Trabajo Integrador (Primer entregable)

DESARROLLO DE SOFTWARE SEGURO

Prof. Lic. Juan Pablo Villalba

Pablo Rodrigo Moya

Matias Curti

Universidad de Gran Rosario

INDICE

Ejercicio Interactivo de Desarrollo de Software Seguro: Creando una Aplicación para "FitLife"

Contexto del Proyecto

Imagina que eres parte del equipo de desarrollo de "FitLife", una empresa ficticia que ofrece una aplicación web para ayudar a las personas a llevar un estilo de vida saludable. La aplicación permite a los usuarios registrarse, iniciar sesión, seguir sus rutinas de ejercicio y gestionar su dieta. Como parte del equipo, tu responsabilidad es asegurar que la aplicación sea segura y proteja la información sensible de los usuarios.

Objetivo del Ejercicio

El objetivo de este ejercicio es que los estudiantes aprendan a aplicar los principios de OWASP en el desarrollo de una aplicación web, utilizando pseudocódigo y código Python para implementar las mejores prácticas de seguridad.

Entregas Parciales

Entrega 1: Análisis de Riesgos y Diseño Seguro (Clase 1)

- Tarea: Investigar y discutir en grupos los 10 principales riesgos de OWASP. Cada grupo seleccionará uno o dos riesgos que podrían afectar a la aplicación "FitLife".
- Entregable: Un breve informe que describa los riesgos seleccionados y sus posibles impactos en la aplicación.
- Pseudocódigo y Python: Crear un pseudocódigo y código Python básico de las funciones de registro e inicio de sesión, incorporando medidas de seguridad basadas en OWASP.

Entrega 2: Validación y Sanitización (Clase 2)

- Tarea: Implementar funciones de validación y sanitización de entradas de usuario.

- Entregable: Código de las funciones de validación y un informe sobre la importancia de la validación de entradas.
- Pseudocódigo y Python: Crear pseudocódigo y código Python para la validación de entradas.

Entrega 3: Autenticación y Manejo de Sesiones (Clase 3)

- Tarea: Implementar un sistema de autenticación seguro y manejo de sesiones.
- Entregable: Código de las funciones de autenticación y un informe sobre las mejores prácticas de gestión de sesiones.
- Pseudocódigo y Python: Crear pseudocódigo y código Python para el manejo de sesiones.

Entrega 4: Pruebas de Seguridad (Clase 4)

- Tarea: Realizar pruebas de seguridad en la aplicación "FitLife" y documentar las vulnerabilidades encontradas.
- Entregable: Informe de pruebas con resultados y correcciones implementadas.
- Pseudocódigo y Python: Estrategias de pruebas de seguridad.

Entrega 5: Implementación de Medidas de Seguridad (Clase 5)

- Tarea: Implementar medidas de seguridad adicionales basadas en los aprendizajes de las clases anteriores.
- Entregable: Código de las medidas de seguridad implementadas y un informe sobre cómo estas medidas ayudan a mitigar los riesgos de seguridad.
- Pseudocódigo y Python: Crear pseudocódigo y código Python para las medidas de seguridad adicionales.

Entrega Final: Aplicación Funcional y Segura (Clase 6)

- Tarea: Entregar la aplicación "FitLife" completa, funcional y segura, implementando todas las medidas de seguridad aprendidas.

- Entregable: Código Python de la aplicación "FitLife" que incluya todas las funcionalidades y prácticas de seguridad basadas en OWASP.

Evaluación

Los estudiantes serán evaluados en función de:

- La calidad de sus entregables.
- La aplicación de prácticas de seguridad basadas en OWASP.
- La claridad y lógica del pseudocódigo y código Python.
- La funcionalidad y seguridad de la aplicación final.
- La colaboración y participación en las actividades grupales.

Conclusión

Este ejercicio interactivo no solo permite a los estudiantes aprender sobre OWASP y las vulnerabilidades comunes en aplicaciones web, sino que también les brinda la oportunidad de aplicar este conocimiento en un contexto práctico y colaborativo. Al trabajar en un proyecto realista como "FitLife" y utilizar tanto pseudocódigo como código Python, los estudiantes desarrollarán habilidades críticas para su futura carrera en el desarrollo de software seguro.

