CYBERSHIELD
CONSULTING

Cybershield Consulting

Informe de resultados de auditoría de seguridad

Análisis estático de capa de servicios de aplicación financiera

Confidencialidad empresarial

8 de febrero de 2025

Proyecto: Versión 01

Contenido

Declaración de confidencialidad

Este documento es propiedad exclusiva de Cybershield. Este documento contiene información propietaria y confidencial. La reproducción, redistribución o utilización, total o parcial, en cualquier forma, requiere el consentimiento de Cybershield. Corporación Cybershield puede compartir este documento con auditores bajo acuerdos de no divulgación para demostrar el cumplimiento de los requisitos de las pruebas de penetración

Descargo de responsabilidad

Un análisis estático se considera un snapshot en el tiempo. Las conclusiones y recomendaciones reflejan la información recopilada durante la evaluación y no los cambios o modificaciones realizados fuera de este período.
Una auditoría de seguridad limitada en su duración no permite una evaluación completa de todos los controles de seguridad. Cybershield priorizó la revisión del código y la identificación de vulnerabilidades más críticas que un atacante podría explotar. Cybershield recomienda llevar a cabo análisis similares anualmente por parte de evaluadores internos o externos para garantizar el éxito ininterrumpido de los controles.

Información de contacto

Responsabilidad	Nombre	Información de contacto
Elaboración, evaluación y redacción	Xavier Carpio	xavier.carpio@epn.edu.ec
Revisión y aprobación	Erick Cabezas	erick.cabezas@epn.edu.ec
	Britanny Flores	britanny.flores@epn.edu.ec
	Carlos Landazuri	carlos.landazuri@epn.edu.ec
	Kenny Pinchao	kenny.pinchao@epn.edu.ec

Evaluación general

El 8 de febrero de 2025, Cybershield Consulting evaluó el estado general de seguridad de su infraestructura en comparación con las mejores prácticas actuales del sector, enfocándose en un análisis estático del código fuente. Todas las actividades realizadas se basaron en la Guía técnica de pruebas y evaluación de seguridad de la información NIST SP 800-115, la Guía de pruebas OWASP (v4) y marcos de análisis personalizados.

Las fases de las actividades de análisis estático realizadas se incluyen a continuación:

- **Planificación**: Se reunieron los objetivos del cliente y se establecieron las reglas de compromiso para el análisis.
- **Revisión de código**: Se analizó el código fuente para identificar vulnerabilidades potenciales, malas prácticas y áreas débiles en la implementación de seguridad.
- **Validación**: Se confirmó la validez de las vulnerabilidades detectadas, identificando su impacto y probabilidad de explotación en el contexto de la organización.
- **Reporte**: Se documentaron todas las vulnerabilidades encontradas, incluyendo recomendaciones específicas para su mitigación, además de un análisis detallado de los puntos débiles y fuertes de la organización.

Cybershield recomienda realizar evaluaciones similares de manera periódica para garantizar la mejora continua de los controles de seguridad implementados.

Componentes de la evaluación

Análisis estático de código

Los componentes típicos de evaluación de un análisis estático incluyen diferentes aspectos y áreas que permiten identificar vulnerabilidades y debilidades en el código fuente de una aplicación. Para empezar, se identificarán vulnerabilidades como inyección de código, exposición a datos sensibles, errores de configuración de seguridad, deserialización insegura, uso de componentes vulnerables. Se realizará una revisión de algoritmos criptográficos, detección de cifrados débiles, análisis de dependencias y bibliotecas.

Para este análisis se realizarán dos evaluaciones, una manual desarrollada por **Xavier Carpio** y en la segunda se utilizará CodeQL de GitHub para evaluar las vulnerabilidades: https://codeql.github.com

Índices de severidad de los hallazgos

La siguiente tabla define los niveles de gravedad y los criterios utilizados en este análisis, basado en los hallazgos reportados por CodeQL de GitHub, para evaluar las vulnerabilidades y el impacto del riesgo

