

Trabajo Integrador (Primer entregable)

DESARROLLO DE SOFTWARE SEGURO

Prof. Lic. Juan Pablo Villalba

Matias Leguizamon

Pablo Rodrigo Moya

Universidad de Gran Rosario

INDICE

Contexto del Proyecto.....	2
Nociones Preliminares.....	5
Analisis de Riesgos y Diseño Seguro.....	5
Vulnerabilidad 1 Pseudocódigo.....	6
Vulnerabilidad 1 Código.....	8
Vulnerabilidad 2 Pseudocódigo.....	10
Vulnerabilidad 2 Código.....	12
Conclusiones.....	14
Bibliografía.....	16

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 raíz de que pueden surgir cambios a lo largo de los próximos entregables y las consideraciones del profesor a cargo de la cátedra pueden variar respecto de la cantidad de entregables y/o el contenido de los mismos, a sabiendas de que el enfoque del trabajo está 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 fácil visualización del proyecto y testeado del mismo, el uso de Python por cuestiones de obligatoriedad, y el uso de importaciones de librerías o frameworks para gestionar las distintas tareas que se irán dando a lo largo y ancho de este trabajo.

Análisis 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 políticas 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 límites 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 dinámicas o no parametrizadas, afectando los datos almacenados, entre otros.

Vulnerabilidad 1

Pseudocódigo:

```
1  InicializarAplicacion()
2  # Inicializa la aplicación, configurando los parámetros y recursos necesario
3
4  establecerClaveSecreta('supersecretkey')
5  # Establece una clave secreta utilizada para la seguridad de la aplicación
6
7  habilitarProteccionCSRF()
8  # Habilita la protección contra ataques CSRF
9
10 configurarAplicacion(
11     DEBUG=True,
12     SECRET_KEY='supersecretkey')
13 # Configura la aplicación con parámetros específicos, como habilitar el modo de depuración
   (DEBUG) y establecer la clave secreta.
14
15 inicializarLoginManager()
16 # Inicializa el gestor de inicio de sesión, que maneja la autenticación de usuarios en la
   aplicación.
17
18 usuarios = diccionarioVacio()
19 # Crea un diccionario vacío para almacenar los usuarios registrados en la aplicación.
20
21 funcion principal():
22     mostrarPagina('index.html')
23     # Muestra la página principal de la aplicación.
```

```
24
25 def registrar():
26     username = obtenerFormulario('new_username')
27     password = obtenerFormulario('new_password')
28     # Obtiene el nombre de usuario y la contraseña del formulario de registro.
29
30     si username existe en usuarios:
31         mostrarMensaje('El usuario ya existe. Intenta con otro.')
32         # Verifica si el nombre de usuario ya existe en el diccionario de usuarios y muestra
33         # un mensaje si es así.
34
35     sino:
36         agregarUsuario(usuarios, username, password)
37         mostrarMensaje('Usuario registrado exitosamente. Ahora puedes hacer login.')
38         # Si el nombre de usuario no existe, lo agrega al diccionario de usuarios y muestra
39         # un mensaje de éxito.
40
41     redirigir('index')
42     # Redirige al usuario a la página principal después del registro.
43
44 def iniciarSesion():
45     username = obtenerFormulario('username')
46     password = obtenerFormulario('password')
47     # Obtiene el nombre de usuario y la contraseña del formulario de inicio de sesión.
48
49     si username existe en usuarios y usuarios[username] == password:
50         establecerSesion('username', username)
51         mostrarMensaje('¡Has iniciado sesión exitosamente!')
52         redirigir('dashboard')
53         # Verifica si el nombre de usuario existe y la contraseña es correcta, establece la
54         # sesión y redirige al panel de control.
55
56     sino:
57         mostrarMensaje('Usuario o contraseña incorrectos. Intenta nuevamente.')
58         redirigir('index')
59         # Si el nombre de usuario no existe o la contraseña es incorrecta, muestra un mensaje
60         # de error y redirige a la página principal.
61
62 si __nombre__ == '__main__':
63     ejecutarAplicacion(debug=True)
64     # Ejecuta la aplicación en modo de depuración si el script se ejecuta directamente.
```


