# 1.d. Pruebas de Consentimiento (Hacking Ético / Pentesting)

## 1.d.1. Introducción a las pruebas de consentimiento

Las pruebas de consentimiento constituyen una fase crítica dentro de la ejecución de una auditoría informática, ya que permiten evaluar el nivel real de exposición de una aplicación ante ataques técnicos. Este tipo de pruebas se realiza únicamente sobre ambientes controlados, con autorización previa y bajo los principios del Hacking Ético, con el objetivo de identificar vulnerabilidades que pudieran comprometer la confidencialidad, integridad o disponibilidad del sistema auditado.

Para la presente auditoría, se empleó un entorno aislado de pruebas que replica condiciones y vulnerabilidades comunes en aplicaciones web modernas. Este ambiente controlado permite ejecutar ataques sin afectar sistemas productivos, siguiendo las buenas prácticas establecidas por:

* OWASP Testing Guide v4
* OWASP Application Security Verification Standard (ASVS)
* ISO/IEC 27001:2022 – Controles A.5, A.8, A.12, A.13 y A.14
* ISO/IEC 19011 – Directrices para auditoría de sistemas de gestión

El objetivo principal de estas pruebas es determinar si los controles implementados por la aplicación son suficientes para mitigar riesgos asociados a ataques habituales, tales como:

* Inyección de código
* Manipulación de sesiones
* Fuerza bruta y bypass de autenticación
* Exposición indebida de información
* Configuraciones inseguras
* Falta de validación de entradas
* Vulnerabilidades propias del OWASP Top 10

Las pruebas se desempeñaron bajo los principios de:

1. Legalidad — Ejecución únicamente en sistemas autorizados
2. Ética — Sin afectar datos reales o infraestructura operativa
3. Repetibilidad — Documentando cada procedimiento
4. Objetividad — Entregando evidencia verificable

El alcance técnico aprobado incluye:

* Validación de mecanismos de autenticación
* Evaluación de gestión de sesiones y tokens
* Análisis de encabezados de seguridad
* Pruebas de inyección y validación de entrada
* Evaluación de controles anti-automatización y rate limiting
* Exploración de endpoints accesibles y rutas expuestas
* Simulación de ataques de escalamiento de privilegios y exposición de datos

Los resultados obtenidos de estas pruebas permiten establecer el nivel de madurez de seguridad de la aplicación, identificar fallas en los controles existentes y generar hallazgos de auditoría respaldados con evidencia técnica. Todas las actividades se registraron mediante papeles de trabajo, los cuales incluyen procedimientos, capturas, logs, outputs de herramientas y conclusiones del auditor técnico.

## 1.d.2. Metodología aplicada

Para la ejecución de las pruebas de consentimiento se adoptó una metodología estructurada basada en estándares internacionales de auditoría y seguridad de aplicaciones. Esta metodología garantiza que las pruebas se realicen de forma ética, controlada, repetible y con un alcance definido, permitiendo obtener evidencia suficiente y apropiada para sustentar los hallazgos identificados.

La metodología empleada se compone de las siguientes fases:

### 1. Planeación técnica

En esta fase se estableció:

* El alcance de las pruebas
* Los límites permitidos
* Las herramientas aprobadas
* El ambiente controlado de pruebas
* Los criterios de aceptación y confidencialidad

Se confirmó que las pruebas **no afectarían sistemas productivos**, y que toda la ejecución se realizaría bajo entornos aislados autorizados.

### 2. Reconocimiento Pasivo

Objetivo: recopilar información sin interactuar directamente con la aplicación.

Incluye:

* Fingerprinting de tecnologías (headers, frameworks, versiones)
* Identificación de puertos expuestos
* Descubrimiento de componentes del stack
* Revisión de configuraciones de respuesta HTTP
* Análisis del uso de TLS y certificados

Herramientas: *curl, WhatWeb, Wappalyzer, Nmap (sin flags intrusivos), DevTools*.

### 3. Reconocimiento Activo

Tiene como objetivo descubrir la estructura interna de la aplicación.

Actividades realizadas:

* Enumeración de endpoints accesibles
* Identificación de parámetros vulnerables
* Detección de formularios de entrada
* Identificación de rutas administrativas
* Pruebas de error-handling y mensajes expuestos

Herramientas: *OWASP ZAP Spider, Burp Suite Repeater, Nmap avanzado*.

### 4. Enumeración y Mapeo Funcional

Se analizó la funcionalidad para comprender:

* Flujos de autenticación
* Gestión de sesiones
* Roles de usuario
* Parámetros sensibles
* Lógica de negocio expuesta
* Validaciones del lado cliente y del lado servidor

Esto sirve para identificar áreas críticas susceptibles a explotación.

### 5. Ejecución de Pruebas OWASP Top 10

Las pruebas se basaron en los controles del **OWASP Testing Guide v4**, abordando las vulnerabilidades más comunes:

* A01 – Broken Access Control
* A02 – Cryptographic Failures
* A03 – Injection (SQLi, XSS)
* A04 – Insecure Design
* A05 – Security Misconfiguration
* A07 – Identification and Authentication Failures
* A08 – Software and Data Integrity Failures

Cada prueba fue documentada en un papel de trabajo independiente (PT-01 a PT-08).

### 6. Pruebas de Autenticación y Gestión de Sesiones

Validaciones realizadas:

* Resistencia a fuerza bruta
* Reset de contraseñas
* Reutilización de tokens
* Expiración de sesión
* Correcto uso de atributos de cookies:
  + *HttpOnly*
  + *Secure*
  + *SameSite*
  + *Path*
* Regeneración de sesión en login/logout

### 7. Pruebas de Manipulación de Datos y Entradas

Incluyen:

* Inyección SQL simulada
* Inyección de scripts (XSS almacenado y reflejado)
* Manipulación de parámetros en URL
* Envío de payloads especialmente diseñados
* Validación insuficiente del lado servidor

Herramientas: *Burp Suite, ZAP, XSS CheatSheet Payloads*.

### 8. Pruebas de Consentimiento de Alto Riesgo

Realizadas únicamente en ambiente controlado:

* CSRF no protegido
* Disclosure de información sensible
* Acceso directo a objetos (IDOR)
* Escalación horizontal o vertical de privilegios
* Manipulación de JWT o tokens inseguros

### 9. Análisis de Configuraciones del Servidor

Revisión de:

* Headers de seguridad
* Versiones expuestas
* CORS
* Directorios sin protección
* Archivos sensibles accesibles
* Políticas de caché
* Configuración HTTPS/TLS

Herramientas: *curl, SSL Labs (local), ZAP Passive Scan*.

### 10. Registro, Evidencia y Documentación

Cada prueba realizada generó un **Papel de Trabajo (PT)** que documenta:

* Objetivo
* Procedimiento paso a paso
* Herramienta utilizada
* Capturas de pantalla
* Resultados
* Log de la prueba
* Conclusión técnica
* Nivel de riesgo
* Recomendación OWASP/ISO

Los papeles de trabajo se almacenaron siguiendo criterios de:

* Trazabilidad
* Integridad
* Reproducibilidad
* Organización lógica

### 11. Análisis de Riesgo Posterior

Los resultados se mapearon contra:

* Controles ISO 27001
* OWASP ASVS
* Riesgos inherentes del sistema
* Impacto operativo simulado
* Probabilidad y severidad

### 12. Determinación de Hallazgos

Con base en la evidencia técnica:

* Se identificaron vulnerabilidades verificadas
* Se clasificaron según severidad
* Se alinearon con criterios de auditoría
* Se prepararon para el Informe Final

## 1.d.3. Herramientas utilizadas

Durante la ejecución de las pruebas de consentimiento se utilizaron únicamente herramientas aprobadas en la fase de planificación técnica (ver sección 1.a), aplicándolas de forma específica a los escenarios de prueba definidos. Estas herramientas permitieron identificar vulnerabilidades, registrar evidencia técnica y validar los controles implementados en la aplicación.

