28 de marzo del 2025, San Jerónimo, Ciudad de México.

Breaking Apps

“Tu conoces del negocio, yo de las vulnerabilidades”

¡Hola! Este es un documento en el que se recopilan algunas de las notas que complementan la exposición que acabas de ver, no es la panacea, pero esperamos que las notas y enlaces condensados aquí sirvan para comenzar tu viaje, si es que se volvió de tu interés, en la importancia de la seguridad en aplicaciones de E-Commerce.

Contenido

Palabras y Términos Clave

- Cybersecurity
- E-Commerce
- Fintech
- Price Manipulation Vulnerabilities
- Parameter Tampering
- API Abuse
- Bussiness Logic Flaws
- Burpsuite
- OWASP ZAP
- Lagging and Monitoring
- Cart IDOR
- Tryhackme
- EShop
- Coupon Reuse

Notas Técnicas

Introducción

Las plataformas de comercio electrónico manejan millones de transacciones diarias, representando aproximadamente **el 20.1% de las compras globales** en 2024 de acuerdo a *Capital One Shopping* y del casi **15% de las compras en México**. Sin embargo, no todos los sitios reciben el mismo nivel de escrutinio en seguridad, lo que aumenta la probabilidad de encontrar vulnerabilidades de **manipulación de precios** en programas de *bug bounty*.

A pesar de esto, muchos investigadores evitan probar estos sistemas debido a su complejidad: flujos de pago intrincados, lógicas de descuentos enrevesadas y validaciones asimétricas (frontend vs. backend). Como resultado, estas implementaciones suelen quedar sin un análisis exhaustivo.

¿Qué es Fintech?

El mundo de las **FinTech**, esa intersección entre tecnología y servicios financieros, ha revolucionado la forma en que manejamos el dinero. Imagina aplicaciones que te permiten pagar con un clic, invertir desde tu teléfono o incluso acceder a créditos en minutos. Pero detrás de esta comodidad hay un desafío enorme: la seguridad.

A grandes rasgos, estamos hablando entonces del conjunto de empresas que usan tecnología para ofrecer servicios financieros innovadores, como:

- **Bancos digitales** (ej. Nubank, Revolut).
- **Apps de pagos** (ej. PayPal, Mercado Pago).
- **Plataformas de inversión** (ej. Robinhood).
- **Criptomonedas** (ej. Bitcoin, Coinbase).

Las aplicaciones FinTech manejan datos sensibles (desde números de tarjeta hasta historiales financieros), lo que las convierte en blancos jugosos para cibercriminales. Un error en su diseño puede llevar a fraudes masivos, robos de identidad o pérdidas millonarias. Por eso, la seguridad no es un añadido, sino el cimiento sobre el que se construyen estas plataformas.

Amenazas Comunes en FinTech

Las apps financieras enfrentan riesgos únicos:

- **Fraude financiero**: Hackers que manipulan transacciones.

- **Robo de identidad**: Suplantación de usuarios para acceder a cuentas.
- **Ataques a APIs**: Muchas *FinTech* usan *APIs* para conectarse con bancos; si no están protegidas, son una puerta abierta.
- **Malware móvil**: Apps falsas que imitan servicios legítimos para robar datos.

Para operar legalmente, las *FinTech* deben navegar un laberinto de regulaciones. En Europa, el GDPR protege la privacidad de los datos; en el sector de pagos, el estándar PCI DSS exige medidas rigurosas; y normas como la PSD2 obligan a autenticaciones robustas. Incumplirlas no solo daña la reputación, sino que puede acarrear multas devastadoras.

¿Cómo se defienden estas plataformas? Empleando autenticación en dos pasos, cifrado de extremo a extremo y sistemas de monitoreo que detectan actividades sospechosas en tiempo real. Muchas también contratan hackers éticos para que prueben sus defensas antes de que lo hagan los criminales.

Pero los retos son muchos. Algunas *FinTech* dependen de sistemas bancarios antiguos y vulnerables. Otras, en su afán por crecer rápido, descuidan protocolos de seguridad. Y aunque la tecnología avanza —con soluciones como IA para detectar fraudes o *blockchain* para transacciones más seguras—, la batalla contra los ciberataques nunca termina.