Vulnerabilidad 1

Código en Python:

```
1  from flask import Flask, render_template, request, redirect, url_for, session, flash
2  from flask_login import LoginManager
3  from flask_wtf.csrf import CSRFProtect
4
5  app = Flask(__name__)
6  app.secret_key = 'supersecretkey' # Necesario para manejar sesiones y CSRF
7  csrf = CSRFProtect(app) # Habilitar protección CSRF
8
9  app.config.update(
10     DEBUG=True, # Es mala práctica dejar activado esto en producción.
11     SECRET_KEY="supersecretkey",)
12
13  login_manager = LoginManager()
14  login_manager.init_app(app)
15
16  # Diccionario temporal para almacenar usuarios (usuario: contraseña)
17  usuarios = {}
18
19  # Ruta principal para mostrar la página de registro/login
20  @app.route('/')
21  def index():
22      return render_template('index.html')
23      # Renderiza la página principal de la aplicación, que contiene los formularios de
24      # registro y login.
25
26  # Ruta para el registro
27  @app.route('/register', methods=['POST'])
28  def register():
29      username = request.form['new_username']
30      password = request.form['new_password']
31      # Obtiene el nombre de usuario y la contraseña del formulario de registro.
32
33      if username in usuarios:
34          flash('El usuario ya existe. Intenta con otro.')
35          # Verifica si el nombre de usuario ya existe en el diccionario de usuarios y muestra
36          # un mensaje si es así.
37      else:
38          usuarios[username] = password
39          flash('Usuario registrado exitosamente. Ahora puedes hacer login.')
40          # Si el nombre de usuario no existe, lo agrega al diccionario de usuarios y muestra
41          # un mensaje de éxito.
42
43      return redirect(url_for('index'))
44      # Redirige al usuario a la página principal después del registro.
```

```

42
43 # Ruta para el login
44 @app.route('/login', methods=['POST'])
45 def login():
46     username = request.form['username']
47     password = request.form['password']
48     # Obtiene el nombre de usuario y la contraseña del formulario de inicio de sesión.
49
50     if username in usuarios and usuarios[username] == password:
51         session['username'] = username
52         flash('¡Has iniciado sesión exitosamente!')
53         return redirect(url_for('dashboard'))
54     # Verifica si el nombre de usuario existe y la contraseña es correcta, establece la
55     # sesión y redirige al panel de control.
56     else:
57         flash('Usuario o contraseña incorrectos. Intenta nuevamente.')
58         return redirect(url_for('index'))
59     # Si el nombre de usuario no existe o la contraseña es incorrecta, muestra un mensaje
60     # de error y redirige a la página principal.
61
62 if __name__ == '__main__':
63     app.run(debug=True)
64     # Ejecuta la aplicación en modo de depuración si el script se ejecuta directamente.
65

```

Código HTML:

```

1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Integrador</title>
7 </head>
8 <body>
9     <h1>Universidad de Gran Rosario (Virtual)</h1>
10    <h2>Tecnatura en Ciberseguridad</h2>
11    <h2>Matias Leguizamon - Pablo Rodrigo Moya </h2>
12    <h3>Trabajo Integrador</h3>
13    <!-- Formulario de Register -->
14    <form action='/register' method="POST" autocomplete="off">
15        <input type="text" name="new_username" placeholder="Nuevo Usuario">
16        <input type="password" name="new_password" placeholder="Contraseña">
17        <button type="submit" name="submit" value="register"> Registrar </button>
18        <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
19        <!-- En realidad como el template maneja flask esta ultima linea se puede simplificar
20        como {{ form.csrf_token }} y listo-->
21    </form>
22
23    <!-- Formulario de Login -->
24    <form action="/login" method="POST" autocomplete="off">
25        <input type="text" name="username" placeholder="Usuario">
26        <input type="password" name="password" placeholder="Contraseña">
27        <button type="submit" name="submit" value="register"> Iniciar Sesión </button>
28        <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
29        <!-- En realidad como el template maneja flask esta ultima linea se puede simplificar
30        como {{ form.csrf_token }} al igual que el de register-->
31    </form>
32 </body>
33 </html>