Requisitos:

Carátula, índice, cuerpo, conclusiones, referencias (APA)

Nociones preliminares

Teniendo en cuenta la necesidad de presentar una aplicación funcional al finalizado este trabajo, a raiz de que pueden surgir cambios a lo largo de los proximos entregables y las consideraciones del profesor a cargo de la catedra pueden variar respecto de la cantidad de entregables y/o el contenido de los mismos, a sabiendas de que el enfoque del trabajo esta en la seguridad en el desarrollo de la aplicación para “FitLife” respecto de las funciones creadas en python y que la misma puede ser trabajada y evaluada al rededor del uso de la misma a través de la consola de comandos, es que decidimos realizar este trabajo con el uso de HTML para presentar una UI para la facil visualización del proyecto y testeo del mismo, el uso de Python por cuestiones de obligatoriedad, y el uso de importaciones de librerias o frameworks para gestionar las distintas tareas que se iran dando a lo largo y ancho de este trabajo.

Analisis de Riesgos y Diseño Seguro

Para este primer entregable, recordando la importancia de la utilización de la información aportada por OWASP TOP 10 como pilar fundamental de este trabajo, los dos riesgos afrontados son:

1) **A01:2021 – Pérdida de Control de Acceso / CWE-352 Cross-Site Request Forgery (CSRF)**

   Explicación: El control de acceso tiene que implementar el cumplimiento de politicas de modo que los usuarios no puedan actuar fuera de los permisos asignados.

   Impacto en FitLife: Divulgación de información no autorizada, modificación o destrucción de datos, ejecución de funciones de negocio fuera de los limites del usuario.

2) **A03:2021 – Inyección**

   Explicación: El formulario acepta datos no validados, ni filtrados.

Impacto en FitLife: Extracción de registros sensibles, se permite invocar consultas dinamicas o no parametrizadas, afectando los datos almacenados, entre otros.

Vulnerabilidad 1

Pseudocódigo:

```
inicializarAplicacion()
# Inicializa la aplicación, configurando los parámetros y recursos necesario

establecerClaveSecreta('supersecretkey')
# Establece una clave secreta utilizada para la seguridad de la aplicación

habilitarProteccionCSRF()
# Habilita la protección contra ataques CSRF

configurarAplicacion(
    DEBUG=True,
    SECRET_KEY='supersecretkey')
# Configura la aplicación con parámetros específicos, como habilitar el modo de depuración (DEBUG) y establecer la clave secreta.

inicializarLoginManager()
# Inicializa el gestor de inicio de sesión, que maneja la autenticación de usuarios en la aplicación.

usuarios = diccionarioVacio()
# Crea un diccionario vacío para almacenar los usuarios registrados en la aplicación.

funcion principal():
    mostrarPagina('index.html')
    # Muestra la página principal de la aplicación.
```

```
funcion registrar():
    username = obtenerFormulario('new_username')
    password = obtenerFormulario('new_password')
    # Obtiene el nombre de usuario y la contraseña del formulario de registro.

    si username existe en usuarios:
        mostrarMensaje('El usuario ya existe. Intenta con otro.')
        # Verifica si el nombre de usuario ya existe en el diccionario de usuarios y muestra un mensaje si es así.

    sino:
        agregarUsuario(usuarios, username, password)
        mostrarMensaje('Usuario registrado exitosamente. Ahora puedes hacer login.')
        # Si el nombre de usuario no existe, lo agrega al diccionario de usuarios y muestra un mensaje de éxito.

    redirigir('index')
    # Redirige al usuario a la página principal después del registro.

funcion iniciarSesion():
    username = obtenerFormulario('username')
    password = obtenerFormulario('password')
    # Obtiene el nombre de usuario y la contraseña del formulario de inicio de sesión.

    si username existe en usuarios y usuarios[username] == password:
        establecerSesion('username', username)
        mostrarMensaje('¡Has iniciado sesión exitosamente!')
        redirigir('dashboard')
        # Verifica si el nombre de usuario existe y la contraseña es correcta, establece la sesión y redirige al panel de control.

    sino:
        mostrarMensaje('Usuario o contraseña incorrectos. Intenta nuevamente.')
        redirigir('index')
        # Si el nombre de usuario no existe o la contraseña es incorrecta, muestra un mensaje de error y redirige a la página principal.

si __nombre__ == '__main__':
    ejecutarAplicacion(debug=True)
    # Ejecuta la aplicación en modo de depuración si el script se ejecuta directamente.
```

