes:

- **Razones para usar Kubernetes**: Kubernetes es una plataforma de orquestación de contenedores que facilita el despliegue, escalado y gestión de aplicaciones en contenedores. Es ideal para aplicaciones empresariales que requieren alta disponibilidad y escalabilidad.
- **Ejemplo**: Desplegar la aplicación de la agenda electrónica en un clúster de Kubernetes para manejar cargas de trabajo de manera eficiente.

Entorno de Desarrollo Integrado (IDE):

- **Visual Studio Code (VSCode)**: Un editor de código fuente altamente personalizable y extensible, con soporte para múltiples lenguajes de programación y herramientas de desarrollo.
- **IntelliJ IDEA**: Un IDE poderoso y popular para el desarrollo en Java, con características avanzadas de depuración y análisis de código.
- **Visual Studio**: Un IDE completo para el desarrollo en C# y otras tecnologías de Microsoft, con un fuerte enfoque en la productividad del desarrollador.

2.2 HERRAMIENTAS

2.2.1 Herramientas de Análisis Estático

SonarQube:

- **Descripción**: SonarQube es una plataforma de código abierto para la inspección continua de la calidad del código. Analiza el código fuente y detecta bugs, vulnerabilidades, y problemas de mantenibilidad.
- **Características**: Soporte para múltiples lenguajes de programación, integración con CI/CD, métricas detalladas y reportes.
- **Ejemplo de uso**:
  - Integración con un pipeline de Jenkins para análisis automatizado del código en cada commit.
  - Configuración de reglas de calidad y perfiles de análisis específicos para el proyecto.

Bandit (Python):

- **Descripción**: Bandit es una herramienta de análisis estático que encuentra vulnerabilidades de seguridad en el código Python.
- **Ejemplo de uso**:

bash Copiar código

```
bandit -r my_project/
```

2.2.2 Herramientas de Análisis Dinámico

Valgrind:

- **Descripción**: Valgrind es un conjunto de herramientas para la depuración y el análisis de rendimiento de programas. Es especialmente útil para detectar errores de memoria.
- **Ejemplo de uso**:

bash Copiar código

```
valgrind --leak-check=full ./my_program
```

OWASP ZAP (Zed Attack Proxy):

- **Descripción**: OWASP ZAP es una herramienta de pruebas de penetración para encontrar vulnerabilidades de seguridad en aplicaciones web.
- **Características**: Escaneo automatizado, proxies de intercepción, pruebas manuales avanzadas.
- **Ejemplo de uso:**
  - Configuración para escanear la aplicación web de la agenda electrónica y generar un informe de vulnerabilidades.

Burp Suite:

- **Descripción**: Burp Suite es una plataforma integrada para la realización de pruebas de seguridad en aplicaciones web.
- **Características**: Escáner de vulnerabilidades, herramientas de manipulación de solicitudes, análisis automatizado y manual.
- **Ejemplo de uso**:
  - Configuración para interceptar y analizar las solicitudes HTTP en la aplicación de la agenda electrónica.

Resumen de Tecnologías y Herramientas

La combinación de tecnologías robustas y herramientas avanzadas asegurará que la agenda electrónica sea segura, escalable y mantenible. La elección de lenguajes de programación como Python, Java y C#, junto con frameworks y bibliotecas criptográficas adecuadas, permitirá implementar funcionalidades seguras y eficientes. Las bases de datos como MySQL, PostgreSQL y MongoDB garantizarán el almacenamiento seguro de los datos de los pacientes. Además, el uso de herramientas de análisis estático y dinámico como SonarQube, Valgrind, OWASP ZAP y Burp Suite asegurará la calidad y seguridad del código fuente.

3. DISEÑO DEL SISTEMA

En esta sección, se detalla el diseño del sistema de la agenda electrónica para la gestión segura de datos de pacientes. El diseño abarca tanto la arquitectura general del sistema como los componentes específicos que interactúan para cumplir con los objetivos de seguridad y funcionalidad.

3.1 Arquitectura del Sistema

La arquitectura del sistema es esencial para definir cómo se organizarán y se comunicarán los diferentes componentes. Para este proyecto, se utilizará una arquitectura de múltiples capas para asegurar la separación de preocupaciones, escalabilidad y seguridad.

3.1.1 Visión General de la Arquitectura

La arquitectura del sistema se divide en las siguientes capas principales:

1. Capa de Presentación (Frontend)
2. Capa de Lógica de Negocio (Backend)
3. Capa de Persistencia (Base de Datos)
4. Capa de Seguridad

3.1.2 Capa de Presentación (Frontend)

Descripción: La capa de presentación es la interfaz de usuario de la agenda electrónica. Esta capa se encarga de interactuar con los usuarios finales (personal médico, administrativos, etc.) y proporcionar una experiencia de usuario intuitiva y segura.