```

Vulnerabilidad 2

Pseudocódigo:

```
1  INICIAR aplicación
2  # Inicia la aplicación principal
3
4  CONFIGURAR clave_secreta = 'supersecretkey'
5  # Configura una clave secreta para la aplicación, utilizada para firmar cookies y otros datos
6
7  HABILITAR protección CSRF
8  # Habilita la protección contra ataques CSRF (Cross-Site Request Forgery)
9
10 CONFIGURAR aplicación:
11     DEBUG = verdadero
12     # Activa el modo de depuración para la aplicación, útil durante el desarrollo
13     SECRET_KEY = "supersecretkey"
14     # Establece la clave secreta para la aplicación
15
16 INICIAR gestor_de_login
17 # Inicia el gestor de inicio de sesión para manejar la autenticación de usuarios
18
19 CLASE Usuario:
20     FUNCIÓN __init__(nombre_usuario, contraseña):
21         ASIGNAR nombre_usuario
22         # Asigna el nombre de usuario al atributo de la clase
23         ASIGNAR contraseña
24         # Asigna la contraseña al atributo de la clase
25
26     FUNCIÓN obtener_id():
27         RETORNAR nombre_usuario
28         # Retorna el nombre de usuario como identificador único
29
30     FUNCIÓN cargar_usuario(nombre_usuario):
31         EJECUTAR consulta SQL 'SELECT * FROM usuarios WHERE username = nombre_usuario'
32         # Ejecuta una consulta SQL para buscar un usuario por su nombre de usuario
33         SI usuario encontrado:
34             RETORNAR Usuario(nombre_usuario, contraseña)
35             # Si se encuentra el usuario, retorna una instancia de la clase Usuario
36         SINO:
37             RETORNAR nulo
38             # Si no se encuentra el usuario, retorna nulo
39
40 CONECTAR a base_de_datos "usuarios.db"
41 # Conecta a la base de datos llamada "usuarios.db"
42
43 EJECUTAR consulta SQL '''
44     CREATE TABLE IF NOT EXISTS usuarios (
45         username TEXT PRIMARY KEY,
46         password TEXT NOT NULL
47     )
48     ...
49 # Ejecuta una consulta SQL para crear la tabla "usuarios" si no existe, con columnas para el
50 # nombre de usuario y la contraseña
51
52 RUTA '/login' MÉTODO POST
53 # Define una ruta para el inicio de sesión que acepta solicitudes POST
```

```
54 FUNCIÓN iniciar_sesión():
55     OBTENER nombre_usuario de formulario
56     # Obtiene el nombre de usuario del formulario de inicio de sesión
57     OBTENER contraseña de formulario
58     # Obtiene la contraseña del formulario de inicio de sesión
59
60     EJECUTAR consulta SQL 'SELECT * FROM usuarios WHERE username = nombre_usuario AND
61     password = contraseña'
62     # Ejecuta una consulta SQL para verificar las credenciales del usuario
63     SI usuario encontrado:
64         CREAR objeto_usuario Usuario(nombre_usuario, contraseña)
65         # Si se encuentra el usuario, crea un objeto de la clase Usuario
66         INICIAR sesión con objeto_usuario
67         # Inicia sesión con el objeto de usuario
68         MOSTRAR mensaje '¡Has iniciado sesión exitosamente!'
69         # Muestra un mensaje de éxito
70         REDIRIGIR a 'tablero'
71         # Redirige al usuario al tablero principal
72     SINO:
73         MOSTRAR mensaje 'Usuario o contraseña incorrectos. Intenta nuevamente.'
74         # Muestra un mensaje de error si las credenciales son incorrectas
75         REDIRIGIR a 'índice'
76         # Redirige al usuario a la página de inicio
77
78 SI __name__ == '__main__':
79     EJECUTAR aplicación en modo debug
80     # Ejecuta la aplicación en modo de depuración si este archivo es el principal
```

