

ShopNow

Building a better tomorrow for buyers and vendors
(especially vendors)



Part 5.....	3
Context & Objectives.....	3
Mapping Security Requirements to Tests.....	3
Test Strategy by Zone.....	5
Zone 1 : Client / Front-end (C1).....	5
Zone 3 : Secure Backend (C2, C5, C4).....	5
Zone 4 : Data (C3).....	5
Zone 5 : External (C6 - Stripe).....	5
Test Plan for Critical Flows.....	6
Flux F1: Authentication (Login/Refresh).....	6
Flux F2: Payment (Transaction).....	6
Flux F4: Orders (View/Modify).....	6
DevSecOps & CI/CD Integration.....	7
Pipeline Stages.....	7
Trade-offs & Velocity.....	7
Conclusion & KPIs.....	8
Non-Negotiable Tests.....	8
KPIs to Measure Success.....	8
Part 6.....	9
Security events mapping.....	9
D1 - Customers data.....	9
D2 - Orders data.....	9

D4 - Authentication tokens.....	10
D6 - Payment data.....	10
C2 - Backend.....	11
C3 - Databases.....	11
C5 - Authentication API.....	12
F1 - Authentication flow.....	12
F2 - Payment flow.....	13
F4 - Orders flow.....	13
A2 - Administrator accounts.....	13
Alert rules in SIEM.....	14
Incident response playbook.....	15
Customer Info Leaked.....	15
Denial of Service (DoS) on Backend Stack.....	16
DoS on Authentication API.....	16
Token Leaks on Authentication Flow.....	17
Zero-Trust enforcement.....	18
Basic steps of good detection metrics.....	18
Resources.....	18
Actors.....	18
Prevent alerts overflow.....	19

Part 5

Context & Objectives

Following the architectural redesign of the ShopNow e-commerce platform, we have defined a comprehensive set of security requirements based on STRIDE and designed a Zero Trust architecture.

To ensure these theoretical measures are effective in production, this document details the Security Test Plan. The objective is to validate that controls such as HMAC signing, mTLS, and Vault integration are correctly implemented and resilient against attacks.

Mapping Security Requirements to Tests

Based on the requirements identified, here is the mapping to specific verification methods.

ID	Requirement	STRIDE	Test Type	Verification Method
R1	Adaptive MFA Enforcement	Spoofing	Functional / Manual	Attempt login with correct password but without MFA. Expectation: Access Denied / Challenge trigger.
R2	HMAC Request Signing	Tampering	DAST / Proxy	Intercept request, modify payload body, replay request. Expectation: Backend rejects due to signature mismatch.
R3	TLS 1.3 & mTLS	Info Disclosure	Network Scan / config	Use <code>nmap --script ssl-enum-ciphers</code> or try connecting to Backend C2 without a client certificate. Expectation: Handshake failure.
R4	Rate Limiting	Denial of Service	Load Testing	Send 200 requests / minute (limit is 100). Expectation: HTTP 429 "Too Many Requests" after threshold.

R5	RBAC Strict Enforcement	Elevation	Authorization (AuthZ)	<p>Authenticate as a customer and attempt to have admin access.</p> <p>Expectation: Error 403 (Forbidden).</p>
R6	Secrets Management	Info Disclosure	SAST / Code Review	<p>Scan code repo for hardcoded secrets.</p>
R7	Input Validation & Parameterized Queries	Tampering	DAST / Fuzzing	<p>Inject SQL payloads into Search/Order fields.</p> <p>Expectation: Input sanitized or rejected; no DB error leakage.</p>
R8	Short-lived JWT & Rotation	Spoofing	Functional	<p>Wait for TTL expiry (15 min) and attempt reuse. Attempt to use an old Refresh Token.</p> <p>Expectation: Error 401 (Unauthorized).</p>
R9	Immutable Logging (WORM)	Repudiation	Integrity Test	<p>Attempt to modify a log entry via the logging service account.</p> <p>Expectation: Write operation rejected.</p>
R10	Egress Filtering	Info Disclosure	Network / Pentest	<p>Attempt to curl an external endpoint from the Backend container.</p> <p>Expectation: Connection timeout/refused.</p>

Test Strategy by Zone

We apply a "Defense in Depth" testing strategy, adapting the tools to the specific risks of each defined zone.

