12/02/2026

Exercice 5

Rapport

COUPEAU Laurent
xxxxx

Table des matières

1. Introduction

This report presents the complete **security testing strategy** for the ShopNow ecommerce platform. It builds directly on:

- **Exercise 3:** Security requirements
- **Exercise 4:** Zero Trust architecture

The CTO requires a **comprehensive, risk-based, STRIDE-aligned test plan** to ensure that:

- security requirements are implemented correctly,
- controls are effective and measurable,
- sensitive flows are protected,
- and the platform is resilient against real-world attacks.

This report defines:

1. A mapping of **requirements → tests**
2. A **test strategy per security zone**
3. A **test plan for sensitive flows (F1, F2, F4)**
4. Integration of tests into a **DevSecOps CI/CD pipeline**
5. A prioritization of tests and KPIs to measure effectiveness

2. Mapping Security Requirements to Tests

Below is a table mapping **10+ security requirements** (from Exercise 3) to **specific test types**, with justification and acceptance criteria.

2.1 Requirements → Tests Mapping Table

Req ID	Security Requirement	STRIDE Threat	Test Type	Acceptance Criteria
S1	Tokens MUST be stored in HttpOnly, Secure, SameSite cookies	Spoofing	Browser security test + DAST	Token not accessible via JS; cookie flags correctly set
S2	MFA mandatory for admins	Spoofing	Functional test + manual test	Admin login impossible without MFA
T1	HMAC signatures on sensitive requests	Tampering	API integrity test	Modified requests rejected with 401/403
T2	JWTs signed with RS256/HS256	Tampering	Unit test + DAST	Tampered JWT rejected; signature validated
I1	TLS 1.3 mandatory	Info Disclosure	TLS configuration test	No TLS downgrade; only strong ciphers allowed
I2	DB encrypted with AES-256	Info Disclosure	Configuration audit	DB encryption enabled and verified
D1	Rate limiting on APIs	DoS	Load test	429 returned after threshold
E1	Strict RBAC	Elevation	Authorization test	Unauthorized roles cannot access protected endpoints
LOG1	Immutable logs	Repudiation	Log integrity test	Logs cannot be modified without detection
PAY1	Payment requests must be signed	Tampering	API test + negative test	Unsigned or modified payment requests rejected
AUTH1	Token rotation on refresh	Spoofing	Functional test	Refresh token replaced on each use

This mapping ensures that **every requirement is testable**, measurable, and aligned with STRIDE.

3. Test Strategy per Security Zone

Zero Trust requires **testing each zone independently** because each has different threats, assets, and controls.

3.1 Zone 1 — Front (Client, C1, F1, F3, F5)

Test Types

- Front-end security tests (XSS, CSRF)
- DAST on exposed endpoints
- Browser cookie tests
- UI tests for authentication flows
- Client-side validation bypass tests

Objectives

- Detect XSS and CSRF vulnerabilities
- Ensure secure token handling (S1)
- Validate that client cannot bypass server-side controls
- Ensure TLS enforcement (I1)

3.2 Zone 2 — Backend Secured (C2, C5, C4, F4, F6)

Test Types

- API security tests
- Fuzzing
- RBAC tests
- Input validation tests
- DoS resilience tests
- Authentication flow tests

Objectives

- Detect injection vulnerabilities (T2, T3)
- Validate RBAC (E1)
- Validate MFA (S2)
- Validate rate limiting (D1)

- Ensure token verification (T2)
- Ensure HMAC signatures (T1)

3.3 Zone 3 — Data Zone (C3, D1, D2, D3, D6)

Test Types

- DB privilege tests
- Encryption verification
- Data integrity tests
- Backup/restore tests
- Access control tests

Objectives

- Ensure DB encryption (I2)
- Validate least privilege DB accounts (E3)
- Prevent SQL injection (T3)
- Ensure data integrity (T)

3.4 Zone 4 — External Zone (C6, A3)

Test Types

- Integration tests with payment provider
- Error handling tests
- Signature validation tests
- Timeout and retry tests

Objectives

- Ensure secure communication with payment provider
- Validate HMAC signatures (T1)
- Ensure minimal data sharing (I)
- Ensure resilience to external failures (D)

4. Test Plan for Sensitive Flows (F1, F2, F4)

Sensitive flows must be tested with **high priority** because they involve identity, payments, and orders.

4.1 F1 — Authentication Flow

Critical Test Cases

Test Case	STRIDE Threat	Requirement	Expected Result
Login with invalid credentials	Spoofing	S2	Account not accessible
Brute force login	Spoofing/DoS	D1	Rate limiting triggers (429)
Token reuse	Spoofing	AUTH1	Reused token rejected
Tampered JWT	Tampering	T2	Token rejected
Token theft simulation	Spoofing	S1	Token inaccessible via JS

4.2 F2 — Payment Flow

Critical Test Cases

Test Case	STRIDE Threat	Requirement	Expected Result
Payment without authentication	Spoofing	E1	Request rejected
Modified amount	Tampering	T1	Rejected due to invalid HMAC
Replay of payment request	Tampering	T1	Rejected as duplicate
Payment with invalid signature	Tampering	T1	Rejected
Payment provider unavailable	DoS	D1	Graceful fallback

4.3 F4 — Orders Flow

Critical Test Cases

Test Case	STRIDE Threat	Requirement	Expected Result
Access another user's order	Elevation	E1	Access denied
Modify order after payment	Tampering	T1	Rejected
Create order without auth	Spoofing	S2	Rejected
Missing audit trail	Repudiation	LOG1	Logged with unique ID

5. Integration into CI/CD (DevSecOps)

Security tests must be automated and integrated into the pipeline.