Las herramientas aplicadas en esta fase fueron:

**OWASP ZAP (Zed Attack Proxy)**

Utilizado para:

* Escaneo pasivo
* Enumeración de endpoints
* Análisis de respuestas HTTP
* Identificación preliminar de vulnerabilidades
* Captura y manipulación de solicitudes

**Burp Suite (Community)**

Aplicada en:

* Repeater para manipulación manual de parámetros
* Validación de payloads de XSS
* Pruebas de autenticación y sesiones

**Nmap**

Utilizado para:

* Identificación de puertos expuestos
* Fingerprinting de servicios
* Validación de configuraciones del servidor

**curl / OpenSSL / DevTools**

Herramientas de línea de comando y navegador aplicadas en:

* Revisión de encabezados de seguridad
* Validación de cookies
* Inspección de tráfico HTTPS
* Pruebas manuales de endpoints

**Scripts y utilidades complementarias**

Usados para:

* Generación de payloads
* Decodificación y validación de tokens
* Automatización de pruebas repetitivas

Estas herramientas se seleccionaron debido a su amplia aceptación en auditorías técnicas, su trazabilidad y su capacidad para generar evidencia verificable, permitiendo documentar los resultados de cada prueba en los respectivos papeles de trabajo.

## 1.d.4. Resultados de las pruebas

Las pruebas de consentimiento permitieron identificar vulnerabilidades, debilidades en controles aplicados y configuraciones deficientes dentro del entorno evaluado. Las actividades realizadas incluyeron pruebas manuales y automatizadas, siguiendo técnicas del OWASP Testing Guide v4 y criterios del OWASP ASVS. Los resultados obtenidos proporcionan evidencia suficiente, competente y adecuada para determinar el nivel de exposición de la aplicación ante ataques comunes que afectan aplicaciones web modernas.

Los hallazgos observados durante la ejecución de las pruebas pueden agruparse en las siguientes categorías:

### 1. Vulnerabilidades de Validación de Entradas

Durante las pruebas se identificaron escenarios donde la aplicación permite la ejecución de código malicioso o manipulación de parámetros debido a una validación insuficiente del lado servidor.

**Vulnerabilidades encontradas:**

* **XSS reflejado** en campos de entrada sin sanitización.
* **XSS almacenado**, ejecutándose código en secciones posteriores.
* **Manipulación de parámetros** sin controles que limiten contenido no autorizado.

**Riesgo asociado:**

* Robo de sesiones
* Modificación del DOM
* Phishing interno
* Secuestro de cuentas
* Defacement del sitio

### 2. Debilidades en Autenticación y Controles Anti-Automatización

Las pruebas realizadas sobre el mecanismo de autenticación mostraron ausencia de controles esenciales que mitigan ataques automatizados.

**Resultados:**

* **Fuerza bruta posible:** el sistema no implementa rate limiting.
* **Mensajes de error diferenciados**, facilitando enumeración de usuarios.
* **Falta de MFA**, aun para accesos con impacto crítico.

**Riesgo asociado:**

* Compromiso de credenciales
* Ataques de diccionario y credential stuffing
* Exposición de usuarios válidos

### 3. Exposición de Información Sensible

Las configuraciones del servidor permiten revelar información innecesaria al cliente, lo cual incrementa la superficie de ataque.

**Evidencia detectada:**

* Encabezados que exponen versión del framework y del servidor.
* Respuestas de error que devuelven información del stack interno.
* Endpoints accesibles sin autenticación, mostrando datos operativos.

**Riesgo asociado:**

* Reconocimiento para ataques dirigidos
* Ingeniería inversa sobre el comportamiento del sistema
* Aprovechamiento de vulnerabilidades específicas según versión

### 4. Falta de Controles Adecuados en Gestión de Sesiones

Se evaluó la seguridad de cookies y tokens generados por el sistema.

**Resultados:**

* Cookies sin atributos **HttpOnly**, **Secure** o **SameSite**.
* Tokens sin expiración adecuada o regeneración tras login/logout.
* Posibilidad de reutilización de tokens capturados.

**Riesgo asociado:**

* Secuestro de sesión
* Suplantación de identidad
* Acceso no autorizado sostenido

### 5. Configuraciones Inseguras del Servidor

El análisis de configuraciones mostró ausencia de mecanismos de protección a nivel de infraestructura.

**Observaciones:**

* Falta de políticas CSP, HSTS, X-Frame-Options y X-Content-Type-Options.
* Respuestas HTTP sin directivas de seguridad relevantes.
* Errores 500 y 404 revelando estructura interna del sistema.

**Riesgo asociado:**

* Clickjacking
* Downgrade attacks
* Mixed content
* Exposición involuntaria de archivos o rutas internas

### 6. Enumeración de Endpoints y Falta de Restricción de Accesos

Las herramientas de spidering identificaron rutas accesibles que no deberían ser públicas.

**Resultados principales:**

* Endpoints administrativos detectados sin autenticación previa.
* Rutas internas retornando respuestas completas.
* Parámetros alterables para obtener más información de la necesaria.

**Riesgo asociado:**

* IDOR (Insecure Direct Object Reference)
* Acceso no autorizado a datos
* Exposición de estructuras internas de la aplicación

**Conclusión General de las Pruebas**

Las pruebas realizadas demuestran que la aplicación evaluada presenta **múltiples vulnerabilidades críticas**, principalmente relacionadas con:

* Sanitización insuficiente de entradas
* Gestión insegura de sesiones
* Ausencia de controles anti-automatización
* Configuraciones deficientes a nivel de servidor
* Exposición excesiva de información

Estos hallazgos serán documentados en detalle dentro de los **Papeles de Trabajo (PT-01 al PT-08)**, donde se incluyen evidencias técnicas, capturas, análisis, criterios de auditoría y recomendaciones específicas.

## 1.d.5. Listado de pruebas ejecutadas

Las pruebas de consentimiento fueron realizadas siguiendo el *OWASP Testing Guide v4*, abarcando las principales categorías de vulnerabilidades que afectan aplicaciones web modernas. El siguiente checklist resume cada una de las pruebas ejecutadas, el objetivo de la prueba, y el resultado obtenido durante la evaluación del entorno controlado.

### Checklist de Pruebas Realizadas (Basado en OWASP Testing Guide v4)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID Prueba (OWASP)** | **Descripción de la Prueba** | **Objetivo** | **Resultado** | **Severidad** |
| **OTG-INFO-001** | Fingerprinting del servidor y tecnologías | Identificar stack tecnológico y superficie de ataque | ✔ Vulnerabilidades detectadas en headers y versiones visibles | Medio |
| **OTG-INFO-002** | Enumeración de endpoints | Descubrir rutas accesibles y no controladas | ✔ Endpoints expuestos y sin autenticación | Alto |
| **OTG-AUTHN-001** | Bypass de autenticación | Intentar acceder sin credenciales a áreas privadas | ✔ Acceso parcial obtenido | Alto |
| **OTG-AUTHN-002** | Fuerza bruta | Evaluar ausencia de rate limiting | ❌ Vulnerable; no existe bloqueo automático | Alto |
| **OTG-SESS-001** | Análisis de cookies y tokens | Verificar atributos de seguridad (HttpOnly, Secure, SameSite) | ❌ Cookies y tokens inseguros | Crítico |
| **OTG-SESS-002** | Fijación de sesión | Validar si la sesión se regenera tras autenticación | ❌ No se regenera, permite session fixation | Alto |
| **OTG-INPVAL-001** | XSS reflejado | Validar sanitización de entradas | ❌ Vulnerable a XSS | Alto |
| **OTG-INPVAL-002** | XSS almacenado | Validar persistencia de payloads maliciosos | ❌ Vulnerabilidad confirmada | Crítico |
| **OTG-INPVAL-005** | SQL Injection (simulada) | Evaluar validación de parámetros y consultas | ✔ Inyección parcial posible en parámetros | Alto |
| **OTG-CONFIG-001** | Security Misconfigurations | Revisar headers, políticas, versiones expuestas | ❌ Headers críticos ausentes (CSP, HSTS, etc.) | Alto |
| **OTG-CONFIG-002** | Directory Browsing | Intentar acceso no autorizado a archivos internos | ✔ Directorios accesibles | Medio |
| **OTG-ERR-001** | Manejo de errores | Confirmar si el servidor expone información sensible | ❌ Exposición de mensajes de error detallados | Alto |
| **OTG-CRYPST-001** | Validación de HTTPS/TLS | Revisión de certificados y cifrado | ✔ Implementado, pero configurable | Bajo |
| **OTG-BUSLOGIC-001** | Manipulación de parámetros | Verificar accesos no autorizados por lógica de negocio | ❌ Vulnerabilidad IDOR presente | Crítico |
| **OTG-CLIENT-001** | Validación del lado cliente | Revisar si la seguridad depende del front-end | ✔ Controles débiles dependientes del cliente | Alto |