Zone 1 : Client / Front-end (C1)

- **Target:** React App, Browser environment.
- **Risks:** XSS, Token exfiltration.
- **Test Types:**
 - **SAST (Static Application Security Testing):** Use tools like **ESLint-plugin-security** or **SonarQube** to detect unsafe React patterns (`dangerouslySetInnerHTML`).
 - **Client-Side Dependency Scanning:** `npm audit` to check for vulnerable frontend libraries.
 - **Token Storage Verification:** Browser DevTools inspection to ensure JWTs are stored in `HttpOnly` cookies and not `localStorage`

Zone 3 : Secure Backend (C2, C5, C4)

- **Target:** Node.js API, Auth Service, Redis.
- **Risks:** Business Logic flaws, Injection, Broken Access Control.
- **Test Types:**
 - **DAST (Dynamic Application Security Testing):** Run **OWASP ZAP** against the API endpoints to identify injection points and misconfigurations.
 - **Unit/Integration Security Tests:** Write tests using **Mocha/Jest** that specifically check failure cases ("Should return 403 if user is not owner of order").
 - **Container Scanning:** Scan Docker images (Trivy/Claire) before deployment to ensure the OS/Runtime has no known CVEs.

Zone 4 : Data (C3)

- **Target:** PostgreSQL Database (PII, Orders).
- **Risks:** Data leakage, unauthorized access.
- **Test Types:**
 - **Configuration Audit:** Verify encryption at rest (AES-256) is enabled.
 - **Privilege Audit:** Verify that the Backend service account cannot execute `DROP TABLE` or `GRANT` commands.

Zone 5 : External (C6 - Stripe)

- **Target:** Payment Integration.
- **Risks:** Man-in-the-Middle, API Contract modification.
- **Test Types:**
 - **Contract Testing:** Verify that data sent to Stripe matches their strict API specifications.

- **Mocking:** Use mock servers to simulate Stripe outages or error responses to ensure the Backend handles failures gracefully (Circuit Breaker testing)

Test Plan for Critical Flows

These flows represent the "Crown Jewels" of ShopNow.

Flux F1: Authentication (Login/Refresh)

- **Threats:** Credential Stuffing, Brute Force, Session Hijacking.
- **Critical Test Cases:**
 - **TC-AUTH-01 (Rate Limiting):** Attempt 10 logins in 1 minute from the same IP.
 - Expected Result: Account locked or IP banned (HTTP 429).
 - **TC-AUTH-02 (Token replay):** Capture a valid JWT, log out, then attempt to use the captured JWT.
 - Expected Result: 401 Unauthorized (Token must be blacklisted in Redis).
 - **TC-AUTH-03 (Weak Password):** Attempt to register with "123456".
 - Expected Result: Rejected by password policy.

Flux F2: Payment (Transaction)

- **Threats:** Tampering, Bypass.
- **Critical Test Cases:**
 - **TC-PAY-01 (Integrity Check):** Intercept the payment request and change `amount: 50.00` to `amount: 1.00` without updating the HMAC signature.
 - Expected Result: Backend rejects request with "Signature Verification Failed".
 - **TC-PAY-02 (Direct API Access):** Attempt to call the Stripe API endpoint directly from the Client, bypassing the Backend.
 - Expected Result: Failed (Stripe API keys are hidden in Backend Vault).

Flux F4: Orders (View/Modify)

- **Threats:** IDOR (Insecure Direct Object Reference), Information Disclosure.
- **Critical Test Cases:**
 - **TC-ORD-01 (Horizontal Escalation/IDOR):** User A (ID: 101) calls `GET /api/orders/202` (belonging to User B).
 - Expected Result: 403 Forbidden. The system must verify `Order.ownerID == currentUser.ID`
 - **TC-ORD-02 (SQL Injection):** Attempt to retrieve orders with ID `' OR '1'='1`
 - Expected Result: Input validated, returns 404 or empty list, not the full database.