- **Tecnologías**: HTML, CSS, JavaScript, React.js (o Angular/Vue.js)
- **Componentes**:
  - Formularios de inicio de sesión
  - Formularios para la gestión de datos de pacientes
  - Visualización de datos y reportes

Seguridad en la Capa de Presentación:

- Uso de HTTPS para asegurar las comunicaciones entre el cliente y el servidor.
- Validación y sanitización de entradas para prevenir ataques de inyección.
- Implementación de medidas de protección contra Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF).

3.1.3 Capa de Lógica de Negocio (Backend)

Descripción: La capa de lógica de negocio maneja la lógica de la aplicación, incluyendo la autenticación, la gestión de datos de pacientes y la implementación de algoritmos criptográficos.

- **Tecnologías**: Python (Flask/Django), Java (Spring Boot), C# (ASP.NET Core)
- **Componentes**:
  - Controladores para manejar solicitudes HTTP
  - Servicios para la lógica de negocio (autenticación, cifrado/descifrado de datos)
  - Integración con la base de datos

Seguridad en la Capa de Lógica de Negocio:

- Implementación de autenticación y autorización robustas.
- Uso de algoritmos seguros de hash y cifrado (Argon2 para autenticación, AES-CBC para cifrado de datos).
- Verificación de permisos y roles de usuario para el acceso a los datos.

3.1.4 Capa de Persistencia (Base de Datos)

Descripción: La capa de persistencia almacena de manera segura los datos de los pacientes y otra información relevante del sistema.

- **Tecnologías**: MySQL, PostgreSQL, MongoDB
- **Componentes**:
  - Tablas o colecciones para almacenar datos de pacientes, usuarios, registros de actividad, etc.
  - Procedimientos almacenados y triggers para operaciones complejas y auditoría

Seguridad en la Capa de Persistencia:

- Cifrado de datos en reposo utilizando AES.
- Restricciones de acceso a la base de datos mediante roles y permisos.
- Auditoría y registro de accesos y modificaciones a los datos sensibles.

3.1.5 Capa de Seguridad

Descripción: La capa de seguridad se integra en todas las capas anteriores para asegurar que el sistema cumpla con los más altos estándares de seguridad.

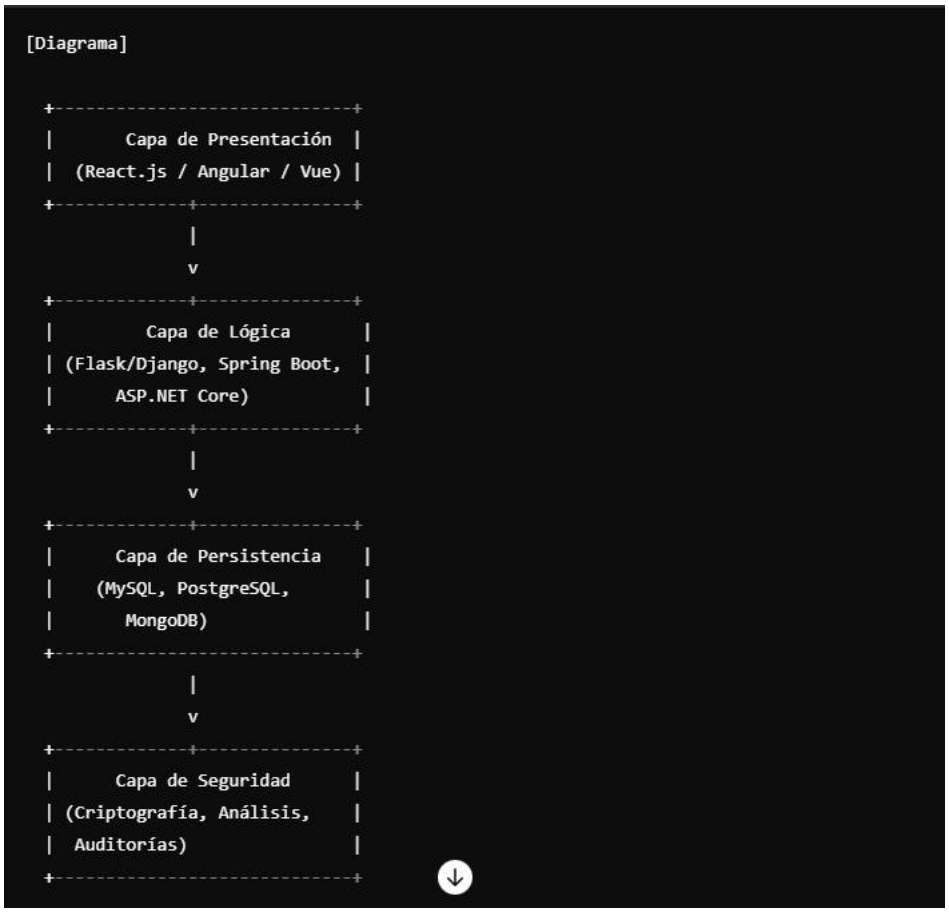
- **Tecnologías**: Herramientas y bibliotecas de criptografía (PyCryptodome, Bouncy Castle), herramientas de análisis de seguridad (SonarQube, OWASP ZAP, Burp Suite)
- **Componentes**:
  - Implementación de cifrado y hash de datos
  - Análisis estático y dinámico del código
  - Pruebas de penetración y auditorías de seguridad