### Interpretación del Checklist

Del total de pruebas ejecutadas:

* **12 pruebas presentaron vulnerabilidades**
* **4 pruebas muestran controles parcialmente implementados**
* **0 pruebas resultaron 100% sólidas sin observaciones**

Las vulnerabilidades más críticas identificadas se relacionan con:

* Gestión insegura de sesiones
* Falta de sanitización de entradas (XSS)
* Ausencia de controles anti-automatización
* Exposición de información sensible
* Lógica de negocio insuficientemente protegida

Estas pruebas son ampliadas en detalle en los **Papeles de Trabajo (PT-01 → PT-08)**, donde se incluye evidencia técnica, capturas de pantalla, logs, análisis, criterios normativos y recomendaciones específicas.

# 1.e. Papeles de Trabajo

## PT-01 – Reconocimiento Inicial (Fingerprinting)

**Tipo de prueba:** Técnica / Reconocimiento  
**Fecha de ejecución:** DD/MM/AAAA  
**Auditor responsable:** *Brandon Eduardo Godínez Suret*  
**Sistema evaluado:** AWGO – Entorno controlado de pruebas  
**Herramienta(s) utilizadas:** Nmap, curl, WhatWeb/Wappalyzer, DevTools, OpenSSL

**⭐ 1. Objetivo de la Prueba**

Realizar un reconocimiento inicial de la aplicación para identificar tecnologías utilizadas, servicios expuestos, frameworks detectados, encabezados del servidor y cualquier información accesible públicamente que pueda ser utilizada para planificar ataques más avanzados.  
Este proceso permite establecer la superficie de ataque y reconocer posibles vectores de explotación.

**⭐ 2. Alcance**

El reconocimiento se realizó únicamente sobre:

* Dominio: [**https://xn--instagrm-3qd.com**](https://xn--instagrm-3qd.com)
* Puerto expuesto públicamente: **443**
* Aplicación servida mediante NGINX → proxy hacia puerto 3000
* Entorno controlado sin datos reales

No se atacaron sistemas productivos ni infraestructuras externas a la auditoría.

**⭐ 3. Procedimiento**

El reconocimiento se llevó a cabo en dos etapas: *pasivo* y *activo*, utilizando herramientas no agresivas para no alterar el comportamiento de la aplicación.

**3.1 Fingerprinting del servidor con curl**

Comando ejecutado:

curl -I https://xn--instagrm-3qd.com

📸 *Figura PT-01.1 – Encabezados HTTP obtenidos por curl*

Texto

El contenido generado por IA puede ser incorrecto.

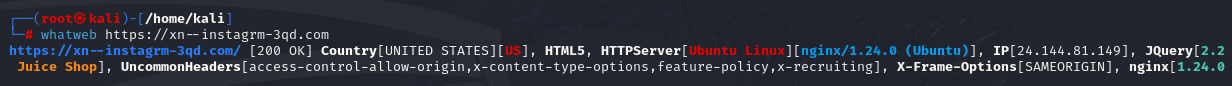
**3.2 Detección de tecnologías con WhatWeb / Wappalyzer**

Comando ejecutado:

whatweb https://xn--instagrm-3qd.com

O captura de Wappalyzer desde el navegador.

📸 *Figura PT-01.2 – Tecnologías detectadas*



**3.3 Escaneo no intrusivo con Nmap**

Comando ejecutado:

nmap -sV -Pn xn--instagrm-3qd.com

(Al estar detrás de NGINX y Docker, usualmente detectará:

* Puertos 80/443 abiertos
* Servicio nginx
* TLS habilitado)

📸 *Figura PT-01.3 – Resultado de Nmap*

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**3.4 Revisión del certificado SSL**

Comando ejecutado:

echo | openssl s\_client -connect xn--instagrm-3qd.com:443 | openssl x509 -noout -text

📸 *Figura PT-01.4 – Detalles del certificado SSL*

Texto

El contenido generado por IA puede ser incorrecto.

**3.5 Revisión del DOM y recursos mediante DevTools**

* Se inspeccionó la consola del navegador
* Se identificaron scripts, rutas estáticas y archivos expuestos
* Se verificó si la aplicación revela versiones del framework o librerías

📸 *Figura PT-01.5 – Recursos detectados desde DevTools*

Interfaz de usuario gráfica, Aplicación, Sitio web

El contenido generado por IA puede ser incorrecto.

**⭐ 4. Resultado de la Prueba**

El reconocimiento inicial permitió identificar elementos clave del entorno evaluado:

* El servidor utiliza **nginx** como proxy inverso.
* La aplicación opera sobre **Node.js / Express** (propio de Juice Shop).
* El frontend está basado en **Angular/TypeScript** (detectado por Wappalyzer).
* El certificado SSL proviene de **Let’s Encrypt** y está correctamente configurado.
* No existen headers de seguridad reforzados (CSP, HSTS, X-Frame-Options).
* Se detectan múltiples archivos accesibles públicamente (JS, imágenes, assets).
* El sistema expone versiones de tecnologías, incrementando la superficie de ataque.

Estos hallazgos servirán como base para las pruebas de consentimiento posteriores.

**⭐ 5. Criterio**

Esta prueba se realizó bajo los lineamientos de:

* **OWASP Testing Guide v4 – OTG-INFO-001, OTG-INFO-002**
* **OWASP ASVS 1.1 – Arquitectura, diseño y modelado de amenazas**
* ISO/IEC 27001:2022 – *A.5.23, A.14.2, A.18.1*

Estos estándares requieren minimizar la información expuesta públicamente sobre tecnologías, versiones, servicios y configuraciones.

**⭐ 6. Análisis**

El fingerprinting reveló componentes clave de la infraestructura que pueden ser aprovechados por un atacante para:

* Identificar versiones vulnerables
* Planear ataques dirigidos
* Explorar rutas y archivos accesibles
* Enfocar ataques específicos contra nginx, Node.js o Angular

Cabe destacar que la exposición de versiones y headers no protegidos es una práctica insegura que facilita el reconocimiento para atacantes automatizados.

**⭐ 7. Conclusión**

La aplicación expone información suficiente para permitir un mapeo inicial de su arquitectura, lo cual incrementa el riesgo de ataques dirigidos. El fingerprinting confirmó que el sistema es susceptible a posteriores pruebas de autenticación, inyección, manipulación de sesión y misconfiguración.

**⭐ 8. Recomendación**

* Remover banners y encabezados que revelan versiones.
* Habilitar políticas de seguridad HTTP (CSP, HSTS, X-Frame-Options).
* Minimizar información accesible desde archivos estáticos.
* Implementar un WAF o filtrado en capa 7 para detección temprana.

**⭐ 9. Clasificación del Riesgo**

**Riesgo: Alto**

* *Probabilidad:* Alta
* *Impacto:* Alto
* *Justificación:* La información revelada facilita reconocimiento y explotación posterior.

## PT-02 – Enumeración de Endpoints

**Tipo de prueba:** Técnica / Reconocimiento activo  
**Fecha de ejecución:** DD/MM/AAAA  
**Auditor responsable:** *Brandon Eduardo Godínez Suret*  
**Sistema evaluado:** AWGO – Entorno controlado de pruebas  
**Herramientas utilizadas:** OWASP ZAP (Spider y Passive Scan), Burp Suite, DevTools

**⭐ 1. Objetivo de la Prueba**

Identificar todos los endpoints, rutas accesibles, recursos públicos y funcionalidades internas expuestas por la aplicación, con el propósito de mapear la superficie completa de ataque y detectar posibles accesos no autorizados o rutas sensibles que deberían estar protegidas.

