Trabajo Integrador (segundo entregable)

DESARROLLO DE SOFTWARE SEGURO

Prof. Lic. Juan Pablo Villalba

Matias Leguizamon

Pablo Rodrigo Moya

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.
Teniendo en cuenta como el enfoque inicial para la resolución de este trabajo contemplaba la realización de código funcional para poder llevar adelante ciertos tipos de pruebas respecto a las dos vulnerabilidades elegidas (OWASP A01:2021 – Pérdida de Control de Acceso / CWE-352 Cross-Site Request Forgery (CSRF) & OWASP A03:2021 – Inyección), a sabiendas de que el adelantarnos al siguiente entregable podría conllevar implicaciones como la dificultad de realizar las actividades solicitadas sin ser repetitivos y/o no agregar material de valor para el trabajo en su conjunto, resolvimos: Agregar un control más al sistema de registro y login, recordando la importancia de sanitizar las entradas de usuario, y como al finalizar este entregable se reflexionará sobre la misma.

Pseudocódigo:

```
> pseudocodigo-entregables > entregable2.py > Pseudocodigo-xss-sanit.py
IMPORTAR "re"

DEFINIR validar_username(username) y validar_password (password) usando regex

MODIFICAR ruta /register con los agregados de validaciones
if not validar_username(username):
return "alerta usuario no cumple requisitos"
if not validar_password(password):
return "alerta password no cunple requisitos"

MODIFICAR ruta /login con los agregados de validacionesif not validar_username(username):
return "alerta usuario no cumple requisitos"
if not validar_password(password):
return "alerta password no cunple requisitos"
```

Código:

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


#2 definimos las nuevas validaciones de usuario y password
def validar_username(username):
    #2 El nombre de usuario debe tener entre 5 y 18 caracteres alfanuméricos
    regex = r'^[a-zA-Z0-9]{5,18}$'
    return re.match(regex, username)

def validar_password(password):
    #2 La contraseña debe tener entre 6 y 18 caracteres, al menos una letra mayúscula, una minúscula, un número y un carácter especial
    regex = r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{6,18}$'
    return re.match(regex, password)
```

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

    #2 Validar el nombre de usuario
    if not validar_username(username):
        flash('El nombre de usuario debe tener entre 3 y 20 caracteres alfanuméricos.')
        return redirect(url_for('index'))

    #2 Validar la contraseña
    if not validar_password(password):
        flash('La contraseña debe tener entre 6 y 20 caracteres, al menos una letra
        mayúscula, una minúscula, un número y un carácter especial.')
        return redirect(url_for('index'))
```

```python
#1 Ruta para el login
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']

    #2 Validar el nombre de usuario
    if not validar_username(username):
        flash('Nombre de usuario inválido.')
        return redirect(url_for('index'))

    #2 Validar la contraseña
    if not validar_password(password):
        flash('Contraseña inválida.')
        return redirect(url_for('index'))
```

Conclusiones generales:

Las vulnerabilidades de Cross-Site Scripting (XSS) representan una amenaza significativa para la seguridad de las aplicaciones web, permitiendo a los atacantes inyectar y ejecutar código malicioso en los navegadores de los usuarios. La sanitización de las entradas de usuarios es una medida crucial para prevenir estos ataques, ya que garantiza que los datos ingresados sean validados y filtrados adecuadamente antes de ser procesados. Al priorizar la sanitización de entradas, se fortalece la seguridad de las aplicaciones y se protege la integridad y la confianza de los usuarios, en este trabajo por ejemplo se utiliza la ayuda de la libreria “re” para poder llevar a cabo esta validación de expresiones regulares.

Se adeuda: mejora de caratura, orden de carpetas de github para facil comprensión y seguimiento del orden de los entregables

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

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`

Resumen lectura obligatoria:

Respecto al material obligatorio de lectura: Los equipos de seguridad informatica tienen que estar presentes desde los inicios del ciclo de vida de una aplicación, que el trabajo comience tarde conlleva aa muchas deudas de seguridad, y gran parte de las veces replantearse la arquitectura de una o muchas partes del sistema.

Las buenas practicas de desarrollo seguro son mucho más importantes que centrarse en herramientas de escaneo reactivas. La base para mitigar gastos esta en “shifting left” que es basicamente la participación de los equipos de seguridad desde (y me repito) el comienzo del ciclo de vida de una aplicación, donde arreglar errores que etapas tardias o mismo en ambiente de producción conlleva indudablemente a mayores gastos para la empresa en materia economica y tambien en ambitos poco tangibles como reputación y confianza.

Bibliografía:

https://docs.python.org/3/library/re.html y edge IA (nadie se sabe un regex de memoria)

Application Security Program Handbook pág. 3 a 28