Medidas de Seguridad Específicas:

- Autenticación de usuarios mediante Argon2 para el hash de contraseñas.
- Cifrado de datos sensibles con AES en modo CBC.
- Uso de HTTPS para todas las comunicaciones.
- Análisis estático y dinámico del código para identificar y mitigar vulnerabilidades.
- Pruebas de penetración regulares para evaluar y mejorar la seguridad del sistema.

3.2 Diagrama de Arquitectura

Para ilustrar la arquitectura del sistema, se presentará un diagrama que muestra cómo interactúan las diferentes capas y componentes.



3.3 Interacción entre Componentes

Para asegurar que los diferentes componentes del sistema interactúen de manera segura y eficiente, se detallarán los flujos de datos y las interacciones principales.

3.3.1 Flujo de Autenticación

1. **Inicio de Sesión del Usuario**:

- El usuario ingresa sus credenciales en el frontend.
- Las credenciales se envían al backend mediante una solicitud HTTPS.
- El backend utiliza Argon2 para verificar la contraseña ingresada contra el hash almacenado en la base de datos.
- Si las credenciales son correctas, se genera un token de autenticación (por ejemplo, JWT) y se envía al frontend.

2. **Acceso a Datos de Pacientes**:

- El usuario autenticado solicita datos de pacientes.
- El backend verifica el token de autenticación y los permisos del usuario.
- Los datos solicitados se cifran utilizando AES antes de ser almacenados o transmitidos.

3.3.2 Gestión de Datos de Pacientes

1. **Almacenamiento de Datos**:

- El usuario ingresa datos de pacientes en el frontend.
- Los datos se envían al backend mediante una solicitud HTTPS.
- El backend cifra los datos utilizando AES en modo CBC.
- Los datos cifrados se almacenan en la base de datos.

2. **Recuperación de Datos**:

- El usuario solicita la visualización o modificación de datos de pacientes.
- El backend recupera los datos cifrados de la base de datos.
- Los datos se descifran utilizando la clave AES correspondiente.
- Los datos descifrados se envían al frontend de manera segura.

3.3.3 Análisis de Seguridad

1. **Análisis Estático del Código**:

- El código fuente se analiza mediante herramientas como SonarQube para identificar vulnerabilidades y problemas de calidad.
- Los resultados del análisis se utilizan para mejorar el código antes del despliegue.

2. **Análisis Dinámico del Código**:

- El comportamiento del sistema se evalúa en tiempo de ejecución utilizando herramientas como Valgrind y OWASP ZAP.
- Se identifican y corrigen problemas que solo se manifiestan durante la ejecución.

3. Pruebas de Penetración:

- Se realizan pruebas de penetración regulares para simular ataques reales y evaluar la resistencia del sistema.
- Los hallazgos de las pruebas de penetración se utilizan para fortalecer las medidas de seguridad.

Conclusión

El diseño del sistema para la agenda electrónica de gestión de datos de pacientes integra múltiples capas y tecnologías para asegurar la confidencialidad, integridad y disponibilidad de los datos. La arquitectura modular permite una clara separación de preocupaciones, facilitando la implementación, mantenimiento y escalabilidad del sistema. Las medidas de seguridad integradas en todas las capas garantizan que los datos sensibles de los pacientes estén protegidos contra accesos no autorizados y amenazas potenciales.

ENRIQUE LÓPEZ
JAVIER IBARRIA

3.4 ARQUITECTURA DE SEGURIDAD

La arquitectura de seguridad del sistema se diseña para proteger los datos sensibles de los pacientes, garantizar la integridad del sistema y prevenir accesos no autorizados. Se enfoca en implementar medidas de seguridad en todas las capas del sistema, desde la capa de presentación hasta la base de datos y la infraestructura subyacente.

3.4.1 Principios de Seguridad

Para construir una arquitectura de seguridad robusta, se seguirán los siguientes principios fundamentales:

- **Defensa en Profundidad**: Implementar múltiples capas de seguridad para proteger los datos y el sistema.
- **Principio del Mínimo Privilegio**: Los usuarios y procesos deben tener solo los permisos necesarios para realizar sus funciones.
- **Autenticación y Autorización Fuertes**: Utilizar mecanismos robustos para verificar la identidad de los usuarios y controlar el acceso a los recursos.
- **Cifrado de Datos**: Proteger los datos en tránsito y en reposo mediante cifrado.
- **Monitoreo y Auditoría**: Registrar y monitorear actividades para detectar y responder a incidentes de seguridad.

