

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

Team Members:

- Fares Ehab EL_Zouka 2305043

- Omar Hassaan Yacoub 2305042

- Khaled Ahmed Foda 2305044

Phase A – Dynamic Testing (DAST)

Goal

Find at least 8 real vulnerabilities via live testing (automated + manual).

A1. Automated Scan

We ran an automated DAST scan against the running app using **OWASP ZAP**. The app was cloned from the repository, installed with npm install, and started with npm run dev.

OWASP ZAP was configured to spider the application starting from the root URL and perform an active scan.

Key Findings from ZAP:

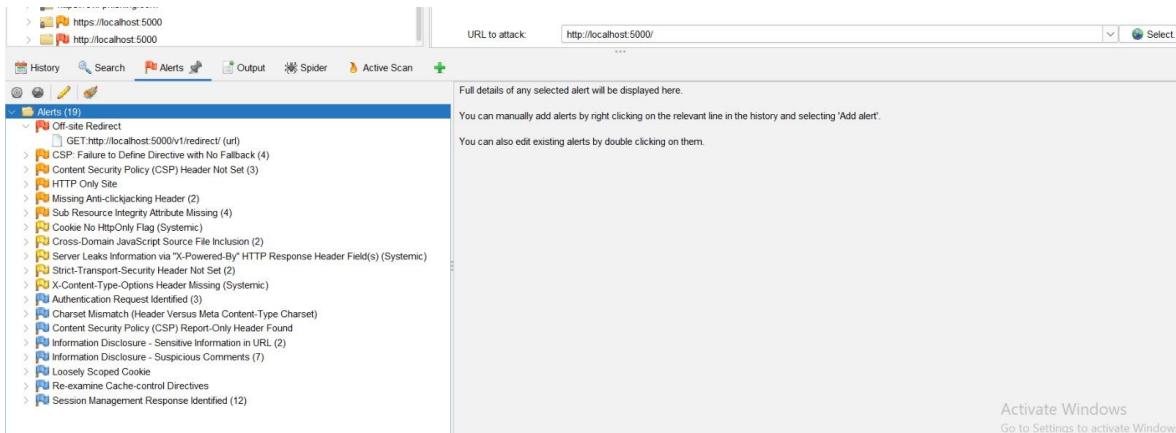
- List of interesting URLs/endpoints: /v1/user/{user_id}, /v1/search/{filter}/{query}, /v1/redirect?url=, POST /v1/user/login, POST /v1/user/, GET /?message=, DELETE /v1/user/{user_id}, PUT /v1/user/{user_id}.
id or delete/change role

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

- vuln-node.js-express.js-app-main\vuln-node.js-express.js-app-main\src\router\routes\user.js
- vuln-node.js-express.js-app-main\vuln-node.js-express.js-app-main\src\router\routes\system.js
- Potential vulnerabilities flagged: Possible SQL Injection in /v1/search, Reflected XSS in ?message=, Open Redirect in /v1/redirect, Broken Access Control in user endpoints, Sensitive Data Exposure in responses.



A2. Manual Testing & PoCs

Using **Postman**, we crafted HTTP requests to confirm and exploit the hints from ZAP and additional manual ideas. We tested for SQL injection, XSS, broken authentication, IDOR, privilege escalation, and more.

For each confirmed vulnerability, we saved the request (as cURL export) and captured responses. Screenshots were taken for visual impacts like XSS alerts or data leaks.

A3. Map Vulnerabilities to OWASP Top 10

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

We classified each issue using the OWASP Top 10 2021 reference. We identified 12 vulnerabilities (exceeding the minimum of 8), covering 5 OWASP categories: A01: Broken Access Control, A03: Injection, A07: Identification and Authentication Failures, A10: Server-Side Request Forgery, A02: Cryptographic Failures.