Severidad	Definición	Acción recomendada
Crítica	Las vulnerabilidades identificadas tienen un impacto severo y pueden ser explotadas fácilmente, lo que podría llevar a un compromiso total del sistema. Estas vulnerabilidades suelen involucrar errores críticos en el código fuente que requieren atención inmediata.	Implementar un plan de acción y resolver el problema de forma prioritaria. Parchear de inmediato.
Alta	Incluye vulnerabilidades que, aunque menos sencillas de explotar, pueden permitir escalación de privilegios, filtración de datos sensibles o interrupciones críticas.	Elaborar un plan de acción detallado y abordar estas vulnerabilidades lo antes posible.
Moderada	Se detectan vulnerabilidades que requieren un contexto específico o condiciones más complejas para ser explotadas, como la ingeniería social o interacciones específicas.	Priorizar estas vulnerabilidades después de resolver las críticas y altas. Parchear en un plazo razonable.
Baja	Incluye problemas que no son directamente explotables, pero podrían incrementar la superficie de ataque, como malas prácticas de codificación o configuraciones inseguras.	Solucionar durante próximas actualizaciones o ventanas de mantenimiento.
Informativa	No se detectaron vulnerabilidades críticas. Se proporciona información relevante, como puntos de optimización, controles adicionales aplicados y detalles sobre la estructura del código.	Usar esta información como referencia para mejorar el código y la documentación.

Factores de riesgo

El riesgo se mide por dos factores: Probabilidad e Impacto:

Probabilidad

La probabilidad mide el potencial de explotación de una vulnerabilidad. Las puntuaciones se otorgan en función de la dificultad del ataque, las herramientas disponibles, el nivel de habilidad del atacante y el entorno del cliente.

Impacto

El impacto mide el efecto potencial de la vulnerabilidad en las operaciones, incluyendo la confidencialidad, integridad y disponibilidad de los sistemas y/o datos del cliente, el daño a la reputación y las pérdidas financieras.

Resumen ejecutivo

Cybershield Consulting evaluó el estado general de seguridad interna de ExpertGuide mediante pruebas de seguridad realizadas en el código de la aplicación. Las siguientes secciones proporcionan una visión general de alto nivel de las vulnerabilidades descubiertas, los intentos exitosos y fallidos, y las fortalezas y debilidades.

Resultados del análisis estático de código

1	4	5	0	0
Crítica	**Alta**	**Moderada**	**Baja**	**Informativa**

Hallazgo	Severidad	Recomendación
V1: Almacenamiento de Contraseñas en Texto Plano	Crítica	Implementar hashing de contraseñas usando algoritmos seguros como Argon2, bcrypt o PBKDF2 con salt único por usuario.
V2: SQL Injection Vulnerable	Alta	Parametrizar todas las consultas SQL, incluso aquellas que no toman entrada del usuario.
V3: Gestión insegura de tokens	Alta	Implementar expiración de tokens, rotación periódica y almacenamiento seguro con timestamps de creación y expiración.
V4: Ausencia rate limiting	Alta	Se debe utilizar un middleware que limite la velocidad para evitar este tipo de ataques.
V5: Flask app se encuentra ejecutando en modo debug	Alta	Asegúrese de que las aplicaciones Flask que se ejecutan en un entorno de producción tengan la depuración deshabilitada.
V6: Información Sensible en Logs	Moderada	Implementar logging seguro que excluya información sensible y establecer niveles apropiados de logging.

Hallazgos técnicos

Resultados técnicos del análisis estático de código

V1: Almacenamiento de Contraseñas en Texto Plano (Crítica)

Descripción	Las contraseñas se almacenan en texto plano en la base de datos sin ningún tipo de hash o salt. Esto se evidencia en la tabla bank.users y en la validación de contraseñas durante el login.
Riesgo	**Probabilidad:** Alta - Los atacantes que obtengan acceso a la base de datos podrán ver directamente todas las contraseñas. **Impacto:** Alto - Compromiso total de todas las cuentas de usuario y posible reutilización de contraseñas en otros servicios.
Módulo	Main - /login db – tabla cuentas
Herramientas utilizadas	Análisis manual
Referencias	CWE-256: Unprotected Storage of Credentials OWASP Top 10 2021: A07 - Identification and Authentication Failures NIST SP 800-63B: 5.1.1.2 Memorized Secret Verifiers