El futuro del área es prometedor, pero su éxito depende de que los usuarios confíen en ellas. Por eso, cada vez se invierte más en biometría, inteligencia artificial y otras innovaciones que refuercen la seguridad. Para quienes buscan una carrera en tecnología o finanzas, este sector ofrece un campo lleno de oportunidades.

En resumen, este mundillo está transformando el mundo financiero, pero su impacto solo perdurará si la seguridad es prioridad. No se trata solo de hacer las cosas más fáciles, sino de hacerlas más seguras para todos.

Vulnerabilidades Comunes para la Manipulación de Precios en E-Commerce

1. Parameter Tampering (Modificación de Parámetros)

- **Descripción**: Alteración manual de parámetros en peticiones HTTP (GET/POST) para cambiar precios, descuentos o cantidades.
- **Ejemplo**:

```
POST /checkout HTTP/1.1
{"product_id": 101, "price": 100, "quantity": 1} → Cambiar a "price": 0.01
```

- **Pentesting**:
  - Usar **Burp Suite** para interceptar y modificar peticiones.
  - Buscar parámetros como price, total, discount_value en JSON/Form-Data.
- **Mitigación**: Validar precios en backend con referencia a una base de datos y usar firmas digitales (HMAC).

2. Abuso de Cupones (Coupon Reuse/Injection)

- **Descripción**: Reutilización de cupones de descuento, fuerza bruta de códigos o inyección de cupones no aplicables.
- **Ejemplo**:
  - Cupón de un solo uso (`OFF50`) reenviado en múltiples compras.
  - Inyección de cupones via parámetros ocultos: `?coupon=FREE100`.
- **Pentesting**:
  - Fuzzear endpoints de cupones con herramientas como **ffuf** o **Wfuzz**.
  - Analizar lógica de aplicación (¿el descuento se valida en frontend o backend?).
- **Mitigación**: Limitar usos por usuario, validar cupones en backend y evitar exposición de códigos en HTML/JS.

3. Manipulación de Carrito (Cart IDOR)

- **Descripción**: Acceso no autorizado a carritos de otros usuarios mediante IDs manipulables (Insecure Direct Object References).
- **Ejemplo**:

```
GET /api/cart?user_id=456 → Cambiar a user_id=123 (admin)
```

- **Pentesting**:
  - Probar enumeración de IDs (ej. cart_id=1001, 1002, etc.).
  - Verificar si los endpoints carecen de control de acceso (JWT, session tokens).
- **Mitigación**: Implementar autorización basada en roles y usar UUIDs en lugar de IDs secuenciales.

4. Negative Values (Valores Negativos)

- **Descripción**: Introducir valores negativos en campos como "cantidad" o "descuento" para generar créditos ilegítimos.
- **Ejemplo**:

```
POST /payment HTTP/1.1
```

`{"quantity": -5, "price": 10} → Total: -$50 (¡el sistema paga al usuario!)`

- **Pentesting**:
  - Enviar valores negativos en todos los campos numéricos.
  - Probar con decimales (ej. -0.01).
- **Mitigación**: Validar rangos numéricos (ej. quantity > 0) y usar tipos de datos unsigned en BD.

5. API Abuse (Llamadas Directas a APIs Internas)

- **Descripción**: APIs expuestas sin autenticación adecuada permiten manipular precios o saltarse flujos de compra.
- **Ejemplo**:

```
POST /api/admin/updatePrice HTTP/1.1
{"product_id": 789, "new_price": 0.01}
```

- **Pentesting**:
  - Buscar endpoints ocultos con **Burp Suite** (ej. /api, /v1/admin).
  - Probar métodos HTTP alternativos (PUT, PATCH).
- **Mitigación**: Restringir APIs con tokens JWT, rate-limiting y WAFs.

Bussiness Logic Vulnerabilities

¿Qué son las Bussiness Logic Vulnerabilities?

Son fallos en el diseño de una aplicación que permiten a un atacante explotar comportamientos no previstos para evadir restricciones, robar datos o manipular procesos.

Ejemplo básico:

- Una tienda online permite comprar **-1** unidades de un producto (¡y el sistema resta dinero en lugar de sumarlo!).

