Trabajo Integrador (tercer 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

Centrandonos en el uso de librerías de manejo de cookies de session, haciendo enfasis en la duración de las sesiones, la perpetuabilidad de la misma, reconociendo la necesidad de almacenar una cookie del lado del servidor / base de datos a los fines de evitar lo mejor posible la cantidad de datos que puedan manipularse desde el lado del cliente, recordando la diferenciación entre local storage y sessión storage, repitiendo por cuestiones de necesidad, comodidad, y practicidad en matería de facilidad, y seguridad, el uso de las herramientas

creadas al rededor del ecosistema Python - Flask, es por lo cual nos decantamos en la confección para su adición final, hasta que las circunstancias y/o las consignas lo requieran, del código a continuación.

Pseudocódigo:

```
IMPORTAR de "datetime" - "timedelta"
IMPORTAR de "flask-session" - "session"

AGREGAR en app.config.update(
    SESSION type = sqlalchemy
    SESSION_ALCHEMY = usar version SQLite4 ("session.db")
    SESSION_LIFETIME = timedelta ( 30 minutos)
    SESSION_COOKIE = solo HTTP
)

INICIALIZAR flask_session en la APP

#en ruta /login
IF user:
    session.clear() #para regenerar id de session
    session.permanent = true #para al salir de la pestaña se guarde la sesion
```

Código:

```
from flask_session import Session
from datetime import timedelta
```

```
app.config.update(  #1 Configuración de la aplicación
    DEBUG=True,  #1 Es mala práctica dejar activado esto en producción.
    SECRET_KEY="supersecretkey",
    SESSION_TYPE='sqlalchemy',  #3 Tipo de almacenamiento de sesión
    SESSION_SQLALCHEMY=SQLite4("sessions.db"),  #3 Base de datos para almacenar sesiones
    PERMANENT_SESSION_LIFETIME=timedelta(minutes=30)  #3 Duración de la sesión
    SESSION_COOKIE_HTTPONLY=True  #3 Evita el acceso a la cookie desde JavaScript
```

```

#3 Inicializar Flask-Session - para que las cookies se trabajen
#  en el servidor / backend - en este caso SQLite
Session(app)

```

```
    #1 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])
        session.clear() #3 Regenerar la ID de session para mitigar session fixation
        login_user(user_obj)
        session.permanent = True  #3 Marcar la sesión como permanente
        flash('¡Has iniciado sesión exitosamente!')
        return redirect(url_for('dashboard'))
    else:
        flash('Usuario o contraseña incorrectos. Intenta nuevamente.')
        return redirect(url_for('index'))

```

Extra:

```
#3 voy a colocar un ejemplo de como podria mitigarse el riesgo a recibir un ataque
session hijacking:
#3 en app.config.update (SESSION_COOKIE_SECURE=True) -- esto hace que las cookies solo se
envien a travez de HTTPS
#3 if __name__ == '__main__':
#3 app.run(ssl_context=('path/to/cert.pem', 'path/to/key.pem'), debug=True)
#3 Aca lo que se busca es especificar los archivos de certificado y la clave, cosa con la
que no cuento, pero es una opción viable a plantear.

```

Conclusiones:

Normalmente las cookies de las sesiones se pueden trabajar de dos formas, una es aquella sesion donde la cookie se guarda en el navegador del usuario, conteniendo datos firmados que Flask podría verificar y utilizar para identificar al usuario; la segunda forma es aquella donde la cookie se maneja desde el servidor, en nuestro caso Flask esta configurado para usar el backend de SQLite como almacenamiento de sesiones.

En esta entrega priorizamos distintos ejes: La sesion tiene que ser persistente, para que al momento de cerrar la pestaña y volver a abrirla no se deslogee automaticamente; La duración de la sesion configurada en el servidor es de 30 minutos, pasado el tiempo pre-establecido la sesión se cierra; El servidor como controlador total de la sesión, disminuyendo la exposición al riesgo de manipuliación por parte del usuario a las cookies que se hubiesen manejado desde su navegador; Evitar que se explote la vulnerabilidad de

tipo sesión Hijacking, a modo de comentario se agregó al final del código una posibilidad de plantearse a futuro, la cual se desistió su uso practico en esta entrega debido a la no poseción de un certificado HTTPS; Evitar que se explote la vulnerabilida de tipo Session Fixation, regenerando la ID de sesión luego de haber iniciado sesión para almacenarla en el servidor con la información nueva.

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:

La importancia del manejo de riesgos es superior a muchas otras áreas, de notable interes para muchos, pero sin establecer una versión especifica para la organización de risk management o utilizar alguna de las ya conocidas NIST RMF, la creación una supply chain security se vera afectada a la larga o a la corta por las malas decisiones del comienzo.

Generalmente luego de haber establecido un plan de risk management para la empresa podemos apreciar como los estandares y frameworks se van adquiriendo al rededor de la supply chain. En el texto se remarca una diferencia entre como los frameworks son gratis y los estandares tienen que ser adquiridos por grandes sumas de dinero.

A lo largo del capitulo 2 se pueden apreciar distintos frameworks y standards, con las respectivas imagenes donde resumen los objetivos, metodologías de trabajo, lo que se busca realizar/lograr, a que áreas se centra cada parte del conjunto de normas o reglas, el poder entender que cada empresa es diferente, a sabiendas de que hay ciertas categorías de empresas respecto al área a la que se dedican, es por lo cual que hay que tener el consideración que se debe hacer un analisis exhaustivo para identificar los puntos más

importantes a mitigar/monitorear/prevenir, el adquirir conocimientos sobre los distintos frameworks y estándares facilita la tarea tanto para poder implementar de principio a fin uno de ellos, como para ensamblar una versión especifica con las necesidades particulares de la empresa en cuestión.

Bibliografía:

welivesecurity.com/la-es/2013/07/01/cookies-aplicaciones-web-diseno-vulnerabilidades/

Session Management - OWASP Cheat Sheet Series

Securing the software supply chain - chapter 2