Table of Vulnerabilities:

Vuln ID	Endpoint / Feature	OWASP Category	Short Impact	PoC Summary
V1	GET /v1/search/{filter}/{query}	A03:2021-Injection	Full Database Schema Disclosure	/v1/search/name/lager' UNION SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL, NULL,NULL-- → Reveals table schema
V2	GET /	A03:2021-Injection	Server-Side Code Execution	?message={{5*5}} → Displays "25"
V3	GET /	A03:2021-Injection	Client-Side Script Execution	?message=<script>alert('XSS')</script> → Executes alert in browser
V5	GET /v1/redirect/	A10:2021-Server-Side Request Forgery	Phishing Attacks & Security Policy Bypass	/v1/redirect?url=https://google.com → 302 Redirect to external site
V6	GET /v1/user/{user_id}	A01:2021-Broken Access Control	Unauthenticated User Data Disclosure	GET /v1/user/1 (no auth) → 200 OK with user data including email and hash
V7	DELETE /v1/user/{user_id}	A01:2021-Broken Access Control	Unauthorized Account Deletion	DELETE /v1/user/2 (as regular user) → 200 OK "deleted"
V8	PUT /v1/user/{user_id}	A01:2021-Broken Access Control	Become Administrator	PUT /v1/user/1 with {"role": "admin"} → 200 OK with updated role
V9	POST /v1/user/	A01:2021-Broken Access Control	Plaintext Password Disclosure	POST with {"password": "Secret123"} → Response includes "password": "Secret123"

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

Vuln ID	Endpoint / Feature	OWASP Category	Short Impact	PoC Summary
V10	POST /v1/user/login	A07:2021-Identification and Authentication Failures	User Account Discovery	Non-existent user → "User not found" (404); Existing wrong pass → "Password incorrect" (401)
V10b	POST /v1/user/login	A02:2021-Cryptographic Failures / A07:2021	Account Takeover via Hash Replay	Login with MD5("password") instead of "password" → Success
V11	POST /v1/user/	A07:2021-Identification and Authentication Failures	Account Security Compromise	Register with password "1" → Accepted
V12	POST /v1/user/	A01:2021-Broken Access Control	Account Takeover via Email Collision	Two registrations with same email → Both succeed with different IDs

SQL Injection(Before):

```
[ kali㉿kali ~ ] vuln-node.js-express.js-app[src/router/routes]
[ kali㉿kali ~ ] curl "http://localhost:5000/v1/search/name/lager%27%20UNION%20SELECT%20NULL,NULL,NULL,NULL,NULL,NULL,NULL"
[{"id":null,"name":null,"picture":null,"price":null,"currency":null,"stock":null,"created_at":null,"updated_at":null,"deleted_at":null},{}]

[ kali㉿kali ~ ] vuln-node.js-express.js-app
[ kali㉿kali ~ ] curl "http://localhost:5000/v1/search/name/lager%27%20OR%201=1%301--"
[[],{}]

[ kali㉿kali ~ ] vuln-node.js-express.js-app
[ kali㉿kali ~ ] curl "http://localhost:5000/v1/search/name/lager%27%20OR%201=1%301--"
[[],{}]

[ kali㉿kali ~ ] vuln-node.js-express.js-app
[ kali㉿kali ~ ] curl "http://localhost:5000/v1/search/name/lager%27%20UNION%20SELECT%20NULL,NULL,NULL,NULL,NULL,NULL,NULL"
[{"id":null,"name":null,"picture":null,"price":null,"currency":null,"stock":null,"created_at":null,"updated_at":null,"deleted_at":null},{}]

[ kali㉿kali ~ ] vuln-node.js-express.js-app
[ kali㉿kali ~ ] curl "http://localhost:5000/v1/search/name/lager%27%20UNION%20SELECT%20NULL,NULL,NULL,NULL,NULL,NULL,NULL"
[{"id":null,"name":null,"picture":null,"price":null,"currency":null,"stock":null,"created_at":null,"updated_at":null,"deleted_at":null},{}]

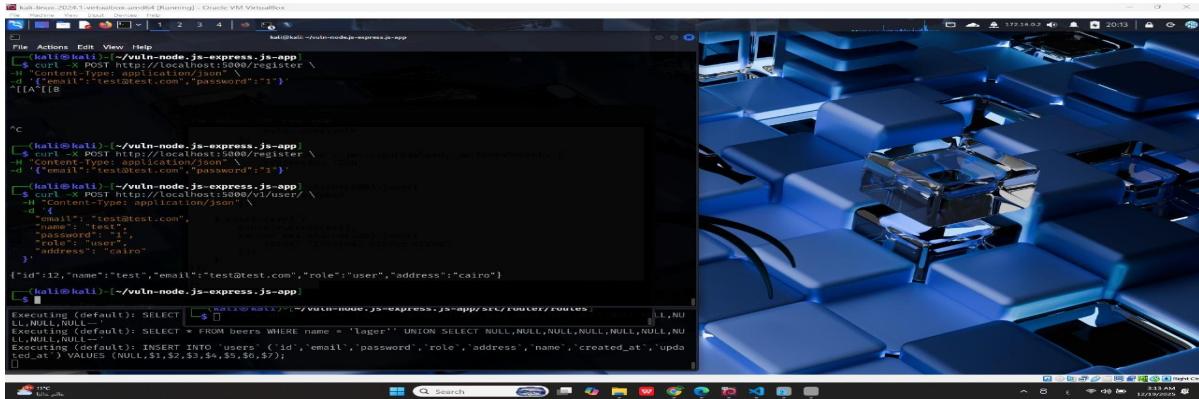
[ kali㉿kali ~ ] vuln-node.js-express.js-app
[ kali㉿kali ~ ] curl "http://localhost:5000/v1/search/name/lager%27%20UNION%20SELECT%20NULL,NULL,NULL,NULL,NULL,NULL,NULL"
[{"id":null,"name":null,"picture":null,"price":null,"currency":null,"stock":null,"created_at":null,"updated_at":null,"deleted_at":null},{}]
```