Diferencia con otros ataques:

- No son errores de código (como SQL Injection o XSS), sino errores en cómo se diseñó la funcionalidad.

Tipos Comunes de Vulnerabilidades de Lógica

Manipulación de Parámetros

¿En qué consiste?
Modificar valores en peticiones HTTP (ej. precios, cantidades, IDs) para saltarse validaciones.

Ejemplo:

- Cambiar "price": 100 → "price": 0.01 en una petición de compra.

Herramientas para probarlo:

- Burp Suite (interceptar/modificar peticiones).
- Browser DevTools (editar campos ocultos en HTML/JavaScript).

Abuso de Flujos de Aplicación

¿En qué consiste?
Saltarse pasos críticos (como verificación de pago o autenticación) alterando el orden normal de una transacción.

Ejemplo:

1. Un flujo de compra normal:
   Añadir al carrito → Pagar → Verificar tarjeta → Confirmar compra.
2. Ataque: Acceder directamente a /confirmar-compra sin pagar.

Cómo detectarlo:

- Mapear todos los endpoints con OWASP ZAP.
- Probar saltarse pasos con Repeater (Burp Suite).

Validación Insegura en Cliente

¿En qué consiste?
La app confía en validaciones hechas en el navegador (JavaScript) que pueden eludirse.

Ejemplo:

- Un cupón de descuento se verifica solo en JavaScript, pero el backend no lo comprueba.
- Solución: Desactivar JS o modificar la petición HTTP.

Mitigación:

- Nunca confiar en validaciones de frontend. Usar checks en backend.

Condiciones de Carrera (Race Conditions)

¿En qué consiste?
Explotar el tiempo entre dos operaciones para obtener beneficios (ej. duplicar saldos).

Ejemplo clásico:

1. Tienes $100 en tu cuenta.
2. Realizas dos transferencias de $100 casi al mismo tiempo.
3. Si el sistema no bloquea recursos, ambas operaciones podrían ser válidas (¡gastas 200teniendosolo200teniendosolo100!).

Herramientas para explotarlo:

- Scripts con Python + Requests para enviar peticiones concurrentes.

¿Cómo Encontrar Estas Vulnerabilidades?

Metodología para Pentesters

1. Entender el flujo normal: ¿Cómo funciona la app para un usuario legítimo?
2. Identificar puntos de decisión: ¿Dónde la app valida permisos, precios o acceso?
3. Probar manipulaciones:
    - Cambiar parámetros (números, textos, IDs).
    - Saltarse pasos.

    - Repetir acciones rápidamente (Race Conditions).

Herramientas Clave

- Burp Suite: Para interceptar/modificar peticiones.
- OWASP ZAP: Automatizar búsqueda de endpoints.
- Python + Requests: Para pruebas de Race Conditions.

¿Por qué son Peligrosas Estas Vulnerabilidades?

- Impacto alto: Pueden llevar a fraudes financieros, robos de datos o acceso no autorizado.
- Difíciles de detectar: No aparecen en scanners automáticos (como SQLi o XSS).
- Dependen del contexto: Requieren entender cómo funciona la app.

Buenas Prácticas para Desarrolladores

- Validar siempre en backend.
- Usar transacciones atómicas (evitar Race Conditions).
- Mapear todos los flujos críticos (pagos, autenticación).

¿Cómo me puedo proteger?

1. Principios de Diseño Seguro

A. Validación Estricta en Backend

- **Nunca confiar en el frontend**: Cualquier validación realizada en JavaScript puede ser eludida.
    - **Ejemplo**: Si un cupón de descuento se valida solo en el navegador, un atacante puede modificar la petición HTTP para aplicar descuentos no autorizados.
    - **Solución**: Todas las reglas de negocio (precios, descuentos, permisos) deben verificarse en el servidor.

B. Mecanismos de Auditoría y Logging

- **Registrar todas las acciones críticas**:
    - Cambios de precios.
    - Modificaciones en saldos o transacciones.
    - Intentos de acceso no autorizados.
- **Ejemplo**: Si un usuario intenta comprar -5 productos, el sistema debe registrar el evento y alertar al equipo de seguridad.