DevSecOps & CI/CD Integration

To ensure Security by Design without slowing down the Agile delivery, we integrate tests into the CI/CD pipeline (GitLab CI/Jenkins).

Pipeline Stages

Stage	Action	Tooling	Blocking?
Commit / PR	SAST: Scan changed code for vulnerabilities. Secret Scanning: Check for keys.	SonarQube TruffleHog	Yes (High/Critical)
Build	SCA: Scan dependencies (npm). Unit Security Tests: Run Jest security assertions.	npm audit Mocha	Yes
Deploy (Staging)	DAST: Run automated attacks against the running app. Container Scan: Check Docker image.	OWASP ZAP (Baseline) Trivy	No (Report only, unless Critical)
Deploy (Prod)	Smoke Test: Verify SSL/TLS and basic availability.	SSL Labs	Yes

Trade-offs & Velocity

- DAST Latency:** Full DAST scans take time.
Strategy: Run a "light" scan on every PR (Top 10 OWASP) and a "deep" scan nightly or weekly.
- False Positives:** Automated tools generate noise.
Strategy: Security team tunes the rulesets. Developers can override a block only with a valid reason (Risk Acceptance).

Conclusion & KPIs

Non-Negotiable Tests

For ShopNow to launch the new architecture safely, the following are mandatory:

1. **Identity Verification:** Automated tests proving that no resource is accessible without a valid, signed JWT.
2. **Payment Integrity:** HMAC verification tests on F2 flux.
3. **Isolation:** Proof that the Database (Zone 4) is unreachable directly from the Internet.

KPIs to Measure Success

To evaluate the strategy:

- **Vulnerability Remediation Time (MTTR):** Average time to fix a vulnerability found by SAST/DAST.
- **Test Coverage:** Percentage of critical flows (F1, F2, F4) covered by automated security tests.
- **Defect Density:** Number of security issues found in Production vs. Staging (Goal: 0 in Prod).

By automating these controls, ShopNow moves from a reactive posture to a proactive Zero Trust posture.

Part 6

Security events mapping

Identify each critical assets has been done, but properly identify which solutions may predict and/or detect wrongful usage/usurpation/active spying on resources requires knowledge.

To understand the scope of each word used, here is a brief lexicography as to let you best understand classification :

Criticality :

- P1 => Highest
- P2 => High
- P3 => Low
- P4 => Lowest

D1 - Customers data

STRIDE vulnerabilities

DANGER	EXEMPLE
<i>Spoofing</i>	Identity theft and phishing
<i>Tampering</i>	Changing personnel info for fraud purposes
<i>Info disclosure</i>	GDPR info leaking means judiciary and reputation challenges

Priorities

LOG TO TRACK	LOG CRITICALITY
Active sessions + IP address + Technical stack	P2
Last modified info + date of change + IP address	P4

D2 - Orders data

STRIDE vulnerabilities

DANGER	EXEMPLE
<i>Tampering</i>	Changing order volumes
<i>Info disclosure</i>	Customer profiling from spending habits and products type

Priorities

LOG TO TRACK	LOG CRITICALITY
Orders placed	P4

Orders placed are not of any interest as itself, but a monitoring solution would be to monitor sudden change on an account (like an increase of more than 200% from the average order per month in a single day). While a sudden shopping-spree can be a legitimate practice (job bonus, inheritance, house moving, etc) it is kinda normal to track it and raise an alert if suspicions arise.