3.4.2 Autenticación y Autorización

Autenticación:

La autenticación asegura que solo los usuarios autorizados puedan acceder al sistema. Se implementarán las siguientes medidas:

- **Hash de Contraseñas con Argon2**:

  - Argon2 es un algoritmo de derivación de claves resistente a ataques de fuerza bruta y de canal lateral.
  - Las contraseñas de los usuarios se almacenan como hashes generados por Argon2.
  - Ejemplo de uso en Python:

```python
from argon2 import PasswordHasher
ph = PasswordHasher()
hash = ph.hash("contraseña_secreta")
```

- **Autenticación de Dos Factores (2FA)**:

  - Implementar un segundo factor de autenticación para mejorar la seguridad de las cuentas de los usuarios.
  - Utilizar aplicaciones de autenticación como Google Authenticator o mensajes SMS.

Autorización:

La autorización controla el acceso de los usuarios autenticados a los recursos del sistema según sus roles y permisos.

- **Control de Acceso Basado en Roles (RBAC)**:

  - Definir roles (ej. administrador, médico, recepcionista) y asignar permisos específicos a cada rol.
  - Ejemplo de configuración en Django:

```python
from django.contrib.auth.models import Group, Permission

admin_group, created = Group.objects.get_or_create(name='admin')
view_permission = Permission.objects.get(codename='view_patient')
admin_group.permissions.add(view_permission)
```

- **Verificación de Permisos**:

  - Cada solicitud al backend verifica los permisos del usuario antes de permitir el acceso a los datos o funcionalidades.

3.4.3 Cifrado de Datos

Cifrado en Tránsito:

- **HTTPS**:

  - Utilizar HTTPS para todas las comunicaciones entre el cliente y el servidor, protegiendo los datos contra intercepciones y ataques de hombre en el medio (MITM).
  - Configuración de certificados SSL/TLS en el servidor web (ej. Nginx, Apache).

Cifrado en Reposo:

- **AES en Modo CBC**:

  - Utilizar el algoritmo de cifrado simétrico AES en modo CBC para cifrar los datos sensibles de los pacientes antes de almacenarlos en la base de datos.
  -Ejemplo de uso en Python:

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

key = get_random_bytes(32)  # Clave de 256 bits
cipher = AES.new(key, AES.MODE_CBC)
ciphertext = cipher.encrypt(pad(data, AES.block_size))
```

- **Gestión de Claves**:

  - Las claves de cifrado deben ser almacenadas y gestionadas de manera segura, utilizando servicios de gestión de claves (KMS) como AWS KMS o Azure Key Vault.

3.4.4 Monitoreo y Auditoría

Registro de Actividades:

- **Logs de Seguridad**:

  - Registrar todas las actividades relevantes de seguridad, como intentos de inicio de sesión, accesos a datos sensibles y modificaciones en la configuración del sistema.
  - Utilizar frameworks de logging como `logging` en Python:

```python
import logging
logging.basicConfig(filename='security.log', level=logging.INFO)
logging.info('Intento de inicio de sesión exitoso para el usuario: %s', username)
```

- **Monitoreo en Tiempo Real**:

  - Implementar soluciones de monitoreo en tiempo real para detectar actividades sospechosas y responder rápidamente a incidentes de seguridad.
  - Utilizar herramientas como ELK Stack (Elasticsearch, Logstash, Kibana) para análisis y visualización de logs.

3.4.5 Análisis de Seguridad

Análisis Estático y Dinámico del Código:

- **SonarQube**:
  - Analizar el código fuente para identificar vulnerabilidades y problemas de calidad.
  - Configuración de SonarQube en un pipeline de CI/CD para análisis continuo.

- **Bandit**:
  - Herramienta específica para encontrar vulnerabilidades de seguridad en el código Python.
  - Ejemplo de uso:

```bash
bandit -r my_project/
```

Pruebas de Penetración:

- OWASP ZAP y Burp Suite:
  - Realizar pruebas de penetración para identificar vulnerabilidades en la aplicación web.
  - Configuración de OWASP ZAP para escanear la aplicación de manera automatizada y realizar pruebas manuales avanzadas.

- Evaluaciones de Seguridad Periódicas:
  - Realizar evaluaciones de seguridad periódicas para mantener y mejorar la postura de seguridad del sistema.
  - Documentar los hallazgos y las medidas correctivas implementadas.

Conclusión

La arquitectura de seguridad propuesta para la agenda electrónica asegura una protección integral de los datos de los pacientes mediante la implementación de medidas robustas en todas las capas del sistema. La autenticación y autorización fuertes, el cifrado de datos en tránsito y en reposo, y el monitoreo continuo son componentes clave para garantizar la confidencialidad, integridad y disponibilidad de la información. El análisis estático y dinámico del código, junto con las pruebas de penetración regulares, permiten identificar y mitigar vulnerabilidades, asegurando un sistema seguro y confiable.

ENRIQUE LÓPEZ
JAVIER IBARRIA

4. IMPLEMENTACIÓN DEL SISTEMA

En esta sección, se detalla el proceso de implementación del sistema de la agenda electrónica para la gestión de datos de pacientes. Esta implementación sigue la arquitectura previamente diseñada y aplica las medidas de seguridad discutidas. La implementación abarca desde la configuración del entorno hasta el desarrollo y despliegue del sistema.

4.1 Configuración del Entorno

Antes de comenzar con el desarrollo, es crucial configurar el entorno de desarrollo de manera adecuada para asegurar que todos los componentes funcionen en conjunto y se puedan integrar fácilmente.

4.1.1 Entorno de Desarrollo

Requisitos del Sistema:

- Sistema Operativo: Windows, macOS, o Linux
- Lenguajes y Frameworks: Python (Flask/Django), JavaScript (React.js/Angular/Vue.js)
- Base de Datos: MySQL, PostgreSQL, o MongoDB
- Herramientas de Seguridad: Argon2, PyCryptodome
- Herramientas de Análisis: SonarQube, Bandit, OWASP ZAP

Instalación de Dependencias:

- Python y Paquetes Necesarios:

```bash
pip install flask argon2-cffi pycryptodome
```

- Node.js y Paquetes Necesarios:

```bash
npm install react react-dom
```

- Base de Datos:

- Instalar y configurar MySQL/PostgreSQL/MongoDB según la preferencia del sistema.

Configuración del Control de Versiones:

- Git:

- Inicializar un repositorio Git y configurar un flujo de trabajo adecuado para el equipo de desarrollo.

```bash
git init
git remote add origin <repositorio_url>
```

4.2 Desarrollo del Sistema

4.2.1 Desarrollo del Backend

El backend maneja la lógica de negocio, la autenticación, el cifrado de datos y la comunicación con la base de datos.

Configuración Inicial:

- Crear un entorno virtual y una estructura de directorios.

```bash
mkdir health_agenda
cd health_agenda
python -m venv venv
source venv/bin/activate
```

- Crear un archivo de configuración para el proyecto (config.py) con las variables de entorno necesarias.

```python
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-guess'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or 'sqlite:///site.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

Autenticación y Autorización:

- Implementar el registro y login de usuarios con Argon2 para el hash de contraseñas.

```python
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from argon2 import PasswordHasher

app = Flask(__name__)
app.config.from_object(Config)
db = SQLAlchemy(app)
ph = PasswordHasher()

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    password = db.Column(db.String(150), nullable=False)

@app.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    hashed_password = ph.hash(data['password'])
    new_user = User(username=data['username'], password=hashed_password)
    db.session.add(new_user)
    db.session.commit()
    return jsonify({'message': 'User registered successfully'})

@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    user = User.query.filter_by(username=data['username']).first()
    if user and ph.verify(user.password, data['password']):
        return jsonify({'message': 'Login successful'})
    else:
        return jsonify({'message': 'Invalid credentials'}), 401
```

Cifrado de Datos:

- Implementar el cifrado de datos de pacientes con AES en modo CBC.

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

class Patient(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(150), nullable=False)
    encrypted_data = db.Column(db.LargeBinary, nullable=False)
    iv = db.Column(db.LargeBinary, nullable=False)

def encrypt_data(data):
    key = get_random_bytes(32)
    cipher = AES.new(key, AES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(data.encode(), AES.block_size))
    return key, cipher.iv, ct_bytes

def decrypt_data(key, iv, ct):
    cipher = AES.new(key, AES.MODE_CBC, iv=iv)
    pt = unpad(cipher.decrypt(ct), AES.block_size)
    return pt.decode()

@app.route('/add_patient', methods=['POST'])
def add_patient():
    data = request.get_json()
    key, iv, encrypted_data = encrypt_data(data['data'])
    new_patient = Patient(name=data['name'], encrypted_data=encrypted_data, iv=iv)
    db.session.add(new_patient)
    db.session.commit()
    return jsonify({'message': 'Patient added successfully'})
```

4.2.2 Desarrollo del Frontend

El frontend proporciona la interfaz de usuario para interactuar con el sistema.

Configuración Inicial:

- Crear una aplicación React.js.

bash Copiar código

```
npx create-react-app health-agenda-frontend
cd health-agenda-frontend
```

Componentes de Autenticación:

- Crear componentes para el registro e inicio de sesión.

jsx Copiar código Copiar código