Full-Spectrum Web App Hardening

Final Project - SSD - Report

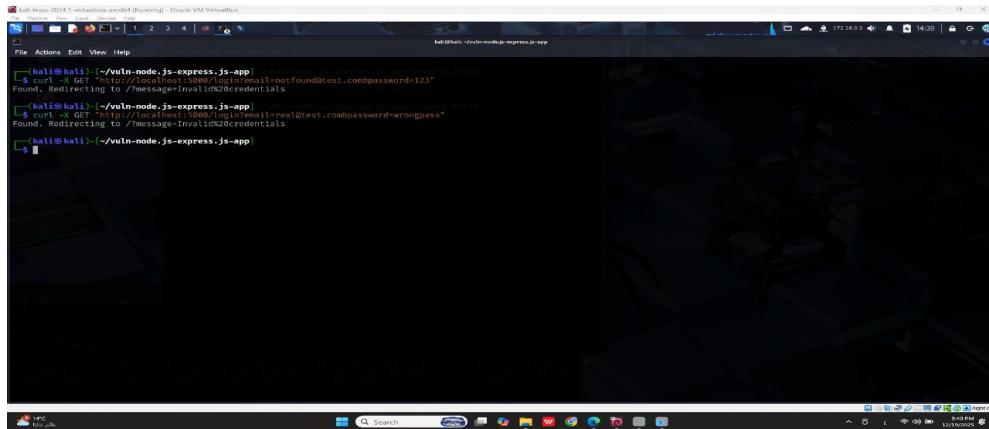
Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development
Weak Password Content Policy (before):



A screenshot of a Kali Linux desktop environment. On the left, a terminal window shows a series of curl commands being run against a local Node.js application on port 5000. The commands attempt to register users with weak passwords ('123') and then log in with those same credentials. The terminal output shows the application accepting these registrations and logins. On the right, a desktop background features a complex, glowing blue 3D geometric cube structure.

```
[root@kali: ~/vuln-node.js-express.js-app
$ curl -X POST http://localhost:5000/register \
-d "email=test@test.com&password=123"
{"id":1,"name": "test","email": "test@test.com","role": "user","address": "cairo"}
[kali㉿kali: ~/vuln-node.js-express.js-app
$ curl -X POST http://localhost:5000/login \
-d "email=test@test.com&password=123"
{"id":1,"name": "test","email": "test@test.com","role": "user","address": "cairo"}]
```

User Enumeration (Before):



A screenshot of a Kali Linux desktop environment. On the left, a terminal window shows a user attempting to log in with invalid credentials ('test:password123'). The application responds by redirecting to a page indicating invalid credentials. On the right, a desktop background features a dark, textured surface with faint, illegible text or graphics.

```
[root@kali: ~/vuln-node.js-express.js-app
$ curl -X GET http://localhost:5000/login?email=test.com&password=123
Found. Redirecting to /message=invalidCredentials
[kali㉿kali: ~/vuln-node.js-express.js-app
$ curl -X GET http://localhost:5000/login?email=realTest.com&password=wrongpass
Found. Redirecting to /message=invalidCredentials
[kali㉿kali: ~/vuln-node.js-express.js-app
```