Vulnerabilidad 1

Código en Python:

```python
from flask import Flask, render_template, request, redirect, url_for, session, flash
from flask_login import LoginManager
from flask_wtf.csrf import CSRFProtect

app = Flask(__name__)
app.secret_key = 'supersecretkey'  # Necesario para manejar sesiones y CSRF
csrf = CSRFProtect(app)  # Habilitar protección CSRF

app.config.update(
    DEBUG=True,  # Es mala práctica dejar activado esto en producción.
    SECRET_KEY="supersecretkey",)

login_manager = LoginManager()
login_manager.init_app(app)

# Diccionario temporal para almacenar usuarios (usuario: contraseña)
usuarios = {}
```

```python
# Ruta principal para mostrar la página de registro/login
@app.route('/')
def index():
    return render_template('index.html')
    # Renderiza la página principal de la aplicación, que contiene los formularios de registro y login.

# Ruta para el registro
@app.route('/register', methods=['POST'])
def register():
    username = request.form['new_username']
    password = request.form['new_password']
    # Obtiene el nombre de usuario y la contraseña del formulario de registro.

    if username in usuarios:
        flash('El usuario ya existe. Intenta con otro.')
        # Verifica si el nombre de usuario ya existe en el diccionario de usuarios y muestra un mensaje si es así.
    else:
        usuarios[username] = password
        flash('Usuario registrado exitosamente. Ahora puedes hacer login.')
        # Si el nombre de usuario no existe, lo agrega al diccionario de usuarios y muestra un mensaje de éxito.

    return redirect(url_for('index'))
    # Redirige al usuario a la página principal después del registro.
```

```python

# Ruta para el login
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    # Obtiene el nombre de usuario y la contraseña del formulario de inicio de sesión.

    if username in usuarios and usuarios[username] == password:
        session['username'] = username
        flash('¡Has iniciado sesión exitosamente!')
        return redirect(url_for('dashboard'))
        # Verifica si el nombre de usuario existe y la contraseña es correcta, establece la sesión y redirige al panel de control.
    else:
        flash('Usuario o contraseña incorrectos. Intenta nuevamente.')
        return redirect(url_for('index'))
        # Si el nombre de usuario no existe o la contraseña es incorrecta, muestra un mensaje de error y redirige a la página principal.

if __name__ == '__main__':
    app.run(debug=True)
    # Ejecuta la aplicación en modo de depuración si el script se ejecuta directamente.

```

Código HTML:

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Integrador</title>
</head>
<body>
    <h1>Universidad de Gran Rosario (Virtual)</h1>
    <h2>Tecnicatura en Ciberseguridad</h2>
    <h2>Matias Curti - Pablo Rodrigo Moya </h2>
    <h3>Trabajo Integrador</h3>
    <!-- Formulario de Register -->
    <form action='/register' method="POST" autocomplete="off">
        <input type="text" name="new_username" placeholder="Nuevo Usuario">
        <input type="password" name="new_password" placeholder="Contraseña">
        <button type="submit" name="submit" value="register"> Registrar </button>
        <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
        <!-- En realidad como el template maneja flask esta ultima linea se puede simplificar como {{ form.csrf_token }} y listo-->
    </form>

    <!-- Formulario de Login -->
    <form action="/login" method="POST" autocomplete="off">
        <input type="text" name="username" placeholder="Usuario">
        <input type="password" name="password" placeholder="Contraseña">
        <button type="submit" name="submit" value="register"> Iniciar Sesión </button>
        <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
        <!-- En realidad como el template maneja flask esta ultima linea se puede simplificar como {{ form.csrf_token }} al igual que el de register-->
    </form>
</body>
</html>
```

Vulnerabilidad 2

Pseudocódigo:

```
INICIAR aplicación
# Inicia la aplicación principal

CONFIGURAR clave_secreta = 'supersecretkey'
# Configura una clave secreta para la aplicación, utilizada para firmar cookies y otros datos

HABILITAR protección CSRF
# Habilita la protección contra ataques CSRF (Cross-Site Request Forgery)

CONFIGURAR aplicación:
    DEBUG = verdadero
    # Activa el modo de depuración para la aplicación, útil durante el desarrollo
    SECRET_KEY = "supersecretkey"
    # Establece la clave secreta para la aplicación

INICIAR gestor_de_login
# Inicia el gestor de inicio de sesión para manejar la autenticación de usuarios

CLASE Usuario:
    FUNCIÓN __init__(nombre_usuario, contraseña):
        ASIGNAR nombre_usuario
        # Asigna el nombre de usuario al atributo de la clase
        ASIGNAR contraseña
        # Asigna la contraseña al atributo de la clase

    FUNCIÓN obtener_id():
        RETORNAR nombre_usuario
        # Retorna el nombre de usuario como identificador único
```