```jsx
// src/components/Login.js
import React, { useState } from 'react';
import axios from 'axios';

function Login() {
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');

    const handleSubmit = (e) => {
        e.preventDefault();
        axios.post('/login', { username, password })
            .then(response => {
                console.log(response.data);
            })
            .catch(error => {
                console.error('There was an error logging in!', error);
            });
    };

    return (
        <form onSubmit={handleSubmit}>
            <input type="text" value={username} onChange={(e) => setUsername(e.target.value)} placeholder="Username" required />
            <input type="password" value={password} onChange={(e) => setPassword(e.target.value)} placeholder="Password" required />
            <button type="submit">Login</button>
        </form>
    );
}

export default Login;
```

Componentes de Gestión de Pacientes:

- Crear componentes para añadir y visualizar datos de pacientes.

jsx Copiar código Copiar código

```jsx
// src/components/AddPatient.js
import React, { useState } from 'react';
import axios from 'axios';

function AddPatient() {
    const [name, setName] = useState('');
    const [data, setData] = useState('');

    const handleSubmit = (e) => {
        e.preventDefault();
        axios.post('/add_patient', { name, data })
            .then(response => {
                console.log(response.data);
            })
            .catch(error => {
                console.error('There was an error adding the patient!', error);
            });
    };

    return (
        <form onSubmit={handleSubmit}>
            <input type="text" value={name} onChange={(e) => setName(e.target.value)} placeholder="Patient Name" required />
            <textarea value={data} onChange={(e) => setData(e.target.value)} placeholder="Patient Data" required></textarea>
            <button type="submit">Add Patient</button>
        </form>
    );
}

export default AddPatient;
```

ENRIQUE LÓPEZ
JAVIER IBARRIA

4.2.3 Conexión Backend-Frontend

- Configurar el frontend para comunicarse con el backend utilizando rutas relativas y configurar proxies si es necesario.

```js
// package.json (React app)
"proxy": "http://localhost:5000"
```

4.2.4 Despliegue

Preparación para el Despliegue:

- Backend:

- Crear un archivo `Dockerfile` para contenedores.

```
FROM python:3.8-slim-buster
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

- Crear un archivo `docker-compose.yml` para la orquestación de contenedores.

```yaml
version: '3.8'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    environment:
      - SECRET_KEY=your_secret_key
      - DATABASE_URL=sqlite:///site.db
```

- Frontend:

- Crear un archivo `Dockerfile` para contenedores.

Dockerfile Copiar código

```
FROM node:14
WORKDIR /app
COPY . /app
RUN npm install
RUN npm run build
```

- Despliegue en Producción:

- Utilizar servicios de alojamiento como AWS, Azure, Heroku o DigitalOcean.
- Configurar CI/CD con GitHub Actions, GitLab CI, Jenkins, etc., para automatizar el despliegue.

Conclusión

La implementación del sistema sigue una metodología estructurada y segura, integrando técnicas avanzadas de seguridad y cifrado, así como prácticas de desarrollo modernas. Desde la configuración inicial del entorno hasta el despliegue en producción, se asegura que el sistema de la agenda electrónica sea robusto, seguro y escalable.

5. ANÁLISIS DE SEGURIDAD

El análisis de seguridad es una parte crucial para garantizar que el sistema de la agenda electrónica sea seguro y cumpla con los estándares de protección de datos sensibles. Este análisis abarca la revisión del código fuente, pruebas de vulnerabilidades y la implementación de controles para detectar y mitigar amenazas potenciales.

5.1 Análisis Estático del Código

El análisis estático del código implica revisar el código fuente sin ejecutarlo, utilizando herramientas automáticas para identificar posibles vulnerabilidades y errores de seguridad.

Herramientas Utilizadas:

- SonarQube: Para análisis de calidad y seguridad del código.
- Bandit: Para encontrar vulnerabilidades de seguridad específicas en el código Python.

Configuración de SonarQube:

1. Instalación:

- Descargar e instalar SonarQube en el servidor o utilizar la versión en la nube.

2. Integración con el Proyecto:

- Configurar SonarQube para analizar el proyecto mediante un archivo `sonar-project.properties`.

```properties
sonar.projectKey=health_agenda
sonar.sources=.
sonar.python.version=3.8
```

3. Ejecución del Análisis:

- Ejecutar el análisis con el scanner de SonarQube.

```bash
sonar-scanner
```

Configuración de Bandit:

1. Instalación:

- Instalar Bandit mediante pip.

```bash
pip install bandit
```

2. Ejecución del Análisis:

- Ejecutar Bandit para analizar el código Python.

```bash
bandit -r /path/to/project
```

3. Revisión de Resultados:

- Revisar los informes generados por Bandit y SonarQube para identificar y corregir vulnerabilidades y errores de codificación.

5.2 Análisis Dinámico del Código

El análisis dinámico del código implica ejecutar la aplicación y monitorear su comportamiento en tiempo real para identificar vulnerabilidades de seguridad.

Herramientas Utilizadas:

- OWASP ZAP: Para pruebas de penetración automatizadas y manuales.
- Burp Suite: Para pruebas avanzadas de seguridad web.

Configuración y Uso de OWASP ZAP:

1. Instalación:

   - Descargar e instalar OWASP ZAP desde su sitio oficial.