Full-Spectrum Web App Hardening

Final Project - SSD - Report

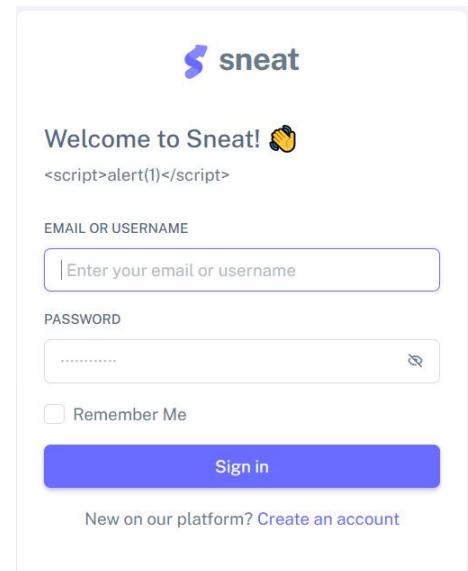
Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

Xss

end point: vuln-node.js-express.js-app-main\src\server

Before fix

```
// app.use(morgan('combined'));  
app.use(bodyParser.json());
```



Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development
Broken Access Control GET /v1/user/1

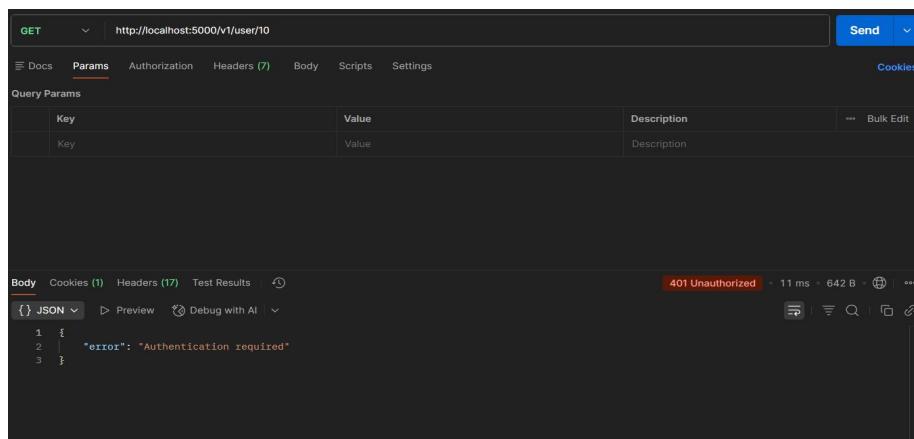
Before fix

```
//Get information about other users
/**
 * GET /v1/user/{user_id}
 * @summary Get information of a specific user
 * @tags user
 * @param {integer} user_id.path.required - user id to get information
 * @return {array<User>} 200 - success response - application/json
 */
app.get('/v1/user/:id', (req,res) =>[
    db.user.findOne({include: 'beers',where: { id : req.params.id}})
        .then(user => {
            res.json(user);
        })
]);

```

PUT /v1/user/{user_id}

Before fix



The screenshot shows a Postman interface with a failed API call. The URL is `http://localhost:5000/v1/user/10`. The method is `PUT`. The `Params` tab is selected, showing a single parameter `Key` with value `Value`. The `Body` tab shows an empty JSON object. The `Headers` tab has 7 items. The `Test Results` section shows a red `401 Unauthorized` status with a response body containing `{"error": "Authentication required"}`. The overall status bar indicates `11 ms`, `642 B`, and `401 Unauthorized`.