5.1 Pipeline Integration

At Each Commit

- SAST
- Secret scanning
- Unit tests (including JWT verification logic)

During Build

- Dependency scanning (SCA)
- Configuration checks (TLS, cookie flags)

On Staging

- DAST
- API fuzzing
- RBAC tests
- Rate limiting tests

Before Production

- Manual pentest

- Payment flow tests
- Admin flow tests

Periodic

- Monthly pentest
- Quarterly red team exercise
- Continuous monitoring

6. Trade-Offs: Coverage vs. Velocity

High coverage → slower pipeline

- DAST and fuzzing are slow
- Pentests require manual effort

Velocity → risk of missing vulnerabilities

- Fast pipelines may skip deep tests

Balanced Strategy

- Fast tests (SAST, SCA, unit tests) → every commit
- Medium tests (DAST, RBAC tests) → staging
- Heavy tests (pentest) → monthly

7. Non-Negotiable Tests for ShopNow

These tests must always run:

- SAST
- SCA
- RBAC tests
- JWT signature tests
- Rate limiting tests
- TLS configuration tests
- Payment signature tests
- MFA tests for admin

These protect the most critical assets and flows.

8. Tests That Can Be Added Progressively

- Full fuzzing
- Red team exercises
- Chaos engineering for DoS resilience
- Advanced behavioral anomaly detection tests

9. KPIs to Measure Test Strategy Effectiveness

- % of critical flows covered by automated tests
- Mean Time to Detect (MTTD) vulnerabilities
- Mean Time to Fix (MTTF) vulnerabilities
- Number of vulnerabilities found per release
- % of tests passing in CI/CD
- Number of blocked credential stuffing attempts
- Number of rejected tampered requests
- Time to run full pipeline

10. Conclusion

This security test plan ensures that:

- All critical requirements from Exercise 3 are validated
- Zero Trust controls from Exercise 4 are enforced
- Sensitive flows (auth, payment, orders) are protected
- The platform is resilient against STRIDE threats
- Security is integrated into the DevSecOps lifecycle

This plan provides a **comprehensive, risk-based, and actionable** approach to validating ShopNow's security posture.