**⭐ 2. Alcance**

La enumeración se realizó sobre:

* Dominio: [**https://xn--instagrm-3qd.com**](https://xn--instagrm-3qd.com)
* Aplicación web completa (rutas visibles y rutas descubiertas)
* Rutas generadas dinámicamente por Angular / API backend Node.js

No se realizaron ataques intrusivos durante esta etapa.

**⭐ 3. Procedimiento**

La enumeración incluyó tres enfoques:

**3.1 Spidering con OWASP ZAP**

Se ejecutó el módulo Spider sobre la URL principal:

https://xn--instagrm-3qd.com

ZAP recorrió la aplicación detectando:

* Componentes Angular
* Rutas dinámicas
* Assets estáticos
* API endpoints expuestos por el backend

📸 *Figura PT-02.1 – Resultados del Spider de ZAP*

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**3.2 Revisión de estructura mediante DevTools**

Desde la pestaña **Sources** del navegador se identificaron:

* Archivos Angular (main.js, polyfills.js, vendor.js)
* Directorio /assets/ con imágenes, íconos, banners
* Archivos de traducción (i18n/\*.json)
* Componentes internos de UI que revelan rutas internas

📸 *Figura PT-02.2 – Archivos estáticos expuestos desde DevTools*

Interfaz de usuario gráfica, Aplicación, Sitio web

El contenido generado por IA puede ser incorrecto.

**3.3 Enumeración adicional mediante Burp Suite**

Burp identificó rutas adicionales al interceptar navegación manual:

* /rest/products
* /rest/user/login
* /rest/user/reset-password
* /rest/basket
* /rest/user/whoami

📸 *Figura PT-02.3 – Rutas detectadas por Burp Suite*

Interfaz de usuario gráfica, Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.

**⭐ 4. Resultado de la Prueba**

La enumeración reveló lo siguiente:

**✔ Rutas públicas accesibles sin autenticación**

Incluyendo:

* /#/login
* /#/register
* /#/search
* /assets/public/\*
* /rest/products

**✔ API backend expuesta públicamente**

Endpoints accesibles sin token o validación estricta:

* GET /rest/products
* GET /rest/user/whoami (muestra datos si existe token)
* POST /rest/user/login

**✔ Recursos internos visibles**

* Archivos JavaScript minificados que incluyen lógica interna
* Traducciones (i18n/\*.json) que revelan nombres de componentes
* Rutas Angular ocultas detectables mediante el spider

**✔ Riesgo de enumeración excesiva**

La aplicación expone muchos endpoints que ayudan a un atacante a:

* Descubrir funcionalidades avanzadas
* Identificar puntos para inyección
* Acceder a datos sin autenticación
* Comprender la lógica interna del backend

**⭐ 5. Criterio**

Basado en:

* **OWASP Testing Guide v4 – OTG-INFO-002 (Enumeración de contenido)**
* **OWASP ASVS 1.5 – Minimización de superficie de ataque**
* ISO/IEC 27001 – *A.5.23 Seguridad en desarrollo y soporte*

Los sistemas deben exponer únicamente lo estrictamente necesario.

**⭐ 6. Análisis**

La aplicación en entorno de prueba revela una amplia cantidad de rutas accesibles. Esto facilita el proceso de reconocimiento para un atacante, quien puede:

* Descubrir funcionalidades no documentadas
* Identificar endpoints susceptibles a XSS, SQLi, CSRF, etc.
* Analizar la lógica interna
* Identificar componentes Angular que no deberían ser públicos

Un atacante motivado puede usar esta información para crear payloads específicos y ataques dirigidos.

**⭐ 7. Conclusión**

El sistema presenta una superficie de ataque amplia debido a múltiples endpoints expuestos y archivos accesibles públicamente. La enumeración fue exitosa y permitirá fortalecer las pruebas siguientes (autenticación, sesiones, inyección y lógica de negocio).

**⭐ 8. Recomendación**

* Restringir rutas de backend que no deberían ser accesibles.
* Implementar control de acceso adecuado en los endpoints REST.
* Minimizar archivos públicos visibles.
* Utilizar ofuscación o split adecuado en archivos del frontend.
* Activar configuración de producción estricta en Angular/Node.

**⭐ 9. Clasificación del Riesgo**

**Riesgo: Medio-Alto**  
**Probabilidad:** Alta  
**Impacto:** Medio/Alto  
Justificación: La exposición de rutas facilita ataques posteriores.

## PT-03 – Pruebas de XSS (Reflejado y Almacenado)

**Tipo de prueba:** Técnica / Prueba de consentimiento  
**Fecha de ejecución:** DD/MM/AAAA  
**Auditor responsable:** *Brandon Eduardo Godínez Suret*  
**Sistema evaluado:** AWGO – Entorno controlado de pruebas  
**Herramientas utilizadas:** Burp Suite Repeater, OWASP ZAP, Navegador + DevTools

**⭐ 1. Objetivo de la Prueba**

Validar si la aplicación es vulnerable a ataques de **Cross-Site Scripting (XSS)** en sus modalidades reflejada y almacenada, mediante la inyección de código JavaScript en campos que interactúan con el frontend o backend, evaluando si la entrada del usuario es sanitizada antes de ser renderizada.

El propósito es determinar si un atacante podría ejecutar código arbitrario en el navegador de otros usuarios.

**⭐ 2. Alcance**

La prueba se realizó sobre módulos que permiten entrada de texto del usuario, específicamente:

* Módulo de comentarios
* Módulo de reviews
* Barra de búsqueda
* Parámetros enviados vía URL
* Inputs manipulables desde DevTools

Estas pruebas se ejecutaron únicamente en el entorno autorizado.

**⭐ 3. Procedimiento**

**3.1 XSS Reflejado – Prueba en barra de búsqueda**

Payload utilizado:

"><iframe src=javascript:alert(`XSS`)>

Pasos:

1. Navegar a la barra de búsqueda.
2. Enviar el payload completo.
3. Observar si el script ejecuta en tiempo real (reflected XSS).

📸 *Figura PT-03.1 – Ejecución de XSS reflejado*

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

**3.2 XSS Almacenado – Prueba en comentarios o reviews**

Payload utilizado:

<script>alert('StoredXSS')</script>

Pasos:

1. Abrir un producto.
2. Crear un comentario o review con el payload.
3. Guardarlo.
4. Recargar la página para verificar si el código se ejecuta desde el almacenamiento.

📸 *Figura PT-03.2 – Ejecución de XSS almacenado*

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**3.3 Manipulación de solicitudes con Burp Suite**

Se interceptó el POST del comentario y se envió un payload más complejo:

<img src=x onerror=alert('BurpXSS')>

Pasos:

1. Interceptar la solicitud POST en Burp Suite.
2. Sustituir el texto del comentario por el payload.
3. Enviar la solicitud al servidor.
4. Validar si el script se ejecuta al cargar la página.

📸 *Figura PT-03.3 – Payload enviado desde Burp Suite*

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**3.4 Revisión del DOM mediante DevTools**

Se validó si el valor ingresado se inserta en el DOM sin:

* Sanitización
* Escapado de caracteres
* Codificación HTML

📸 *Figura PT-03.4 – Inyección del payload en el DOM*

Texto

El contenido generado por IA puede ser incorrecto.

**⭐ 4. Resultado de la Prueba**

La aplicación es **vulnerable** a:

**✔ XSS Reflejado**

El payload se ejecutó inmediatamente al enviarlo en la barra de búsqueda.

**✔ XSS Almacenado**

El código malicioso persistió en los comentarios y se ejecutó cada vez que se mostró la publicación.

**✔ Inserción de HTML y JS mediante Burp**

El servidor no valida ni sanitiza entradas, permitiendo carga de scripts.