Evidencia

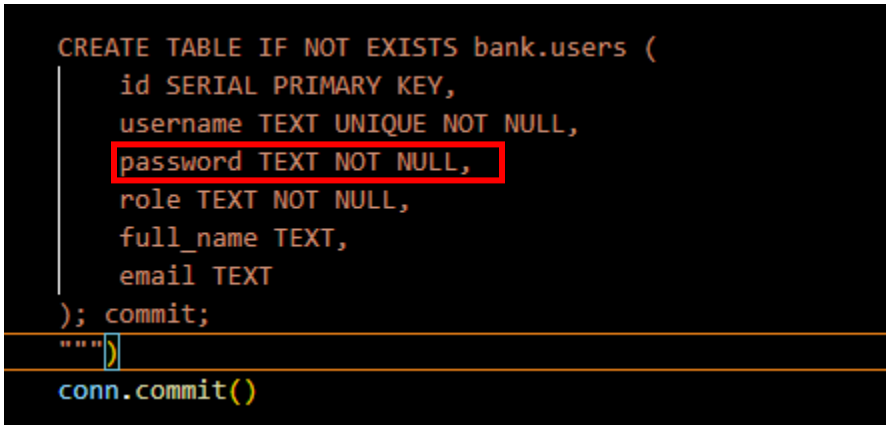

Figura 1. Código vulnerable con contraseñas no hasheadas

Recomendación

Implementar hashing de contraseñas usando algoritmos seguros como Argon2, bcrypt o PBKDF2 con salt único por usuario.

Ejemplo de remediación en código

```
from argon2 import PasswordHasher
ph = PasswordHasher()
```

```
def hash_password(password: str) -> str:
      return ph.hash(password)


cur.execute(""" CREATE TABLE IF NOT EXISTS bank.users (
      id SERIAL PRIMARY KEY,
      username TEXT UNIQUE NOT NULL,
      password_hash TEXT NOT NULL,
      role TEXT NOT NULL, full_name TEXT,
      email TEXT );
""")
```

V2: SQL Injection Vulnerable (Alto)

Descripción	Aunque se utiliza psycopg2 con parámetros en la mayoría de las consultas, existen instancias donde las consultas son construidas directamente, lo que podría permitir SQL injection.
Riesgo	**Probabilidad**: Media - Aunque la mayoría de las consultas están parametrizadas, las consultas directas podrían ser explotadas. **Impacto**: Alto - Un atacante podría ejecutar consultas arbitrarias en la base de datos.
Módulo	db – init_db
Herramientas utilizadas	Análisis manual
Referencias	CWE-89: SQL Injection OWASP Top 10 2021: A03 - Injection NIST SP 800-53: SI-10 Information Input Validation

Evidencia

```
# Insertar datos de ejemplo si no existen usuarios
cur.execute("SELECT COUNT(*) FROM bank.users;")
```

Figura 2. Consulta no parametrizada

Recomendación

Parametrizar todas las consultas SQL, incluso aquellas que no toman entrada del usuario.

Ejemplo remediación en código

```
# Clase para manejar consultas comunes
class DatabaseQueries:
    @staticmethod
    def count_users(cur) -> int:
        cur.execute("SELECT COUNT(*) FROM bank.users")
        return cur.fetchone()[0]

    @staticmethod
    def get_user_by_username(cur, username: str):
        cur.execute("""
            SELECT id, username, password_hash, role
            FROM bank.users
            WHERE username = %s
        """, (username,))
        return cur.fetchone()

    @staticmethod
    def create_user(cur, username: str, password_hash: str, role: str):
        cur.execute("""
            INSERT INTO bank.users (username, password_hash, role)
            VALUES (%s, %s, %s)
            RETURNING id
        """, (username, password_hash, role))
        return cur.fetchone()[0]

# Uso
db = DatabaseQueries()
user_count = db.count_users(cur)
```

V3: Gestión Insegura de Tokens (Alto)