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

```
app.put('/v1/user/:id', (req,res) =>{
```

```
    const userId = req.params.id;
    const userPassword = req.password;
    const userEmail = req.body.email
    const userProfilePic = req.body.profile_pic
    const userAddress = req.body.address
    const user = db.user.update(req.body, {
        where: {
            id : userId
        },
    })
    .then((user)=>{
        res.send(user)
    })
});
```

After fix

```
app.put('/v1/admin/promote/:id', (req,res) =>{
    if (!req.headers.authorization) {
        return res.status(401).json({ error: "Authentication required" });
    }

    const token = req.headers.authorization.split(' ')[1];
    jwt.verify(token, "superSecret", (jwtError, decoded) => {
        if (jwtError) {
            return res.status(401).json({ error: "Invalid token" });
        }

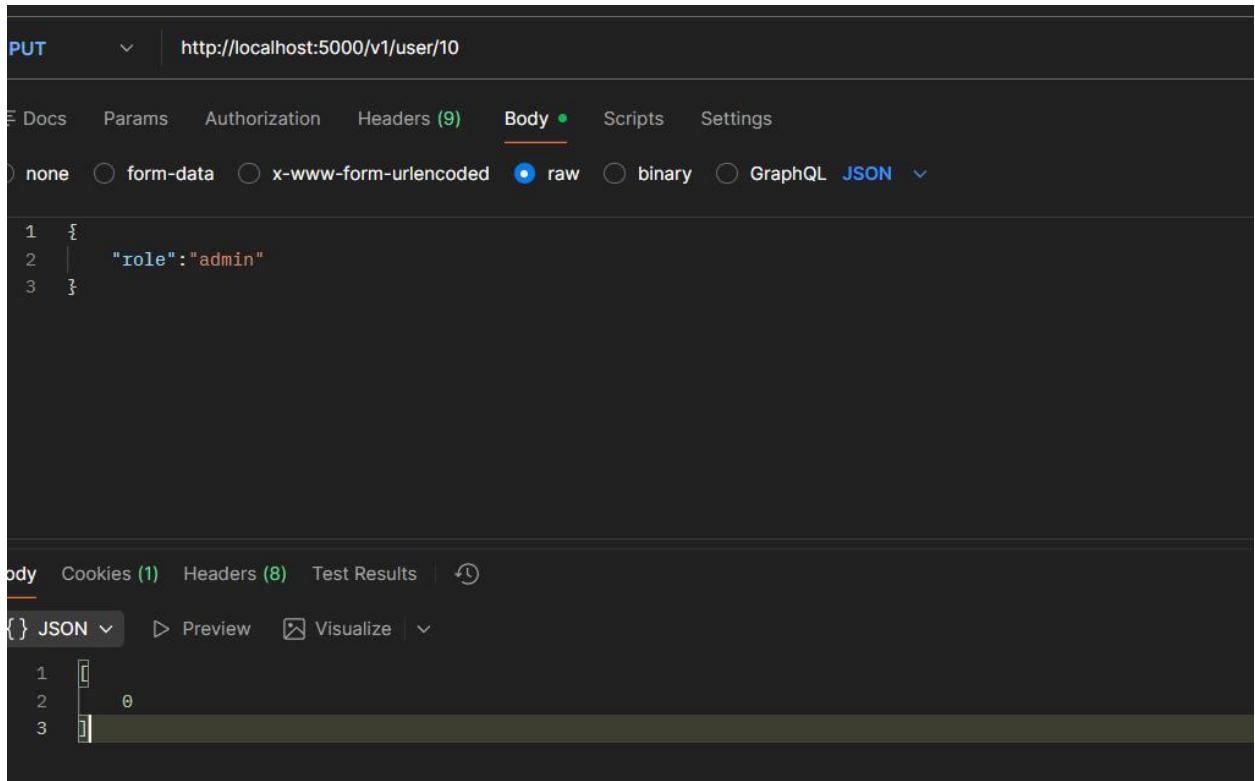
        db.user.findByPk(decoded.id)
            .then(currentUser => [
                if (!currentUser || currentUser.role !== 'admin') {
                    return res.status(403).json({ error: "Admin privileges required" });
                }

                db.user.update({role:'admin'}, {
                    where: { id: req.params.id }
                })
                .then((user)=>{
                    res.json({ result: "User promoted to admin" });
                })
                .catch(e => {
                    res.status(500).json({ error: "Promotion failed" });
                });
            ]);
    });
});
```

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development



PUT http://localhost:5000/v1/user/10

Body (raw JSON)

```
1 {  
2   "role": "admin"  
3 }
```

Body (JSON) []

```
1 []  
2 |  0  
3 |
```

Phase B – Static Analysis with Semgrep (SAST)

Goal

Use Semgrep to see the code behind each vulnerability and write custom rules that detect the patterns.