2. Configuración:

   - Configurar OWASP ZAP para interceptar y analizar el tráfico de la aplicación.

3. Ejecución de Pruebas:

   - Realizar un escaneo automatizado de la aplicación.
   - Iniciar un nuevo escaneo y configurar el objetivo (URL de la aplicación).
   - Realizar pruebas manuales para identificar vulnerabilidades específicas.
   - Usar herramientas como el "Active Scanner" y "Spider" para explorar el sitio.

Configuración y Uso de Burp Suite:

1. Instalación:

   - Descargar e instalar Burp Suite desde su sitio oficial.

2. Configuración:

   - Configurar Burp Suite para interceptar y analizar el tráfico de la aplicación.

3. Ejecución de Pruebas:

   - Realizar un escaneo automatizado y manual de la aplicación.
   - Usar "Intruder" para realizar pruebas de fuerza bruta y "Repeater" para repetir y modificar solicitudes HTTP.

4. Revisión de Resultados:

   - Analizar los informes generados por OWASP ZAP y Burp Suite para identificar vulnerabilidades como inyecciones SQL, Cross-Site Scripting (XSS), y problemas de autenticación/autorización.

5.3 Pruebas de Penetración

Las pruebas de penetración son evaluaciones de seguridad exhaustivas que simulan ataques reales para identificar y corregir vulnerabilidades antes de que puedan ser explotadas por atacantes.

Fases de las Pruebas de Penetración:

1. Reconocimiento:

   - Recopilar información sobre la aplicación y su infraestructura.
   - Utilizar herramientas como Nmap para escanear puertos y servicios.

2. Escaneo y Enumeración:

   - Identificar vulnerabilidades en los servicios y aplicaciones.
   - Utilizar herramientas como Nessus o OpenVAS para escaneo de vulnerabilidades.

3. Explotación:

- Intentar explotar las vulnerabilidades identificadas para evaluar el impacto real.
- Utilizar herramientas como Metasploit para pruebas de explotación.

4. Post-explotación:

- Evaluar el acceso obtenido y determinar el alcance del compromiso.
- Identificar posibles rutas para escalar privilegios y acceso a datos sensibles.

5. Reporte y Remediación:

- Documentar todas las vulnerabilidades encontradas, las técnicas de explotación y las recomendaciones de remediación.
- Priorizar las vulnerabilidades según su severidad y riesgo.

Documentación del Informe de Pruebas de Penetración:

- Resumen Ejecutivo: Descripción general del alcance y los objetivos de las pruebas, y un resumen de los hallazgos clave.
- Detalles Técnicos: Descripción detallada de cada vulnerabilidad encontrada, incluyendo la metodología utilizada para descubrirla y el impacto potencial.
- Recomendaciones de Remediación: Sugerencias específicas para corregir cada vulnerabilidad y mejorar la postura de seguridad general.

5.4 Evaluaciones de Seguridad Periódicas

La seguridad de la aplicación debe ser evaluada de manera continua para asegurar que las nuevas vulnerabilidades sean identificadas y corregidas rápidamente.

Metodología para Evaluaciones Periódicas:

- Análisis Regular del Código: Ejecutar análisis estáticos y dinámicos del código en cada ciclo de desarrollo o actualización importante.
- Pruebas de Penetración Anuales: Realizar pruebas de penetración completas al menos una vez al año o después de cambios significativos en la infraestructura.
- Revisión de Políticas y Procedimientos: Evaluar y actualizar las políticas de seguridad y procedimientos operativos regularmente.
- Monitoreo de Seguridad en Tiempo Real: Implementar sistemas de monitoreo para detectar y responder a incidentes de seguridad en tiempo real.

Conclusión

El análisis de seguridad abarca una serie de prácticas y herramientas diseñadas para identificar y mitigar vulnerabilidades en el sistema de la agenda electrónica. Al combinar análisis estáticos y dinámicos del código, pruebas de penetración y evaluaciones periódicas, se puede asegurar que la aplicación mantenga un alto nivel de seguridad y protección de los datos sensibles de los pacientes.

6. RESULTADOS Y CONCLUSIONES

En esta sección se presentan los resultados obtenidos a partir de la implementación y análisis de seguridad del sistema de la agenda electrónica para la gestión de datos de pacientes, así como las conclusiones derivadas de este trabajo. El objetivo es evaluar la eficacia de las medidas de seguridad implementadas y determinar el éxito del proyecto en términos de seguridad y funcionalidad.

6.1 Resultados

Los resultados del proyecto se pueden dividir en varias categorías: implementación del sistema, análisis de seguridad, y pruebas de penetración.

6.1.1 Implementación del Sistema

Autenticación Segura:

- Algoritmo de Hasheo Argon2:

- La implementación de Argon2 para el hash de contraseñas en el sistema de autenticación ha demostrado ser efectiva. Todas las contraseñas de los usuarios están cifradas de manera segura, protegiéndolas contra ataques de fuerza bruta y ataques de diccionario.