Descripción	Los tokens de autenticación no tienen expiración y no se implementa rotación de tokens. Además, se utiliza un algoritmo simple (secrets.token_hex) sin consideraciones adicionales de seguridad.
Riesgo	**Probabilidad**: Media - Los tokens pueden ser interceptados y usados indefinidamente. **Impacto**: Alto - Compromiso permanente de la sesión del usuario afectado.
Módulo	main.py - login
Herramientas utilizadas	Análisis manual
Referencias	CWE-613: Insufficient Session Expiration OWASP Top 10 2021: A07 - Identification and Authentication Failures NIST SP 800-63B: 4.3 Session Management

Evidencias

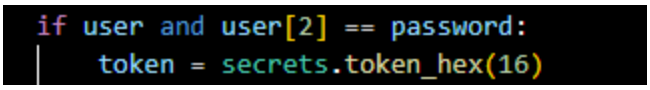

Figura 3. Gestión insegura de token de acceso

Recomendación

Implementar expiración de tokens, rotación periódica y almacenamiento seguro con timestamps de creación y expiración.

Ejemplo remediación en código

```
from datetime import datetime
import secrets
import jwt
# Crear token JWT con claims seguros
token = jwt.encode({
            'user_id': user_id,
            'exp': expiration,
            'iat': datetime.utcnow(),
            'jti': secrets.token_hex(16)  # Identificador único del token
        }, self.secret_key, algorithm='HS256')
```

V4: Ausencia rate limiting (Alto)

Descripción	Los manejadores de solicitudes HTTP no deben realizar operaciones costosas como acceder al sistema de archivos, ejecutar un comando del sistema operativo o interactuar con una base de datos sin limitar la velocidad a la que se aceptan las solicitudes. Para la autorización en rutas no existe un límite de peticiones, lo que podría causar un desbordamiento
Riesgo	Probabilidad: Alto – Existen numerosas herramientas automatizadas que pueden realizar ataques DoS al sitio web. Impacto: Moderado – La disponibilidad del sistema se ve comprometida.
Módulo	main.py – TODAS LAS RUTAS
Herramientas utilizadas	Análisis manual
Referencias	CWE-307: Improper Restriction of Excessive Authentication Attempts OWASP Top 10 2021: A04 - Insecure Design NIST SP 800-53: SC-5 Denial of Service Protection

Evidencias

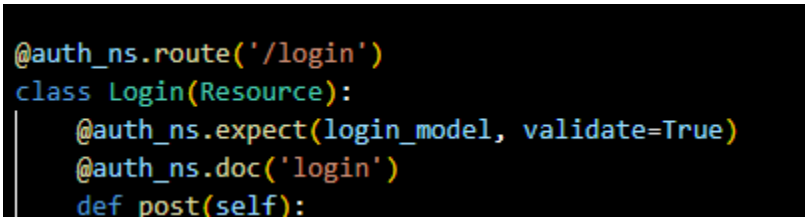

Figura 4. Ruta sin límite de peticiones

Remediación

Implementar un middleware encargado de gestionar y aplicar límites al número de peticiones HTTP que los usuarios pueden realizar. Este middleware permitirá controlar el tráfico hacia el servidor, previniendo el abuso de recursos y mejorando la estabilidad del sistema mediante el establecimiento de restricciones específicas para cada usuario, dirección IP u otras métricas configurables.

Ejemplo recomendación en código

```
from flask import request
from functools import wraps

def rate_limit(requests: int, window: int, by: str = 'ip'):
  def decorator(f):
    @wraps(f)
    def wrapped(*args, **kwargs):
      if is_rate_limited(by, requests, window):
        return {
          'error': 'Too many requests',
          'retry_after': window
        }, 429
      return f(*args, **kwargs)
    return wrapped
  return decorator
```

V5: Flask app se encuentra ejecutando en modo debug (Alto)

Descripción	Parece que una aplicación Flask se ejecuta en modo de depuración. Esto puede permitir que un atacante ejecute código arbitrario a través del depurador.
Riesgo	Probabilidad: Alto – Existen scripts y algoritmos en Internet que permiten identificar esta vulnerabilidad

	Impacto: Alto – El acceso al debugger permite correr código arbitrario.
Módulo	Autorización de rutas
Herramientas utilizadas	CodeQL
Referencias	Flask Quickstart Documentation: Debug Mode. Werkzeug Documentation: Debugging Applications. CWE-215: Insertion of Sensitive Information Into Debugging Code CWE-489: Active Debug Code