D4 - Authentication tokens

STRIDE vulnerabilities

DANGER	EXEMPLE
<i>Spoofing</i>	Account takeover by token stealing
<i>Tampering</i>	JWT payload modification
<i>Info disclosure</i>	Tokens leaking

Priorities

LOG TO TRACK	LOG CRITICALITY
Active authentication + IP addresses	P3

D6 - Payment data

STRIDE vulnerabilities

DANGER	EXEMPLE
<i>Repudiation</i>	Can lead to bank ban on account
<i>Info disclosure</i>	Banking informations leaking
<i>Denial of service</i>	A wrongful usage may lead to a platform banning from provider

Priorities

LOG TO TRACK	LOG CRITICALITY
Active sessions + sessions ending	P2
Payment provider return codes	P1

C2 - Backend

STRIDE vulnerabilities

DANGER	EXEMPLE
Spoofing	Stolen keys or secrets may spoof API
Tampering	SQL attack via injection will alter tables
Repudiation	No log won't help us in judiciary cases
Info disclosure	Errors can leak data
Denial of service	API saturation will shutdown service
Elevation of privilege	Server takeover possible

Priorities

LOG TO TRACK	LOG CRITICALITY
Active connections with IP and actions	P1
CPU/RAM load charge	P3
API requests per minute	P2
4xx and 5xx codes	P2

C3 - Databases

STRIDE vulnerabilities

DANGER	EXEMPLE
Spoofing	Stolen DB credentials allows for unauthorized access
Tampering	Table changes from within
Info disclosure	A full database info leaking can close a business
Denial of service	High load on DB can lead to latency or crash
Elevation of privilege	SU access can force down databases and thus entire stack

Priorities

LOG TO TRACK	LOG CRITICALITY
Active connections + IP addresses	P1
Queries per seconds	P2

CPU/RAM load charge	P4
---------------------	----

C5 - Authentication API

STRIDE vulnerabilities

DANGER	EXEMPLE
<i>Spoofing</i>	Token spoofing to increase wrongful usage
<i>Tampering</i>	Can pass on false data within pipeline
<i>Info disclosure</i>	Token leaking in logs and errors
<i>Denial of service</i>	Endpoint saturation and shutdown
<i>Elevation of privilege</i>	Direct SU usage within production pipeline

Priorities

LOG TO TRACK	LOG CRITICALITY
Active pipeline	P2
Endpoint healthcheck	P1
Latency	P4

F1 - Authentication flow

STRIDE vulnerabilities

DANGER	EXEMPLE
<i>Spoofing</i>	Credentials stealing
<i>Info disclosure</i>	Credentials leaking

Priorities

LOG TO TRACK	LOG CRITICALITY
Start + end of sessions	P1
Error codes during client-server handshake	P2

F2 - Payment flow

STRIDE vulnerabilities

DANGER	EXEMPLE
<i>Tampering</i>	Changing payment amounts
<i>Info disclosure</i>	Leaking payment/banking data

Priorities

LOG TO TRACK	LOG CRITICALITY
Provider active session + end of session	
Provider return codes	

F4 - Orders flow

STRIDE vulnerabilities

DANGER	EXEMPLE
<i>Info disclosure</i>	Accessing user orders

Priorities

LOG TO TRACK	LOG CRITICALITY
Orders placing with customer ID	P3

A2 - Administrator accounts

STRIDE vulnerabilities

DANGER	EXEMPLE
<i>Tampering</i>	Can alter data directly
<i>Info disclosure</i>	Can leak data from databases
<i>Denial of service</i>	Can shutdown entire stack
<i>Elevation of privilege</i>	Already having access on while stack and can open backdoors

Priorities

<i>No log to track actors connected in stack, and there shouldn't be any.</i>

Alert rules in SIEM

Alerting may be tweaked to better serve alerting purposes, without making it too massive.