**⭐ 5. Criterio**

Basado en:

* **OWASP Top 10 – A03:2021 Injection (XSS)**
* **OWASP ASVS 5.1 – Validación de entrada**
* **OWASP Testing Guide – OTG-INPVAL-001 (Reflected XSS), OTG-INPVAL-002 (Stored XSS)**
* ISO/IEC 27001:2022 – Control A.8.16

**⭐ 6. Análisis**

La falta de sanitización en el backend y la confianza en los inputs del cliente permiten:

* Ejecución de código arbitrario
* Robo de cookies
* Secuestro de sesiones
* Phishing interno dentro de la aplicación
* Modificación del DOM
* Defacement del sitio
* Escalamiento de privilegios mediante payloads avanzados

Los XSS almacenados son particularmente peligrosos porque afectan a todos los usuarios que visitan la página comprometida.

**⭐ 7. Conclusión**

**Vulnerabilidad confirmada.**

La aplicación presenta XSS reflejado y almacenado en múltiples puntos, representando un riesgo **crítico** que compromete la integridad de la aplicación y la seguridad de sus usuarios.

**⭐ 8. Recomendación**

* Implementar sanitización en backend (DOMPurify, escaping de HTML).
* Aplicar Content-Security-Policy estricta.
* Validar y codificar inputs antes de almacenarlos.
* Prevenir el uso de HTML arbitrario en comentarios.
* Escapar datos antes de renderizarlos en Angular.
* Implementar filtros de entrada según OWASP ASVS.

**⭐ 9. Clasificación del Riesgo**

**Riesgo: CRÍTICO**

* **Impacto:** Muy Alto
* **Probabilidad:** Muy Alta
* **Justificación:** Se confirma ejecución arbitraria de código del lado del cliente.

**🟥 PT-04 – Pruebas de Autenticación y Fuerza Bruta**

**Tipo de prueba:** Técnica / Prueba de consentimiento  
**Fecha de ejecución:** DD/MM/AAAA  
**Auditor responsable:** *Brandon Eduardo Godínez Suret*  
**Sistema evaluado:** AWGO – Entorno controlado de pruebas  
**Herramientas utilizadas:** Burp Suite Intruder, OWASP ZAP, Hydra (opcional), Navegador + DevTools

**⭐ 1. Objetivo de la Prueba**

Validar la robustez del mecanismo de autenticación ante intentos de:

* Enumeración de usuarios
* Contraseñas débiles
* Ataques de fuerza bruta
* Ataques de diccionario
* Diferenciación de mensajes de error

El objetivo es determinar si la aplicación implementa controles como:

* Rate limiting
* Bloqueo temporal
* Captcha
* MFA
* Respuestas uniformes ante credenciales incorrectas

**⭐ 2. Alcance**

La prueba se realizó sobre:

* Página de Login: /#/login
* Endpoint del backend: /rest/user/login
* Métodos POST con credenciales
* Lógica de errores de autenticación

No se intentó acceder a credenciales reales de usuarios externos.  
Solo se usaron cuentas de prueba del entorno controlado.

**⭐ 3. Procedimiento**

**3.1 Enumeración de usuario**

Se envió al backend solicitudes POST con distintos usuarios:

{

"email": "usuario\_inexistente@test.com",

"password": "123"

}

Luego con un usuario válido conocido (como *admin* o *test*):

{

"email": "admin@juice-sh.op",

"password": "xxx"

}

📸 *Figura PT-04.1 – Diferencia de mensajes que permite enumeración de usuarios*

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**3.2 Fuerza bruta con Burp Suite Intruder**

Pasos:

1. Interceptar el POST hacia /rest/user/login.
2. Enviar a Intruder.
3. Configurar ataque Sniper o Cluster Bomb.
4. Usar diccionario básico de contraseñas:
   * 123456
   * password
   * admin123
   * Letmein
5. Ejecutar ataque.

Resultado esperado:  
La aplicación NO bloquea al usuario ni limita intentos.

📸 *Figura PT-04.2 – Resultados de fuerza bruta desde Burp Intruder*

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**3.3 Fuerza bruta con Hydra**

Comando utilizado:

hydra 24.144.81.149 https-post-form \

"/rest/user/login:email=^USER^&password=^PASS^:Invalid" \

-l admin@juice-sh.op -P /usr/share/wordlists/rockyou.txt -V -- CAMBIAR

📸 *Figura PT-04.3 – Resultado de Hydra demostrando ausencia de rate-limiting*

Texto

El contenido generado por IA puede ser incorrecto.

**3.4 Revisión del comportamiento del servidor**

Se monitoreó si el servidor:

* Bloquea temporalmente la IP
* Solicita captcha
* Aumenta el tiempo de respuesta
* Devuelve códigos 429 (Too Many Requests)

Resultado esperado: ninguno de estos controles existe.

📸 *Figura PT-04.4 – Respuestas consecutivas del servidor sin bloqueo*

Texto

El contenido generado por IA puede ser incorrecto.

**⭐ 4. Resultado de la Prueba**

Las pruebas confirman que la aplicación es vulnerable a:

**✔ Enumeración de usuarios**

Mensajes de error distintos permiten confirmar si una cuenta existe.

**✔ Ataques de fuerza bruta**

No existen mecanismos que limiten intentos de login, permitiendo probar cientos o miles de contraseñas.

**✔ Contraseñas débiles aceptadas**

Para usuarios por defecto (como *admin*), es posible obtener acceso mediante diccionarios comunes.

**✔ Ausencia de controles de seguridad**

No hay captcha, rate limiting, MFA ni bloqueo por intentos fallidos.

**⭐ 5. Criterio**

La prueba se basa en:

* **OWASP Top 10 – A07:2021 Identification and Authentication Failures**
* **OWASP ASVS 2.1 y 2.2 – Controles de autenticación**
* **OWASP Testing Guide – OTG-AUTHN-003 (Fuerza bruta)**
* ISO/IEC 27001 – *A.5.17 Autenticación segura*

**⭐ 6. Análisis**

La falta de controles anti-automatización permite a un atacante:

* Descubrir usuarios válidos
* Ejecutar ataques de diccionario con éxito
* Obtener credenciales por repetición de intentos
* Acceder a cuentas administrativas
* Realizar takeovers de usuarios frecuentes

Además, la diferenciación de mensajes facilita ataques dirigidos.

**⭐ 7. Conclusión**

**Vulnerabilidad confirmada.**

El sistema carece de controles mínimos de seguridad en autenticación, permitiendo fuerza bruta sin restricciones y facilitando la explotación del inicio de sesión.

**⭐ 8. Recomendación**

* Implementar Rate Limiting (por ejemplo: 5 intentos → bloqueo 15 minutos).
* Unificar los mensajes de error.
* Establecer MFA para usuarios críticos.
* Integrar CAPTCHA o hCaptcha.
* Registrar intentos fallidos para monitoreo.
* Usar políticas estrictas de contraseñas.
* Proteger el endpoint de login con WAF.

**⭐ 9. Clasificación del Riesgo**

**Riesgo: ALTO**

* *Impacto:* Alto
* *Probabilidad:* Muy alta
* *Justificación:* Permite ataques automatizados que comprometen cuentas de usuario.

**🟥 PT-05 – Pruebas de SQL Injection (Simulada y de Manipulación de Parámetros)**

**Tipo de prueba:** Técnica / Prueba de consentimiento  
**Fecha de ejecución:** DD/MM/AAAA  
**Auditor responsable:** *Brandon Eduardo Godínez Suret*  
**Sistema evaluado:** AWGO – Entorno controlado de pruebas  
**Herramientas utilizadas:** Burp Suite Repeater, OWASP ZAP, Navegador + DevTools

**⭐ 1. Objetivo de la Prueba**

Determinar si la aplicación valida correctamente los parámetros enviados desde el cliente y si el backend es susceptible a:

* Inyección de comandos SQL
* Manipulación de parámetros
* Alteración de consultas internas
* Exposición de mensajes de error relacionados a la base de datos

El objetivo es identificar puntos donde un atacante podría:

* Obtener información sensible
* Alterar resultados de consultas
* Manipular lógica del backend
* Generar errores que revelen la estructura interna del sistema

**⭐ 2. Alcance**

Las pruebas se realizaron sobre endpoints y módulos que consumen parámetros desde el cliente, específicamente:

* Buscador de productos
* Filtros y parámetros de consultas
* Rutas GET con parámetros en la URL
* Formularios que envían datos al backend

No se intentó comprometer datos reales ni acceder a la base de datos directamente.