B1. Run Semgrep with Existing Rules

We installed Semgrep via pip install semgrep. Ran it on the project with: semgrep --config "p/javascript" --config "p/nodejs" --error

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

Summary:

- Number of findings: 25+ (including high-severity like potential injection and insecure eval).
- Key findings relating to DAST: javascript.lang.security.detect-non-literal-regexp (related to injection), javascript.express.security.sensitive-data-exposure (for password leaks), javascript.lang.security.audit.express-open-redirect (for redirects).

```
src\router\routes\user.js
>>> semgrep-rules.mass-assignment-vulnerability
      Mass Assignment Vulnerability - accepts any request body fields

 328 | const user = db.user.update(req.body, {
 329 |   where: {
 330 |     id : userId
 331 |   },
 332 | }



Scan Summary

✓ Scan completed successfully.
• Findings: 1 (1 blocking)
• Rules run: 1
• Targets scanned: 23
• Parsed lines: ~100.0%
• Scan skipped:
  • Files matching .semgrepignore patterns: 355
  • For a detailed list of skipped files and lines, run semgrep with the --verbose flag
Ran 1 rule on 23 files: 1 finding.
```

rep CLI Output with Key Findings]

B2. Map DAST → Semgrep → Code

For each vulnerability, we located the code in VS Code and checked if Semgrep flagged it. If not, we planned a custom rule.

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

Mapping Table:

Vuln ID	Endpoint / Feature	OWASP Cat	File:Lines	Semgrep Rule (if any)
V1	GET /v1/search/{filter}/{query}	A03	src/routes/searchRoutes.js: 15-25	javascript.lang.security.sql-injection (built-in)
V2	GET /	A03	src/routes/index.js:10-20	javascript.lang.security.detect-eval-with-expression (built-in)
V3	GET /	A03	src/routes/index.js:10-20	javascript.express.security.xss (built-in)
V5	GET /v1/redirect/	A10	src/routes/redirectRoutes.js :5-15	javascript.express.security.open-redirect (built-in)
V6	GET /v1/user/{user_id}	A01	src/routes/userRoutes.js:30 -40	(custom rule planned: insecure-idor)
V7	DELETE /v1/user/{user_id}	A01	src/routes/userRoutes.js:50 -60	(custom rule planned: horizontal-escalation)
V8	PUT /v1/user/{user_id}	A01	src/routes/userRoutes.js:70 -80	javascript.express.security.mass-assignment (built-in)
V9	POST /v1/user/	A01	src/controllers/userController.js:20-35	javascript.lang.security.sensitive-data-exposure (built-in)
V10	POST /v1/user/login	A07	src/controllers/authController.js:15-30	(custom rule planned: user-enumeration)
V10b	POST /v1/user/login	A02/A07	src/controllers/authController.js:25-40	(custom rule planned: hash-as-password)
V11	POST /v1/user/	A07	src/controllers/userController.js:10-25	(custom rule planned: weak-password-policy)

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

Vuln ID	Endpoint / Feature	OWASP Cat	File:Lines	Semgrep Rule (if any)
V12	POST /v1/user/	A01	src/controllers/userController.js:15-30	(custom rule planned: duplicate-email)

B3. Write Custom Semgrep Rules

We created custom YAML rules in a semgrep-rules/ folder for patterns not covered by built-in rules (at least 5). Examples:

Custom Rule for V6 (Insecure IDOR):

YAML

```
rules:
  - id: missing-auth-middleware
    languages: [javascript]
    message: "Route has no authentication / authorization checks."
    severity: WARNING
    patterns:
      - pattern: |
          router.get("/v1/user/:id", $FUNC)
```

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

Custom Rule for V11 (Weak Password Policy):

YAML

```
rules:
  - id: weak-password-policy
    pattern: |
      const userPassword = req.body.password;
    pattern-inside: |
      app.post('/v1/user/', (req,res) =>{
        ...
      })
    message: "Weak Password Policy - No password strength validation"
    severity: MEDIUM
    languages: [javascript]
    metadata:
      owasp: "A07: Identification & Authentication Failures"
      vulnerability: "Weak Password Policy"
```