Vulnerabilidad 2

Código:

```
1  from sqlite4 import SQLite4
2  from flask import Flask, render_template, request, redirect, url_for, session, flash
3  from flask_login import LoginManager, UserMixin, login_user
4  from flask_wtf.csrf import CSRFProtect
5
6  app = Flask(__name__) # Crear una instancia de la aplicación Flask
7  app.secret_key = 'supersecretkey' # Necesario para manejar sesiones y CSRF
8  csrf = CSRFProtect(app) # Habilitar protección CSRF
9
10 app.config.update( # Configuración de la aplicación
11     DEBUG=True, # Es mala práctica dejar activado esto en producción.
12     SECRET_KEY="supersecretkey",
13 )
14
15 # Inicializar el gestor de inicio de sesión
16 login_manager = LoginManager()
17 login_manager.init_app(app)
18
19 class User(UserMixin):
20     def __init__(self, username, password):
21         self.username = username
22         self.password = password
23
24     def get_id(self):
25         return self.username
26
27 @login_manager.user_loader
28 def load_user(username):
29     database.execute('SELECT * FROM usuarios WHERE username = ?', (username,))
30     user = database.fetchone()
31     if user:
32         return User(username=user[0], password=user[1])
33     return None
34
35 # Conexión a la base de datos SQLite4
36 database = SQLite4(["usuarios.db"])
37 database.connect()
38
```

```
38
39 # Crear tabla de usuarios si no existe
40 database.execute('''
41 CREATE TABLE IF NOT EXISTS usuarios (
42     username TEXT PRIMARY KEY,
43     password TEXT NOT NULL
44 )
45 ''')
46
47 # Ruta principal para mostrar la página de registro/login
48 @app.route('/')
49 def index():
50     return render_template('index.html')
51     # Renderiza la página principal de la aplicación, que contiene los formularios de
52     # registro y login.
53
54 # Ruta para el registro
55 @app.route('/register', methods=['POST'])
56 def register():
57     username = request.form['new_username']
58     password = request.form['new_password']
59     # Obtiene el nombre de usuario y la contraseña del formulario de registro.
60
61     # Verificar si el usuario ya existe
62     database.execute('SELECT * FROM usuarios WHERE username = ?', (username,))
63     if database.fetchone():
64         flash('El usuario ya existe. Intenta con otro.')
65     else:
66         # Insertar el nuevo usuario en la base de datos
67         database.execute('INSERT INTO usuarios (username, password) VALUES (?, ?)',
68                         (username, password))
69         database.execute('COMMIT')
70         flash('Usuario registrado exitosamente. Ahora puedes hacer login.')
71
72     return redirect(url_for('index'))
73     # Redirige al usuario a la página principal después del registro.
```

```
71 | # Redirige al usuario a la página principal después del registro.
72 |
73 | # Ruta para el login
74 | @app.route('/login', methods=['POST'])
75 | def login():
76 |     username = request.form['username']
77 |     password = request.form['password']
78 |     # Obtiene el nombre de usuario y la contraseña del formulario de inicio de sesión.
79 |
80 |     # Verificar las credenciales del usuario
81 |     database.execute('SELECT * FROM usuarios WHERE username = ? AND password = ?', (username,
82 |     password))
83 |     user = database.fetchone()
84 |     if user:
85 |         user_obj = User(username=user[0], password=user[1])
86 |         login_user(user_obj)
87 |         flash('¡Has iniciado sesión exitosamente!')
88 |         return redirect(url_for('dashboard'))
89 |     else:
90 |         flash('Usuario o contraseña incorrectos. Intenta nuevamente.')
91 |         return redirect(url_for('index'))
92 |
93 | if __name__ == '__main__':
94 |     app.run(debug=True)
95 | # 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 librerías, 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 crear un diccionario provisorio a los fines de poder guardar los usuarios, se define la ruta de /register donde el formulario en HTML debería usar los métodos mencionados en la aplicación.py, se crea ruta /login para hacer exactamente lo mismo que se realizó anteriormente con register, y se plantea una validación básica 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 redireccionamiento a un dashboard ficticio. La importancia de que se presente el token de csrf 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 análisis 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 énfasis en la seguridad pero la simpleza de utilizar herramientas que poco tiempo se les dedico a su análisis, 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:

Visibility



Controls whether the problems are visible throughout the editor and workbench.

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