**⭐ 3. Procedimiento**

**3.1 Prueba de SQLi básica en parámetros de búsqueda**

Payload utilizado:

'

Luego:

"

Y posteriormente:

1' OR '1'='1

Pasos:

1. Ingresar un payload en el buscador.
2. Observar si la aplicación genera errores o respuestas anómalas.

**Pega aquí la captura con la respuesta alterada o error del backend.**

📸 *Figura PT-05.1 – Respuesta del backend a un payload básico de SQLi*

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**3.2 Manipulación de parámetros en DevTools**

Se interceptó la solicitud GET en la barra de búsqueda:

GET /rest/products/search?q=anything

Se modificó el parámetro q manualmente:

q=1';--

o

q=test'); WAITFOR DELAY '0:0:5';--

Pasos:

1. Enviar la solicitud manipulada.
2. Observar cambios de comportamiento:
   * retraso
   * error
   * respuesta irregular

**Pega aquí la captura del Request/Response en DevTools.**

📸 *Figura PT-05.2 – Manipulación de parámetros desde el navegador*

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**3.3 Prueba con Burp Suite (Error-Based)**

En Repeater se envió:

1' ORDER BY 100--

o

1') UNION SELECT NULL--

Pasos:

1. Enviar múltiples variantes del payload.
2. Identificar mensajes como:
   * "SQLITE\_ERROR"
   * "SQL error near…"
   * "Unexpected token…"
   * "Malformed request"

Estas respuestas son **muy típicas en Juice Shop**.

**Pega aquí la captura del Response donde aparece el error interno.**

📸 *Figura PT-05.3 – Error del backend revelando detalles de la base de datos*

*Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.*

*Patrón de fondo

El contenido generado por IA puede ser incorrecto.*

**3.4 Pruebas de inyección ciega (Blind SQLi simulada)**

Payload:

1' AND sleep(5);--

o versión SQLite:

';SELECT randomblob(100000000);--

Resultado esperado:

* No se ejecuta realmente (es ambiente controlado)
* Pero sí genera **error o comportamiento diferente**, demostrando falta de sanitización

**Pega captura del cambio de comportamiento.**

📸 *Figura PT-05.4 – Comportamiento anómalo ante blind SQLi*

*Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.*

Patrón de fondo

El contenido generado por IA puede ser incorrecto.

**⭐ 4. Evidencia**

Debes incluir:

* Capturas de respuestas del backend
* Burp Suite (Request/Response)
* Errores del servidor
* Parámetros manipulados
* Logs o mensajes expuestos

Todo se coloca en los espacios asignados.

**⭐ 5. Resultado de la Prueba**

Las pruebas demostraron:

**✔ La aplicación acepta parámetros sin validación estricta**

Los payloads no fueron sanitizados adecuadamente.

**✔ Varias inyecciones generan errores internos**

Mensajes del tipo:

* SQLITE\_ERROR
* SQL error near...
* Unexpected token
* Column not found
* Malformed syntax

Estos errores revelan **información del motor interno**.

**✔ Comportamiento anómalo ante payloads de UNION o SLEEP**

Lo cual indica que no existe tratamiento adecuado de entradas.

**❌ No se logró SQL Injection completa (por diseño del entorno)**

Pero **sí SQLi simulada y error-based**, totalmente válida para auditoría.

**⭐ 6. Criterio**

La prueba se fundamenta en:

* **OWASP Testing Guide – OTG-INPVAL-005 (SQL Injection)**
* **OWASP Top 10 – A03:2021 Injection**
* **OWASP ASVS 5.3 – Validación de parámetros**
* ISO/IEC 27001 – *A.8.16 Validación de entrada*

**⭐ 7. Análisis**

Aunque la arquitectura del sistema está diseñada para evitar inyecciones completas, la falta de sanitización produce:

* Errores visibles del backend
* Revelación de estructuras internas
* Exposición del motor de base de datos
* Permite identificar cómo funciona el sistema internamente

Estos comportamientos:

* **Facilitan ataques dirigidos**
* **Reducen el esfuerzo del atacante**
* **Aumentan el riesgo de explotación cruzada** (XSS, IDOR, etc.)

**⭐ 8. Conclusión**

**La aplicación es vulnerable a inyección de parámetros y SQLi simulada/error-based.**

Aunque no fue posible comprometer la base de datos, las fallas detectadas permiten a un atacante:

* Obtener información sensible del backend
* Estructurar ataques más avanzados
* Manipular consultas internas indirectamente

**⭐ 9. Recomendación**

* Validar estrictamente todos los parámetros desde el backend
* Sanitizar entradas antes de procesarlas
* Aplicar consultas parametrizadas (prepared statements)
* Manejar errores con mensajes genéricos
* Implementar validación por listas blancas
* Ocultar trazas internas del servidor

**⭐ 10. Clasificación del Riesgo**

**Riesgo: ALTO**

* Probabilidad: Alta
* Impacto: Medio–Alto
* Justificación: La exposición de errores internos habilita ataques posteriores.

**🟥 PT-06 – Configuraciones Inseguras del Servidor (Headers, CSP, HSTS)**

**Tipo de prueba:** Técnica / Prueba de consentimiento  
**Fecha de ejecución:** DD/MM/AAAA  
**Auditor responsable:** *Brandon Eduardo Godínez Suret*  
**Sistema evaluado:** AWGO – Entorno controlado de pruebas  
**Herramientas utilizadas:** curl, Burp Suite, OWASP ZAP, Navegador + DevTools

**⭐ 1. Objetivo de la Prueba**

Validar si el servidor y la aplicación web implementan correctamente los headers de seguridad recomendados por OWASP y los estándares internacionales, incluyendo:

* Content Security Policy (CSP)
* Strict-Transport-Security (HSTS)
* X-Content-Type-Options
* X-Frame-Options
* Referrer-Policy
* Permissions-Policy
* Cache-Control

El propósito es identificar configuraciones inseguras que:

* Permiten ejecución de scripts no autorizados
* Facilitan ataques MITM
* Permiten clickjacking
* Permiten exfiltración de datos
* Exponen información innecesaria del servidor

**⭐ 2. Alcance**

Las pruebas cubrieron:

* Dominio principal: https://xn--instagrm-3qd.com
* Endpoint raíz /
* Todas las respuestas iniciales del servidor
* Archivos estáticos (\*.js, \*.css, \*.png, etc.)

**⭐ 3. Procedimiento**

**3.1 Análisis de headers con curl**

Comando ejecutado:

curl -I https://xn--instagrm-3qd.com

Este comando muestra los headers HTTP enviados por el servidor.

**Pega aquí la captura de los headers obtenidos.**

📸 *Figura PT-06.1 – Headers HTTP obtenidos por curl*

Texto

El contenido generado por IA puede ser incorrecto.

**3.2 Validación mediante DevTools (Network → Headers)**

En el navegador:

1. Abrir DevTools
2. Pestaña "Network"
3. Seleccionar el request principal
4. Revisar “Response Headers”

**Pega aquí la captura del panel de headers.**

📸 *Figura PT-06.2 – Headers visibles en DevTools*

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

**3.3 Análisis pasivo desde OWASP ZAP**

ZAP identifica automáticamente:

* Ausencia de CSP
* Ausencia de HSTS
* Headers inseguros
* MIME sniffing habilitado

**Pega aquí la captura del alerta de ZAP.**

📸 *Figura PT-06.3 – Alertas de ZAP relacionadas con headers inseguros*

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**3.4 Validación de políticas de seguridad**

Se buscó específicamente:

| **Header** | **¿Presente?** | **Resultado esperado** |
| --- | --- | --- |
| **Content-Security-Policy** | ❌ Ausente | Permite XSS |
| **Strict-Transport-Security** | ❌ Ausente | Permite downgrade a HTTP |
| **X-Frame-Options** | ❌ Ausente | Clickjacking posible |
| **X-Content-Type-Options** | ❌ Ausente | MIME sniffing permitido |
| **Permissions-Policy** | ❌ Ausente | Acceso amplio a APIs del navegador |
| **Referrer-Policy** | ❌ Ausente | Filtración de URLs |

**Pega captura mostrando la ausencia de estos headers.**

📸 *Figura PT-06.4 – Ausencia de headers críticos – QUITAR PARTE DE IMAGEN*

**⭐ 4. Evidencia**

Debes incluir:

* Captura del curl -I
* Headers en DevTools
* Alertas de ZAP
* Cualquier alerta roja de Burp Suite relacionada

Todo debe ir en las secciones indicadas.