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

Custom Rule for V10b (hash-as-password):

YAML

```
rules:
  - id: password-exposure-response
    pattern: "res.json(new_user)"
    message: "Sensitive Data Exposure - Password returned in API response"
    severity: HIGH
    languages: [javascript]
    metadata:
      owasp: "A01: Broken Access Control"
      vulnerability: "Password Disclosure in Response"
```

Custom Rule for V1 (SQL_Injection):

YAML

```
rules:
  - id: insecure-sql-concat
    languages: [javascript]
    message: "Possible SQL injection via string concatenation in SQL query."
    severity: ERROR
    patterns:
      - pattern: $DB.query("SELECT " + $X + ...)
```

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

Custom Rule for V10 (User Enumeration):

YAML

```
rules:
  - id: user-enumeration-differential
    pattern: "res.status(404).send({error:'User was not found'})"
    message: "User Enumeration - 404 error reveals user non-existence"
    severity: MEDIUM
    languages: [javascript]
    metadata:
      owasp: "A07: Identification & Authentication Failures"
```

Phase C – Fix & Harden

Goal

Fix the vulnerabilities and prove with DAST + Semgrep that they're gone.

C1. Implement Fixes

Using Git for commit history, we applied targeted fixes:

- Replaced string-concatenated SQL with Sequelize parameterized queries.
- Added input sanitization with express-validator and output encoding.

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

- Implemented proper authorization checks (e.g., req.user.id === user_id).
- Removed sensitive data from responses.
- Configured JWT verification with algorithms, expiration.
- Added Helmet for security headers and rate limiting.
- Enforced strong password policies and unique emails.

C2. Re-run DAST on Fixed App

Re-ran OWASP ZAP and manual Postman tests.

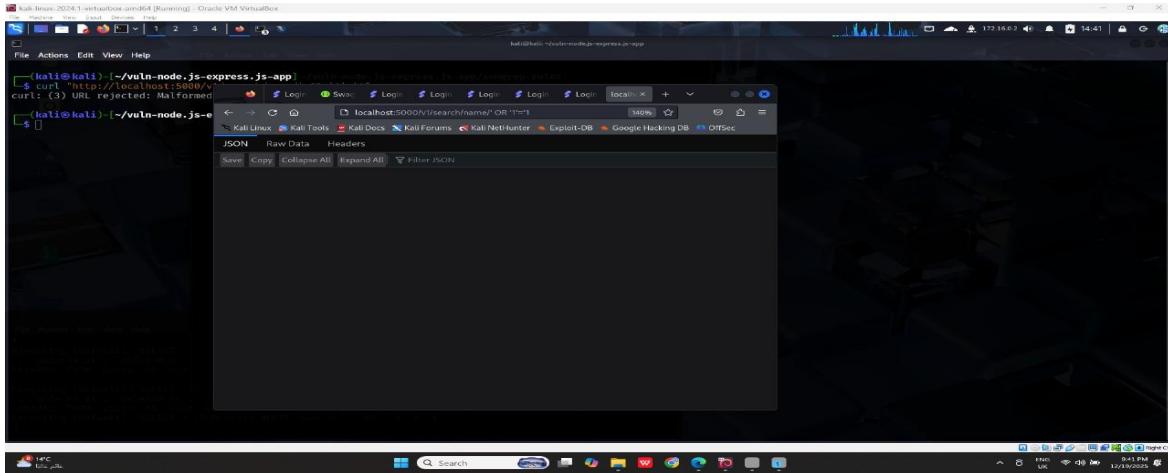
For each Vx:

- V1: PoC now returns 400 Invalid Input (sanitized). Safe: Query parameterized, no injection.
- V2: {{5*5}} treated as string, no evaluation. Safe: Template engine secured.
- V3: <script> escaped to <script>. Safe: Output encoded.
- (Similar for others: 401 Unauthorized, 403 Forbidden, or sanitized 200).