MONITOR	CONDITION	DANGER TO PREVENT	PRIORITY
Failed logins	> 5 logins with error within 5 minutes	Potential password brute-forcing	P1
Inhabitual customer actions	Lot of personal info changing	Fraud suspicion	P2 <i>Could be a false positive</i>
Huge orders amount	High variation of orders within 5 minutes	Fraud suspicion / credentials stolen	P2 <i>Could be a false positive</i>
Port scanning	A scan is in progress on network	Denial of Service, Spoofing, Privilege elevation attempts, etc	P2
Login attempts from foreign IP	IP range from abroad trying endpoints or internal connections. Mostly Asia from Russia and China, including neighbours	Denial of Service, Spoofing, Privilege elevation attempts, etc	P1 <i>Autobanning must be activated on firewall</i>
High DB queries	High variation of queries within 5 minutes range per the average	Denial of Service, Spoofing, Privilege elevation attempts, etc	P1
API requests	High variation of request within 5 minutes range per the average	Denial of Service, Spoofing, Privilege elevation attempts, etc	P1
Endpoints latency	Healthcheck latency over 1000ms	Denial of Service, service/instance stuck	P1
CPU overloading	CPU usage over 90% for a sustained period of time (1 hour)	Denial of Service, Privilege elevation attempts	P3

RAM overloading	RAM usage over 90% for a sustained period of time (1 hour)	Denial of Service, Privilege elevation attempts	P3
-----------------	--	---	----

Incident response playbook

PS from Gauvain : As I'm not really into this field, I asked the help from a metasourcing authority (aka I asked an AI for insights), but nothing here is left uncomprehended and nothing has been pasted.

Customer Info Leaked

GDPR requires notification to the supervisory authority within 72 hours of becoming aware of the breach.

DETECTION

- Data Loss Prevention triggers, unusual queries on DB, or unexpected outbound traffic volume
- Dark web monitoring tools (already too late at this point)

Validation: Verify if the data is actually PII (Personally Identifiable Information). Classify the data sensitivity (High/Medium/Low).

CONTAINMENT

- Isolate the affected system/server from the network
 - Revoke access credentials
 - Take endpoints offline or switch to private

Legal: Notify the Data Protection Officer (DPO) and Legal team. 72 hours to address it publicly.

ERADICATION

- Identify the root cause (Misconfiguration, SQL Injection, Phishing)
- Patch the vulnerability (apply security patch, fix permissions, sanitize code)
- Delete any compromised accounts or backdoors left by attackers

RECOVERY

- Restore data from a clean backup if data integrity was compromised
- Reset all passwords for all internal accounts

Notification: Under DPO guidance, notify the relevant Data Protection Authority (DPA) and affected data subjects if required

POST-MORTEM

- Implement stricter egress filtering and database activity monitoring (DAM)
- Update data classification policies and conduct GDPR awareness training

Denial of Service (DoS) on Backend Stack

Dedicated API and Database are overwhelmed.

DETECTION

- Spikes in HTTP 5xx errors (server error), CPU/Memory at 100%, increased database connection pool
- Access logs show high request volume from specific IPs or User-Agents
- Latency alerts

CONTAINMENT

- Enable higher Rate Limits (by IP or protocols) at WAF
- If needed, close up network access from outside both ways

ERADICATION

- Identify malicious IP ranges and block them at WAF
- If it's an Application Layer attack (complex search queries), disable that specific endpoint

RECOVERY

- Restart services
- Slowly route traffic back to the backend resource by resource
- Verify cache load

POST-MORTEM

- An temporary auto-scaler may-be in order to absorb high load (legitimate or not)
- Optimize heavy database queries with replicas if the DB was suffering
- Establish "Degraded Mode" features with limited queries and services

DoS on Authentication API

Attackers targeting `/login` or `/oauth/token`.

DETECTION

- High failure rate on endpoints
- High volume of distinct usernames from a single IP (range or address)
- Sudden spike in accounts errors

CONTAINMENT

- Enable MFA or Bot-killing solutions (like strong CAPTCHA)
- Enforce account blocking temp policy to lock-out hackers

ERADICATION

- Block requests based on malicious agents with pattern recognition
- Add suspects IPs to a ban list

RECOVERY

- Verify if MFA are still running
- Unlock legitimate user accounts that may have been auto-locked
- Prepare to absorb the amount of client complains

POST-MORTEM

- Dark Web monitoring to explore passwords leaks
- Hashing passwords in databases
- Enforce MFA for all account, no exception

Token Leaks on Authentication Flow

JWTs or Session IDs leaked, leading to impersonation.