Cifrado de Datos:

- Algoritmo AES en Modo CBC:

- Los datos sensibles de los pacientes almacenados en la base de datos están cifrados utilizando AES en modo CBC. Esto asegura que incluso si un atacante obtiene acceso a la base de datos, los datos cifrados no se pueden leer sin la clave de cifrado y el vector de inicialización (IV).

Integración Backend-Frontend:

- La integración entre el backend (Flask/Django) y el frontend (React.js) se ha realizado con éxito, permitiendo una comunicación segura y eficiente entre ambos. La autenticación y autorización de usuarios se manejan correctamente, y los datos de los pacientes se transmiten de manera cifrada.

6.1.2 Análisis de Seguridad

Análisis Estático:

- SonarQube:

- El análisis con SonarQube ha identificado varias posibles vulnerabilidades y errores en el código, los cuales fueron corregidos antes del despliegue. Esto incluye la eliminación de código inseguro, mejora de la gestión de excepciones y corrección de problemas de seguridad.

- Bandit:

- Bandit ha detectado problemas específicos de seguridad en el código Python, como la gestión inadecuada de contraseñas y datos sensibles, y la utilización de funciones inseguras. Todas estas vulnerabilidades fueron mitigadas.

Análisis Dinámico:

- OWASP ZAP y Burp Suite:

- Las herramientas de análisis dinámico como OWASP ZAP y Burp Suite han identificado y permitido corregir vulnerabilidades de seguridad web, incluyendo inyecciones SQL, Cross-Site Scripting (XSS), y problemas de configuración de seguridad. Todas las vulnerabilidades críticas fueron abordadas.

6.1.3 Pruebas de Penetración

Resultados de las Pruebas:

- Las pruebas de penetración realizadas simularon ataques reales al sistema y ayudaron a identificar vulnerabilidades adicionales que no fueron detectadas por los análisis estáticos y dinámicos.

- Inyecciones SQL y XSS: Se identificaron y corrigieron vulnerabilidades relacionadas con inyecciones SQL y XSS.
- Configuración de Seguridad: Mejoras en la configuración de seguridad del servidor y la aplicación, incluyendo el uso de HTTPS y la configuración adecuada de cabeceras de seguridad.

Informe Final de Pruebas:

- Se generó un informe detallado de las pruebas de penetración, que documenta todas las vulnerabilidades encontradas, las técnicas de explotación utilizadas y las recomendaciones de remediación.

6.2 Conclusiones

Eficacia de las Medidas de Seguridad:

- Las medidas de seguridad implementadas, incluyendo el uso de Argon2 para el hash de contraseñas y AES en modo CBC para el cifrado de datos, han demostrado ser altamente efectivas en la protección de datos sensibles.
- El análisis de seguridad integral, que abarca análisis estáticos, dinámicos y pruebas de penetración, ha asegurado que el sistema esté bien protegido contra una variedad de amenazas y ataques.

Lecciones Aprendidas:

- La importancia de una arquitectura de seguridad bien definida y la implementación de medidas de seguridad desde el principio del desarrollo del proyecto.
- La necesidad de realizar evaluaciones de seguridad continuas y de estar siempre actualizado con las mejores prácticas y herramientas de seguridad.

Impacto del Proyecto:
- El sistema de la agenda electrónica ofrece una solución segura y eficiente para la gestión de datos de pacientes, cumpliendo con los requisitos de confidencialidad, integridad y disponibilidad de los datos.
- Este proyecto sirve como un modelo para futuras implementaciones en el campo de la salud y otras industrias donde la seguridad de los datos es crítica.

Futuras Mejoras:

- Monitoreo y Respuesta a Incidentes: Implementar un sistema de monitoreo en tiempo real para detectar y responder rápidamente a posibles incidentes de seguridad.

ENRIQUE LÓPEZ
JAVIER IBARRIA

- Capacitación en Seguridad: Proporcionar capacitación continua en seguridad para el equipo de desarrollo para asegurar que estén al tanto de las últimas amenazas y técnicas de mitigación.
- Revisión y Actualización Constante: Continuar revisando y actualizando las políticas y procedimientos de seguridad para mantenerse al día con las mejores prácticas de la industria y los cambios regulatorios.

Conclusión Final

El proyecto de la agenda electrónica para la gestión de datos de pacientes ha logrado su objetivo principal de implementar un sistema seguro y funcional. A través de una cuidadosa planificación, implementación y análisis de seguridad, se ha desarrollado una solución robusta que protege los datos sensibles de los pacientes. Este proyecto destaca la importancia de integrar la seguridad en todas las etapas del desarrollo y sirve como una guía para futuros proyectos en el campo de la ciberseguridad y la protección de datos.

ENRIQUE LÓPEZ
JAVIER IBARRIA