```

FUNCIÓN cargar_usuario(nombre_usuario):
    EJECUTAR consulta SQL 'SELECT * FROM usuarios WHERE username = nombre_usuario'
    # Ejecuta una consulta SQL para buscar un usuario por su nombre de usuario
    SI usuario encontrado:
        RETORNAR Usuario(nombre_usuario, contraseña)
        # Si se encuentra el usuario, retorna una instancia de la clase Usuario
    SINO:
        RETORNAR nulo
        # Si no se encuentra el usuario, retorna nulo

CONECTAR a base_de_datos "usuarios.db"
# Conecta a la base de datos llamada "usuarios.db"

EJECUTAR consulta SQL '''
    CREATE TABLE IF NOT EXISTS usuarios (
        username TEXT PRIMARY KEY,
        password TEXT NOT NULL
    )
'''
# Ejecuta una consulta SQL para crear la tabla "usuarios" si no existe, con columnas para el nombre de usuario y la contraseña

RUTA '/login' MÉTODO POST
# Define una ruta para el inicio de sesión que acepta solicitudes POST

```

```
FUNCIÓN iniciar_sesión():
    OBTENER nombre_usuario de formulario
    # Obtiene el nombre de usuario del formulario de inicio de sesión
    OBTENER contraseña de formulario
    # Obtiene la contraseña del formulario de inicio de sesión

    EJECUTAR consulta SQL 'SELECT * FROM usuarios WHERE username = nombre_usuario AND
    password = contraseña'
    # Ejecuta una consulta SQL para verificar las credenciales del usuario
    SI usuario encontrado:
        CREAR objeto_usuario Usuario(nombre_usuario, contraseña)
        # Si se encuentra el usuario, crea un objeto de la clase Usuario
        INICIAR sesión con objeto_usuario
        # Inicia sesión con el objeto de usuario
        MOSTRAR mensaje '¡Has iniciado sesión exitosamente!'
        # Muestra un mensaje de éxito
        REDIRIGIR a 'tablero'
        # Redirige al usuario al tablero principal
    SINO:
        MOSTRAR mensaje 'Usuario o contraseña incorrectos. Intenta nuevamente.'
        # Muestra un mensaje de error si las credenciales son incorrectas
        REDIRIGIR a 'índice'
        # Redirige al usuario a la página de inicio

SI __name__ == '__main__':
    EJECUTAR aplicación en modo debug
    # Ejecuta la aplicación en modo de depuración si este archivo es el principal

```

Vulnerabilidad 2

Código:

```python
from sqlite4 import SQLite4
from flask import Flask, render_template, request, redirect, url_for, session, flash
from flask_login import LoginManager, UserMixin, login_user
from flask_wtf.csrf import CSRFProtect

app = Flask(__name__)  # Crear una instancia de la aplicación Flask
app.secret_key = 'supersecretkey'  # Necesario para manejar sesiones y CSRF
csrf = CSRFProtect(app)  # Habilitar protección CSRF

app.config.update(  # Configuración de la aplicación
    DEBUG=True,  # Es mala práctica dejar activado esto en producción.
    SECRET_KEY="supersecretkey",
)

# Inicializar el gestor de inicio de sesión
login_manager = LoginManager()
login_manager.init_app(app)

class User(UserMixin):
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def get_id(self):
        return self.username

@login_manager.user_loader
def load_user(username):
    database.execute('SELECT * FROM usuarios WHERE username = ?', (username,))
    user = database.fetchone()
    if user:
        return User(username=user[0], password=user[1])
    return None

# Conexión a la base de datos SQLite4
database = SQLite4("usuarios.db")
database.connect()
```

```python

# Crear tabla de usuarios si no existe
database.execute('''
CREATE TABLE IF NOT EXISTS usuarios (
    username TEXT PRIMARY KEY,
    password TEXT NOT NULL
)
''')

# Ruta principal para mostrar la página de registro/login
@app.route('/')
def index():
    return render_template('index.html')
    # Renderiza la página principal de la aplicación, que contiene los formularios de registro y login.

# Ruta para el registro
@app.route('/register', methods=['POST'])
def register():
    username = request.form['new_username']
    password = request.form['new_password']
    # Obtiene el nombre de usuario y la contraseña del formulario de registro.

    # Verificar si el usuario ya existe
    database.execute('SELECT * FROM usuarios WHERE username = ?', (username,))
    if database.fetchone():
        flash('El usuario ya existe. Intenta con otro.')
    else:
        # Insertar el nuevo usuario en la base de datos
        database.execute('INSERT INTO usuarios (username, password) VALUES (?, ?)', (username, password))
        database.execute('COMMIT')
        flash('Usuario registrado exitosamente. Ahora puedes hacer login.')

    return redirect(url_for('index'))
    # Redirige al usuario a la página principal después del registro.
```

```python
    # Redirige al usuario a la página principal después del registro.