exemple de pipeline

```yaml
# ------------------------------------------------------------
# ShopNow — Secure DevSecOps Pipeline
# Objectif : intégrer la sécurité à chaque étape du cycle
# Aligné avec STRIDE, Zero Trust et les exigences Exercice 3
# ------------------------------------------------------------

stages:
  - precommit      # Vérifications avant commit
  - sast           # Analyse statique du code
  - build          # Build + scan des dépendances
  - test           # Tests unitaires
  - security       # Tests de sécurité dynamiques
  - staging        # Déploiement en staging
  - pentest        # Pentest manuel
  - deploy         # Déploiement en production

variables:
  NODE_ENV: test
```

```yaml
# ------------------------------------------------------------
# 1. PRE-COMMIT : Empêcher les erreurs avant qu'elles arrivent
# ------------------------------------------------------------
precommit:
  stage: precommit
  script:
    - echo "🔍 Secret scanning..."
    - trufflehog filesystem .        # Détecte les secrets/API keys dans le code
    - echo "🔎 Linting..."
    - npm run lint                   # Vérifie la qualité du code
  allow_failure: false               # Bloque le pipeline si échec
  tags:
    - security
```

```yaml
# ------------------------------------------------------------
# 2. SAST : Analyse statique du code
# Détecte injections, mauvaises pratiques, vulnérabilités
# ------------------------------------------------------------
sast:
  stage: sast
  script:
    - echo "🛡️ Running SAST..."
    - npm audit --json               # Analyse des dépendances Node
    - semgrep --config auto          # Analyse statique multi-langages
  artifacts:
    paths:
      - sast-report.json
  allow_failure: false
```

```yaml
# ------------------------------------------------------------
# 3. BUILD : Construction + scan des dépendances + scan container
# ------------------------------------------------------------
build:
  stage: build
  script:
    - echo "📦 Installing dependencies..."
    - npm install
    - echo "🧩 Scanning dependencies..."
    - npm audit --production         # Vérifie les vulnérabilités en prod
    - echo "🐳 Building Docker image..."
    - docker build -t shopnow-backend .
    - echo "🔐 Scanning Docker image..."
    - trivy image shopnow-backend    # Scan CVE dans l'image Docker
  allow_failure: false
```

```yaml
# ------------------------------------------------------------
# 4. UNIT TESTS : Tests fonctionnels + sécurité logique
# JWT, RBAC, rotation tokens, etc.
# ------------------------------------------------------------
unit_tests:
  stage: test
  script:
    - echo "🧪 Running unit tests..."
    - npm run test:unit              # Tests classiques
    - echo "🔐 Testing JWT validation..."
    - npm run test:jwt               # Vérifie signature RS256/HS256
    - echo "🔐 Testing RBAC logic..."
    - npm run test:rbac              # Vérifie les rôles et permissions
  allow_failure: false
```

```yaml
# ------------------------------------------------------------
# 5. DAST : Tests dynamiques sur l'application en staging
# Fuzzing, rate limiting, paiement, auth
# ------------------------------------------------------------
dast:
  stage: security
  script:
    - echo "🕷️ Running DAST..."
    - zap-baseline.py -t https://staging.shopnow.com -r dast-report.html
    - echo "🌀 Running API fuzzing..."
    - restler fuzz --api_spec api.json
    - echo "🚦 Testing rate limiting..."
    - npm run test:ratelimit          # Vérifie DoS (D1)
    - echo "💳 Testing payment flow..."
    - npm run test:payment            # Vérifie HMAC, tampering (T1)
  artifacts:
    paths:
      - dast-report.html
  allow_failure: false
```

```yaml
# ------------------------------------------------------------
# 6. SECURITY GATE : Bloque si vulnérabilités critiques
# ------------------------------------------------------------
security_gate:
  stage: security
  script:
    - echo "🚫 Checking for critical vulnerabilities..."
    - ./scripts/check_vulns.sh sast-report.json dast-report.html
  allow_failure: false

# ------------------------------------------------------------
# 7. MANUAL PENTEST : Validation humaine des flux critiques
# Auth, paiement, admin, élévation de privilèges
# ------------------------------------------------------------
manual_pentest:
  stage: pentest
  script:
    - echo "🛠️ Manual pentest required before production."
  when: manual
  allow_failure: false
```

```yaml
# ------------------------------------------------------------
# 7. MANUAL PENTEST : Validation humaine des flux critiques
# Auth, paiement, admin, élévation de privilèges
# ------------------------------------------------------------
manual_pentest:
  stage: pentest
  script:
    - echo "🛠️ Manual pentest required before production."
  when: manual
  allow_failure: false
```

```yaml
# ------------------------------------------------------------
# 8. DEPLOY : Déploiement en production
# Avec WAF, SIEM, monitoring Zero Trust
# ------------------------------------------------------------
deploy_prod:
  stage: deploy
  script:
    - echo "🚀 Deploying to production..."
    - kubectl apply -f k8s/          # Déploiement Kubernetes
  when: manual
  environment:
    name: production
```

GitHub Actions Version

```yaml
name: ShopNow Security Pipeline

on:
  push:
    branches: [ "main" ]
  pull_request:

jobs:

  precommit:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Secret scanning
        uses: trufflesecurity/trufflehog@main
      - name: Lint
        run: npm run lint
```

```yaml
  sast:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Run Semgrep
        uses: returntocorp/semgrep-action@v1
      - name: NPM Audit
        run: npm audit --json

  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Install dependencies
        run: npm install
      - name: Build Docker image
        run: docker build -t shopnow-backend .
      - name: Scan Docker image
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: shopnow-backend
```

1. Precommit

Objectif : empêcher les erreurs avant qu'elles n'entrent dans le repo. Tests effectués :

- Secret scanning → évite fuite de clés API
- Linting → qualité du code

Lien Zero Trust : éviter la compromission par erreurs humaines.

2. SAST

Objectif : détecter les vulnérabilités dans le code. Tests effectués :

- Semgrep → injections, mauvaises pratiques
- npm audit → dépendances vulnérables

Lien STRIDE : Tampering, Elevation, Information Disclosure.

3. Build

Objectif : construire un artefact sécurisé. Tests effectués :

- Scan des dépendances
- Scan de l'image Docker

Lien Zero Trust : intégrité du code et de la supply chain.

4. Unit Tests

Objectif : valider la logique de sécurité. Tests effectués :

- JWT signature
- RBAC
- Token rotation

Lien STRIDE : Spoofing, Elevation.

5. DAST

Objectif : tester l'application en conditions réelles. Tests effectués :

- ZAP (OWASP)
- Fuzzing
- Rate limiting
- Paiement (HMAC, tampering)

Lien STRIDE : DoS, Tampering, Spoofing.

6. Security Gate

Objectif : bloquer les builds non conformes. Tests effectués :

- Analyse des rapports SAST/DAST
- Vérification des vulnérabilités critiques

Lien Zero Trust : Never Trust, Always Verify.

7. Manual Pentest

Objectif : valider les flux critiques. Tests effectués :

- Auth bypass
- Payment tampering
- Admin escalation

Lien STRIDE : Elevation, Tampering, Spoofing.

8. Deploy

Objectif : déployer en production avec protections actives. Contrôles :

- WAF
- SIEM
- Monitoring

Lien Zero Trust : visibilité, traçabilité, contrôle dynamique.