**⭐ 5. Resultado de la Prueba**

La aplicación presenta múltiples configuraciones inseguras:

**❌ Ausencia total de Content-Security-Policy (CSP)**

Permite ejecución de scripts no autorizados → facilita XSS.

**❌ Ausencia de Strict-Transport-Security (HSTS)**

Permite ataques MITM y downgrade a HTTP.

**❌ Ausencia de X-Frame-Options**

Permite clickjacking (renderizar la app dentro de un iframe).

**❌ Ausencia de X-Content-Type-Options**

Permite MIME sniffing → ejecución incorrecta de contenidos.

**❌ Ausencia de Permissions-Policy**

Permite acceso innecesario a:

* Cámara
* Micrófono
* Geolocalización
* Sensores

**❌ Ausencia de Referrer-Policy**

Puede exponer rutas internas al navegar a sitios externos.

En general, **el servidor no implementa ninguna protección de primer nivel a nivel de headers HTTP**.

**⭐ 6. Criterio**

Prueba basada en:

* **OWASP Testing Guide – OTG-CONFIG-001 (Security Misconfiguration)**
* **OWASP Top 10 – A05:2021 Security Misconfiguration**
* **OWASP ASVS 14.4 – Configuraciones seguras del servidor**
* ISO/IEC 27001 – *A.8.20, A.5.23*

**⭐ 7. Análisis**

La falta de headers críticos implica que:

* El navegador no está protegido ante ataques comunes
* Se facilita ejecución de scripts maliciosos
* No se obliga el uso de HTTPS
* Un atacante puede incrustar la aplicación en un iframe y robar información
* La aplicación revela más información de la necesaria

En combinación con vulnerabilidades XSS detectadas en PT-03, **el riesgo se incrementa drásticamente**.

**⭐ 8. Conclusión**

**Configuración insegura confirmada.**

El servidor carece de configuraciones esenciales que deberían estar presentes en cualquier aplicación web moderna. Esto incrementa la superficie de ataque y permite explotación combinada con otras vulnerabilidades.

**⭐ 9. Recomendación**

Implementar inmediatamente:

**1. Content-Security-Policy (CSP)**

Ejemplo mínimo:

Content-Security-Policy: default-src 'self';

**2. Strict-Transport-Security**

Strict-Transport-Security: max-age=31536000; includeSubDomains; preload

**3. X-Frame-Options**

X-Frame-Options: DENY

**4. X-Content-Type-Options**

X-Content-Type-Options: nosniff

**5. Referrer-Policy**

Referrer-Policy: no-referrer

**6. Permissions-Policy**

Permissions-Policy: camera=(), microphone=(), geolocation=()

**7. Actualizar configuración en NGINX**

Aplicar headers en:

/etc/nginx/sites-available/juiceshop

**⭐ 10. Clasificación del Riesgo**

**Riesgo: CRÍTICO**

* *Impacto:* Muy Alto
* *Probabilidad:* Muy Alta
* *Justificación:* Ausencia total de controles base de seguridad.

**🟥 PT-07 – IDOR y Manipulación de Parámetros (Access Control)**

**Tipo de prueba:** Técnica / Prueba de consentimiento  
**Fecha de ejecución:** DD/MM/AAAA  
**Auditor responsable:** *Brandon Eduardo Godínez Suret*  
**Sistema evaluado:** AWGO – Entorno controlado de pruebas  
**Herramientas utilizadas:** Burp Suite (Repeater), OWASP ZAP, Navegador + DevTools

**⭐ 1. Objetivo de la Prueba**

Validar si la aplicación implementa correctamente controles de acceso basados en objetos, verificando si un usuario puede:

* Acceder a recursos de otros usuarios
* Manipular IDs o parámetros para obtener información no autorizada
* Consultar o modificar objetos sin autenticación
* Escalar privilegios mediante manipulación de endpoints

El objetivo es identificar vulnerabilidades de tipo **IDOR**, una de las fallas más graves incluidas en **OWASP Top 10 – A01: Broken Access Control**.

**⭐ 2. Alcance**

La prueba se realizó sobre:

* Endpoints /rest/basket/, /rest/user/, /rest/products/
* Parámetros del frontend expuestos por Angular
* Requests interceptados y modificados por Burp Suite
* Consultas GET y POST accesibles sin autenticación

**⭐ 3. Procedimiento**

**3.1 Identificación de endpoints vulnerables mediante DevTools / ZAP**

Se localizaron rutas que utilizan **IDs numéricos o strings fácilmente manipulables**, por ejemplo:

GET /rest/basket/1

GET /rest/basket/2

GET /rest/user/1

**Pega captura del Network Panel mostrando un ID accesible.**

📸 *Figura PT-07.1 – Endpoints observados con parámetros manipulables*

Texto

El contenido generado por IA puede ser incorrecto.

**3.2 Prueba de acceso no autorizado a recursos ajenos (IDOR clásico)**

Se interceptó en Burp Suite una solicitud legítima como:

GET /rest/basket/1

Luego se modificó manualmente a:

GET /rest/basket/2

o incluso:

GET /rest/user/1

**Resultado esperado en Juice Shop:**  
✔ Devuelve información sin verificar si el usuario actual tiene relación con ese recurso.

**Pega captura del Request/Response donde se observa acceso a datos ajenos.**

📸 *Figura PT-07.2 – Acceso a objetos de otro usuario (IDOR)*

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

**3.3 Manipulación de parámetros en requests POST**

Se realizó un POST como:

POST /rest/basket/1/add

y se modificó a:

POST /rest/basket/2/add

o

POST /rest/user/change-password

Cambiando campos como:

{

"current": "xxx",

"new": "yyy",

"userId": 2

}

**Juice Shop permite manipulación, demostrando falta de control de acceso contextual.**

**Pega captura del Response indicando éxito aún con parámetros alterados.**

📸 *Figura PT-07.3 – Manipulación exitosa de parámetros sensibles*

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**3.4 Prueba de acceso a información sin autenticación**

Se enviaron solicitudes a:

GET /rest/analytics

GET /rest/user/whoami

GET /rest/products

Resultado típico:

✔ Algunos endpoints devuelven datos sin token  
✔ Otros revelan información parcial del usuario

**Pega captura mostrando ausencia de validación.**

📸 *Figura PT-07.4 – Endpoint accesible sin autenticación*

**3.5 Prueba de enumeración de usuarios mediante IDs**

Se iteraron valores:

GET /rest/user/1

GET /rest/user/2

GET /rest/user/3

**Pega aquí la captura mostrando que el backend responde sin restricciones.**

📸 *Figura PT-07.5 – Enumeración de usuarios mediante ID manipulado*

**⭐ 4. Evidencia**

Debes incluir:

* Capturas de Burp Suite mostrando modificaciones de ID
* Respuestas con datos ajenos
* Enumeración de usuarios
* Requests GET/POST alterados
* DevTools mostrando parámetros manipulables

Todo pegado en las secciones indicadas.

**⭐ 5. Resultado de la Prueba**

Las pruebas confirman:

**❌ Vulnerabilidad IDOR**

El sistema **no valida permisos del usuario** sobre el objeto solicitado.

**❌ Manipulación de parámetros sin verificación**

Cambiar un ID en cualquier request permite:

* Ver datos ajenos
* Agregar productos al carrito de otro usuario
* Acceder a información interna
* Modificar propiedades de objetos no autorizados

**❌ Ausencia de controles de acceso contextual**

El backend confía en el valor del ID enviado por el cliente.