# Ruta para el login
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    # Obtiene el nombre de usuario y la contraseña del formulario de inicio de sesión.

    # Verificar las credenciales del usuario
    database.execute('SELECT * FROM usuarios WHERE username = ? AND password = ?', (username,
    password))
    user = database.fetchone()
    if user:
        user_obj = User(username=user[0], password=user[1])
        login_user(user_obj)
        flash('¡Has iniciado sesión exitosamente!')
        return redirect(url_for('dashboard'))
    else:
        flash('Usuario o contraseña incorrectos. Intenta nuevamente.')
        return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(debug=True)
    # Ejecuta la aplicación en modo de depuración si el script se ejecuta directamente.

```

Código HTML: No presenta cambios.

Conclusiones en base a cada Vulnerabilidad.

Vulnerabilidad 1 y su código:

Haciendo un recorrido de lo realizado en el código, donde se importan librerias, se configura la app y habilitamos csrf, como al comienzo del desarrollo no se presentaba la necesidad de crear una base de datos para usuarios lo que se realizó fue crerar un diccionario provisorio a los fines de poder guardar los usuarios, se define la ruta de /register donde el formulario en HTML deberia usar los metodos mencionados en la aplicación.py, se crea ruta /login para hacer exacamente lo mismo que se realizó anteriormente con register, y se plantea una validación basica donde si el usuario digitado se encuentra en el diccionario de usuario y a su vez ese usuario digitado y buscado tiene la misma contraseña que la digitada en el form, entonces hay una señal de login exitoso y redirecsionamiento a un dashboard ficticio. La importancia de que se presente el token de crsf para validar que cada vez que un usuario realice una petición es el servidor quien verifica que el token no cambió entre solicitudes.

Vulnerabilidad 2 y su código:

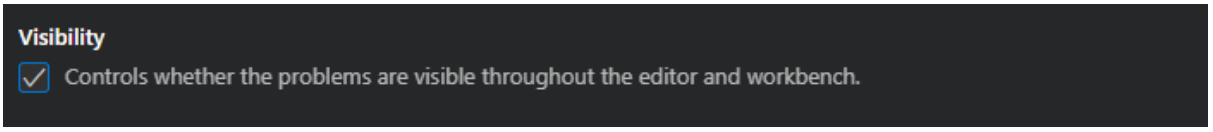
Teniendo en cuenta que la necesidad de gestionar marcadores para simular una consulta SQL en la validación usuario y contraseña, a sabiendas de que estamos haciendo alusión al primer entregable y el analisis de las tecnologías a utilizar se sobre entiende que no debe ser tan profundo, se decidió por crear una tabla sqlite4 paara los usuarios y contraseñas basica a los fines de dar a entender el funcionamiento del código. Teniendo en cuenta que permitir la inserción directa de valores de variables en las consultas SQL es uno de los principales factores de riesgos de las aplicaciones web actuales, el concatenar cadenas directamente y no reemplazar esa consulta por marcadores de posición deja expuesta a la aplicación a ser atacada por practicamente cualquier principiante que haya comenzado su camino en el pentesting por ser una de las vulnerabilidades que más presencia tiene en los ultimos años.

Conclusiones generales:

Reiterando la importancia del enfasis en la seguridad pero la simpleza de utilizar herramientas que poco tiempo se les dedico a su analisis, nos centramos en hacer entender la funcionalidad, en presentar código legible, bien comentado, por el cual se logre comprender como la presencia del mismo permite lograr que FitLife pueda ser un poco más seguro.

El código estará siendo actualizado a medida que se requiera en un repositorio publico de github de uno de los integrantes del grupo: (https://github.com/Test576M/FitLife)

Para visualizar los pseudocódigos se crearon archivos .py sin funcionalidad alguna más que utilizar los colores por defecto de la sintaxis de python, por lo cual se tuvo que deshabilitar:



Para correr en local el FitLife se necesita de instalar en la computadora: Python, Flask, sqlite4, flask_login, flask_wtf.

Tras correr el programa por Visual Studio Code, y para correr en local FitLife se necesita ingresar a : `http://127.0.0.1:5500/FitLife-Test/template/index.html`

Bibliografía:

https://realpython.com/prevent-python-sql-injection/

https://www.geeksforgeeks.org/csrf-protection-in-flask/

https://owasp.org/Top10/es/A03_2021-Injection/

https://owasp.org/Top10/es/A01_2021-Broken_Access_Control/

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

https://flask-wtf.readthedocs.io/en/0.15.x/api/#flask_wtf.FlaskForm