Evidencia

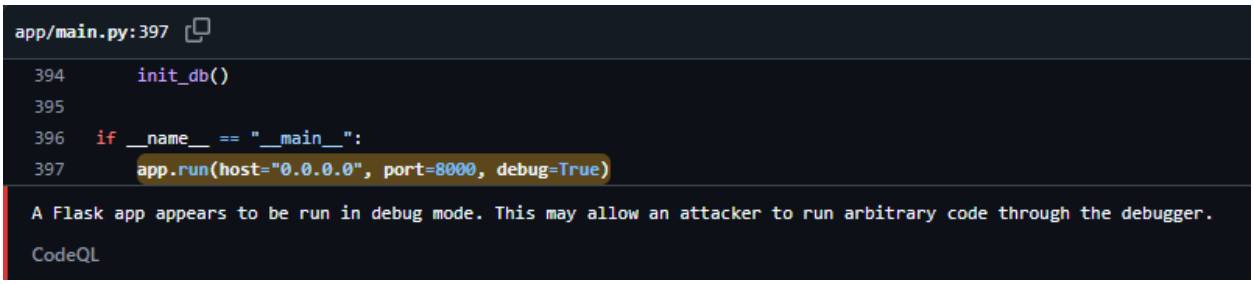

Figura 5. Flask ejecutándose en entorno debug

Recomendación

Asegúrese de que las aplicaciones Flask que se ejecutan en un entorno de producción tengan la depuración deshabilitada.

V6: Información Sensible en Logs (Moderada)

Descripción	Se registran tokens de autenticación y potencialmente otra información sensible en los logs de la aplicación.
Riesgo	**Probabilidad**: Media - Los logs pueden ser accedidos por personal no autorizado o comprometidos. **Impacto**: Medio - Exposición de información sensible que podría llevar a compromiso de sesiones.
Módulo	main.py – toker_required
Herramientas utilizadas	Análisis manual
Referencias	CWE-532: Insertion of Sensitive Information into Log File OWASP Top 10 2021: A09 - Security Logging and Monitoring Failures NIST SP 800-53: SI-11 Error Handling

Evidencia

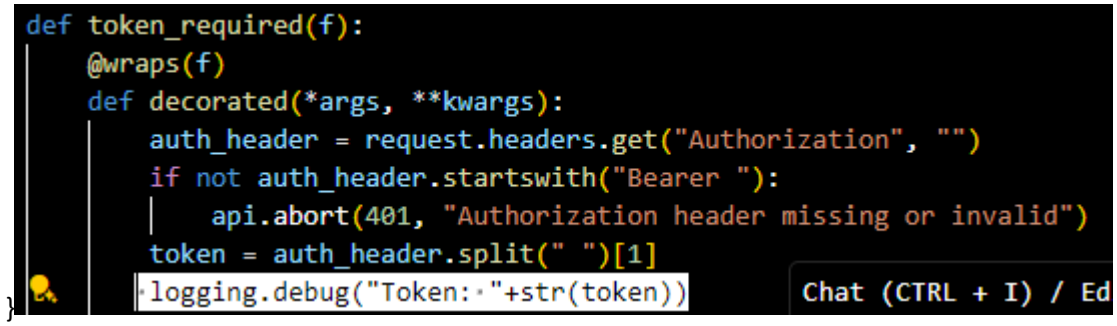

Figura 6. Token incluido en log

Recomendación

Implementar logging seguro que excluya información sensible y establecer niveles apropiados de logging.

Ejemplo remediación en código

```
import logging
import json
from typing import Any, Dict
from datetime import datetime

class SecureLogger:
    def __init__(self, app_name: str):
        self.logger = logging.getLogger(app_name)
        self._setup_logger()

    def _setup_logger(self):
        formatter = logging.Formatter(
            '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
        )

        # Configurar handler para archivo
        file_handler = logging.FileHandler('secure.log')
        file_handler.setFormatter(formatter)
        self.logger.addHandler(file_handler)
        self.logger.setLevel(logging.INFO)
```