C. Transacciones Atómicas (Para Evitar Race Conditions)

- Usar **bloqueos (locks)** o **mutex** en operaciones críticas (ej. transferencias de dinero).
- **Ejemplo práctico**:
    - Si dos solicitudes de compra llegan al mismo tiempo, el sistema debe procesarlas en secuencia, no en paralelo.

2. Técnicas de Mitigación para Desarrolladores

A. Sanitización de Entradas

- Asegurarse de que los datos cumplan con lo esperado:
    - **Ejemplo**: Si un campo debe ser un número positivo, rechazar valores negativos o decimales.

B. Límites de Tasa (Rate Limiting)

- Evitar abuso de funcionalidades (ej. enviar 1000 cupones en segundos).
    - **Herramientas útiles**:
        - **API Gateways** (Kong, AWS API Gateway).
        - **Reglas en WAFs** (Cloudflare, ModSecurity).

C. Revisión de Flujos Críticos

- **Preguntas clave**:
  - ¿Qué pasa si un usuario salta un paso en el checkout?
  - ¿Se puede reutilizar un token de pago?
  - ¿Qué ocurre si se envía un valor nulo en un campo obligatorio?

3. Estrategias para Pentesters y Equipos de Seguridad

A. Pruebas Manuales de Lógica de Negocio

- **Técnicas útiles**:
  - **Interceptar y modificar peticiones** (Burp Suite, OWASP ZAP).
  - **Probar valores extremos**:
    - Números negativos.
    - Strings largos (¿rompe la lógica?).
    - IDs aleatorios (¿hay Insecure Direct Object References?).

B. Análisis de Casos de Uso Anómalos

- **Ejemplo de prueba**:
  1. Añadir un producto al carrito.
  2. Cambiar el precio vía Burp Suite.
  3. Verificar si el backend acepta el cambio.

C. Automatización de Pruebas (QA + Security)

- Usar **scripts** (Python + Requests) para simular Race Conditions.
- Integrar **pruebas de seguridad en CI/CD** (GitHub Actions, GitLab CI).

¡Manos a la obra!

Sí te interesa vivir de primera mano los fundamentos de ataques al área FinTech, en específico, a los E-Commerce, adjuntamos un pequeño grupo de laboratorios con el fin de profundizar tus conocimientos en el área.

OWASP Juice Shop

- **Enfoque**: Aplicación moderna (Node.js) con docenas de vulnerabilidades, incluyendo manipulación de precios, JWT flaws, y APIs inseguras.
- **Descarga**: https://github.com/juice-shop/juice-shop
- **Ventaja**: Incluye un **tutorial interactivo** y está diseñado para aprender de forma guiada.

DVWA + E-Commerce Module

- **Enfoque**: Modifica **Damn Vulnerable Web App (DVWA)** para simular un e-commerce básico.
- **Recurso**: Usa el módulo **"Business Logic"** de DVWA para practicar manipulación de precios.
- **Setup**: https://github.com/digininja/DVWA

PortSwigger Web Security Academy (Burp Suite)

- **Laboratorios**: Tienen ejercicios específicos de **lógica de negocio** (incluyendo modificación de precios).
- **Acceso**: https://portswigger.net/web-security/all-labs#business-logic
- **Ejemplo**: *"Lab: Excessive trust in client-side controls"*.

TryHackMe / HackTheBox

- **TryHackMe - Room "E-Commerce"**:
  - Escenario: Explotar un carrito de compras vulnerable.
  - Enlace: https://tryhackme.com (busca "e-commerce").
- **HackTheBox - Máquina "Bastion"**:
  - Incluye vulnerabilidades en aplicaciones web similares a e-commerce.

Laboratorios para APIs Inseguras (Relevantes para E-Commerce)

A. crAPI (Completely Ridiculous API)

- **Enfoque**: API vulnerable con endpoints de compras, carrito y perfiles.
- **GitHub**: https://github.com/OWASP/crAPI
- **Ataque clave**:

B. Vulnerable REST API (vAPI)

- **GitHub**: https://github.com/roottusk/vAPI
- **Incluye**: Endpoints simulando pagos y descuentos.