DETECTION

- Access logs showing the same user connecting from different countries in a short time range
- Users claiming actions were taken on their account they didn't do
- SAST secret scanning or internal audits finding hard-coded tokens in repos or logs

CONTAINMENT

- Immediate revocation:
 - If using JWT: Rotate the signing keys (Private Key/Secret). *Note: This invalidates ALL users, forcing a global logout.*
 - If using Session IDs/Opaque Tokens: Delete the compromised tokens from the store (Redis/DB).
- Fix the Leak: If tokens are leaking via logs, stop the logging service or apply a filter immediately.

ERADICATION

- Patch the vulnerability allowing the leak (fix XSS vulnerability, sanitizing logs).
- Identify the scope: Was it one user or all users?
- If specific tokens were stolen, add their JTI (JWT ID) to a "Deny List" or Blacklist.

RECOVERY

- Force password resets for affected accounts (precautionary).
- Users will be forced to log in again.
- Monitor for "Account Recovery" attempts, as attackers might try to regain access.

POST-MORTEM

- Why were tokens visible ? Sanitation on codebase and logs are needed
- Shorten token expiration times. Try out action-based tokens for short-lived (like 5 minutes)
- Implement automated secret scanning in the CI/CD pipeline (if applicable)

Zero-Trust enforcement

Detection of errors, leaks and security failures should ALWAYS lead to a reinforcement of internal security, trigger code refactoring if needed and every action possible to take to ensure that no repetition could come.

Security leaks may be linked to technical debt or historical stack building at a time where tools were scarce if existent, but time flies by and changes must be implemented as to not compromise security.

Opening the flood of permissions, setting up accounts with stack-wide control and relying on old services to act through API and resources let the gates open for any malicious actor, as pattern recognition and specific long-term accesses to brute-force will end up as bad publicity.

Enforcing zero-trust as more and more leaks, too verbose logs and any point of failure are discovered is a priority as to contain any hack to the minimum attack surface possible.

Basic steps of good detection metrics

Resources

1. Track resources
2. Track resources actions
3. Track resources actions objectives
4. Track resources actions objectives outcomes

Actors

1. Track actors
2. Track actors actions
3. Track actors actions objectives
4. Track actors actions objectives outcomes

The good way to leverage metrics and logs and thus having an efficient monitoring dashboard and monitored stack is to not leave anything unwatched, however the reason.

It's best to have TOO MUCH logs than TOO FEW.

Then, everything happens for a reason. Analysing patterns and detecting variations is key to a good alerting system.

Monitoring a resource and how it interacts indicates that the network in his wholeness must be in direct control. If even one part is not in a private subnetwork, detached from the internal stack or dependent from a third-party networker, nothing complete can be achieved. So another principle for a good metric system is to be in complete and SOLE control of the stack.

Prevent alerts overflow

Let's face it and no sugar-coating : you won't. At least at first. Every bit of pattern that slightly goes off-rails will trigger an alert. Trust me on this : it will. It is then up to you to balance it out with the time being so false positives are left apart while only true alerts are raised.

To prevent an overdose of alerts, leading to "alert fatigue" (low concentration, higher risk of letting a true alert undetected/closed without thinking twice), the best is to put more people on RUN tasks, with more people watching about alerts as they rise, or use solutions with some AI insights fully trained to identify patterns and SUGGEST (not undertake) actions with clear and concise review capacities.

- You can choose to twinkle down alerts, but you would miss on a lot of triggers
- You can choose to tire your people until resignation (and cost engagement tied to turnover) if you don't let them get away from these mechanical, low-intellect fastidious tasks for all their week time
- You can simplify the stack to let each resource be fully monitored while decreasing the total amount of logs to watch, would your product permit this
- You can outsource these tasks to dedicated team, based in countries with 24h rolling teams or to AI agents with low-quality results requiring human manual correction

So best is to have a dedicated, in-house, fully-equipped team to rotate missions and avoid intellectual starvation and exhaustion.