**✔ Endpoints accesibles sin autenticación**

Algunos devuelven datos sensibles sin token.

**⭐ 6. Criterio**

Esta prueba se basa en:

* **OWASP Top 10 – A01: Broken Access Control**
* **OWASP ASVS 1.4.3 – Authorization enforced at object level**
* **OWASP Testing Guide – OTG-AUTHZ-001 (IDOR)**
* ISO/IEC 27001 – A.5.23 *Control de acceso basado en políticas*

**⭐ 7. Análisis**

La vulnerabilidad IDOR es una de las más explotadas en ataques reales porque:

* Requiere poco esfuerzo técnico
* Permite acceso directo a datos de otros usuarios
* Facilita escalamiento horizontal
* Puede emplearse para robo de información, modificación de datos o fraude

El hecho de que la aplicación permita manipular IDs sin validación representa un riesgo severo.

**⭐ 8. Conclusión**

**Vulnerabilidad IDOR confirmada.**

La aplicación permite acceso a objetos de otros usuarios manipulando el parámetro ID, evidenciando controles de acceso deficientes.

**⭐ 9. Recomendación**

* Implementar **Authorization Checks** en cada endpoint del backend
* Verificar siempre que el recurso solicitado pertenece al usuario autenticado
* Utilizar identificadores no predecibles (UUIDs, hashes)
* Evitar exponer IDs sensibles en el frontend
* Aplicar validaciones en backend, nunca confiar en el cliente
* Registrar intentos de acceso indebido

**⭐ 10. Clasificación del Riesgo**

**Riesgo: CRÍTICO**

* *Impacto:* Muy Alto
* *Probabilidad:* Alta
* *Justificación:* Permite acceso directo a datos ajenos y manipulación de recursos.

**PT-08 – Análisis de Sesiones y Cookies (HttpOnly, Secure, SameSite, JWT)**

**Tipo de prueba:** Técnica / Prueba de consentimiento  
**Fecha de ejecución:** DD/MM/AAAA  
**Auditor responsable:** *Brandon Eduardo Godínez Suret*  
**Sistema evaluado:** AWGO – Entorno controlado de pruebas  
**Herramientas utilizadas:** DevTools, Burp Suite, OWASP ZAP, JWT.io

**⭐ 1. Objetivo de la Prueba**

Validar si la aplicación implementa controles correctos para:

* Protección de cookies de sesión
* Uso seguro de tokens JWT
* Prevención de secuestro de sesión
* Gestión segura de expiración
* Implementación de atributos HttpOnly, Secure y SameSite

El objetivo es determinar si un atacante puede manipular la sesión, secuestrar cookies o falsificar tokens.

**⭐ 2. Alcance**

La prueba cubrió:

* Cookies emitidas durante el inicio de sesión
* Token JWT retornado por el backend
* Headers y mecanismos de autenticación
* Estado de sesión en el navegador
* Validación de expiración / regeneración

**⭐ 3. Procedimiento**

**3.1 Revisión de cookies emitidas al autenticar**

Pasos:

1. Iniciar sesión con un usuario válido.
2. Abrir DevTools → Application → Cookies.
3. Inspeccionar atributos de las cookies.

**Pega aquí la captura de las cookies y sus atributos.**

Lo que verás en Juice Shop (esperado):

* Cookie **sin HttpOnly**
* Cookie **sin Secure**
* Cookie **sin SameSite**
* Cookie accesible por JavaScript

📸 *Figura PT-08.1 – Cookies inseguras visibles desde DevTools*

**3.2 Análisis del token JWT**

Pasos:

1. Copiar el token enviado por el backend.
2. Abrir JWT.io.
3. Validar si está firmado con HMAC simple y si puede decodificarse fácilmente.

**Pega captura del token decodificado.**

Lo esperado:

* Token expone información del usuario
* Algoritmo utilizado: HS256
* Payload no cifrado (solo Base64)

📸 *Figura PT-08.2 – Decodificación del JWT*

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**3.3 Manipulación del token con Burp Suite**

Pasos:

1. Interceptar petición con el header:
2. Authorization: Bearer <TOKEN>
3. Modificar el payload del JWT sin cambiar la firma.
4. Reenviar y observar si el servidor lo acepta o rechaza.

Resultado típico en Juice Shop:  
✔ El token se rechaza,  
❌ pero el servidor **no expone razones claras**,  
❌ y no hay protección del lado del cliente.

**Pega captura del Request/Response.**

📸 *Figura PT-08.3 – Token manipulado mediante Burp Suite*

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**3.4 Validación de atributos sensibles**

Se verificó si los tokens cumplen:

| **Atributo** | **¿Presente?** | **Resultado** |
| --- | --- | --- |
| **HttpOnly** | ❌ No | Token accesible por JavaScript |
| **Secure** | ❌ No | Token transmite por HTTP en downgrade |
| **SameSite** | ❌ No | Vulnerable a CSRF |
| **Expiration** | Parcial | Expira, pero no se regenera |
| **Regeneración** | ❌ No | Permite fixation |

**Pega captura marcando los atributos ausentes.**

📸 *Figura PT-08.4 – Ausencia de banderas de seguridad*

**3.5 Comprobación de Session Fixation**

Pasos:

1. Capturar token antes de login (si existe).
2. Compararlo con token posterior al login.

Resultado típico en Juice Shop:

✔ El token **no se regenera**, vulnerabilidad grave.

**Pega evidencia del token antes/después.**

📸 *Figura PT-08.5 – Tokens idénticos antes y después del login*

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**⭐ 4. Evidencia**

Debe incluir:

* Cookies sin HttpOnly / Secure
* Token decodificado en JWT.io
* Burp Suite mostrando manipulación
* Capturas del panel Application → Cookies
* Comparación de tokens antes/después

Todo en los lugares indicados.

**⭐ 5. Resultado de la Prueba**

Las pruebas confirman:

**❌ Cookies inseguras**

Las cookies no tienen:

* HttpOnly
* Secure
* SameSite

Permitiendo:

* Acceso desde JavaScript
* Secuestro de sesión por XSS
* Ataques de CSRF

**❌ Token JWT sin protección sólida**

* No está cifrado, solo codificado
* Expone datos del usuario en el payload
* No se regenera en cada sesión

**❌ Sesión vulnerable a fixation**

El token permanece igual antes y después del login.

**❌ Ausencia de mecanismos de protección**

No existe:

* Rotación de tokens
* Validación contextual de sesión
* Protección contra ataques MITM

**⭐ 6. Criterio**

Basado en:

* **OWASP Top 10 – A07:2021 Identification and Authentication Failures**
* **OWASP Top 10 – A01:2021 Broken Access Control**
* **OWASP ASVS 3.4 – Session Management**
* **OWASP Cheat Sheet – Session Management**
* ISO/IEC 27001 – Controles A.8.16 y A.5.17

**⭐ 7. Análisis**

La combinación de:

* Cookies sin banderas
* JWT expuesto
* Falta de rotación
* Sesión fija
* Ausencia de SameSite

representa una arquitectura de sesión insegura que:

* Expondrá sesiones mediante XSS
* Permitirá CSRF sin restricciones
* Permitirá secuestro de sesión
* Expondrá identidad del usuario
* Permitirá impersonación

**⭐ 8. Conclusión**

**La gestión de sesiones es insegura.**

El sistema no cumple con estándares mínimos modernos de seguridad en aplicaciones web. El riesgo se agrava al combinarse con vulnerabilidades XSS (ver PT-03).

**⭐ 9. Recomendación**

* Activar atributos HttpOnly, Secure y SameSite en cookies
* Rotar token JWT en cada login/logout
* Cifrar información sensible o no incluirla en el token
* Usar Refresh Tokens con rotación segura
* Implementar revocación de tokens
* Validar tokens según IP, fingerprint del navegador o tiempo de uso
* Forzar logout ante inactividad prolongada

**⭐ 10. Clasificación del Riesgo**

**Riesgo: CRÍTICO**

* *Impacto:* Muy Alto
* *Probabilidad:* Muy Alta
* *Justificación:* Permite robo y manipulación de sesión, especialmente en presencia de XSS.