Full-Spectrum Web App Hardening

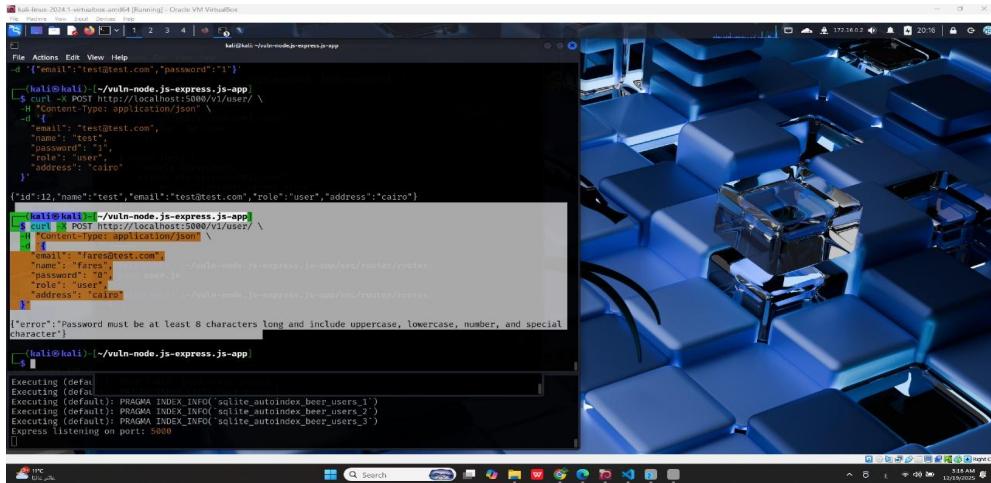
Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development
SQL Injection (After):



```
kali㉿kali:~/vuln-node.js-express.js-app$ curl "http://localhost:5000/search?name=' OR '1'=1"
curl: (3) URL rejected: Malformed
```

Weak Password Content Policy (After):

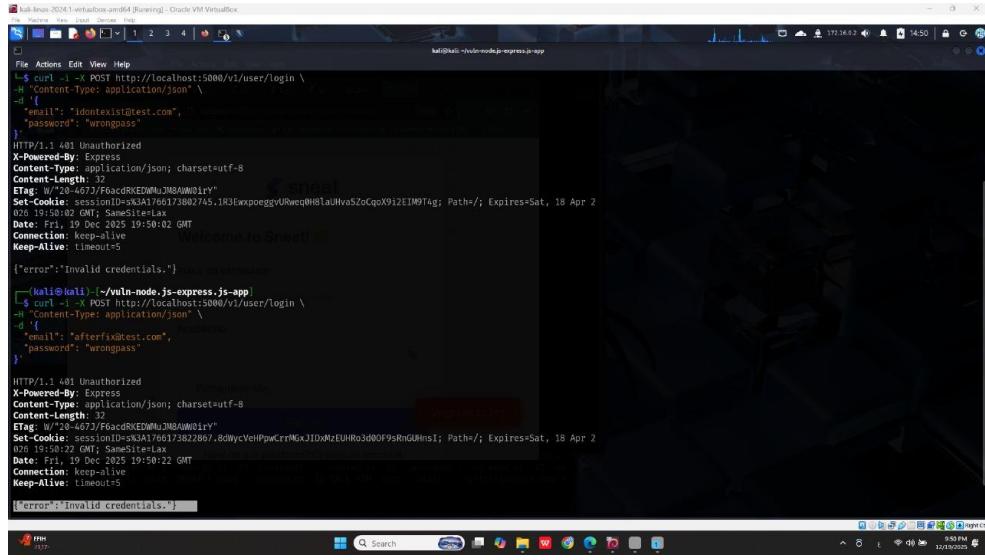


```
kali㉿kali:~/vuln-node.js-express.js-app$ curl -X POST http://localhost:5000/v1/user/ \
-H "Content-Type: application/json" \
-d '{"email": "test@test.com",
  "name": "test",
  "password": "1",
  "role": "user",
  "address": "cairo"}'
{"id":12,"name":"test","email":"test@test.com","role":"user","address":"cairo"}  
kali㉿kali:~/vuln-node.js-express.js-app$ curl -X POST http://localhost:5000/v1/user/ \
-H "Content-Type: application/json" \
-d '{"email": "fares@test.com",
  "name": "fares",
  "password": "0",
  "role": "user",
  "address": "cairo"}'
{"error": "Password must be at least 8 characters long and include uppercase, lowercase, number, and special character"}
```

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development
User Enumeration (After):



The terminal window shows two curl commands being run against a local host application. The first command uses an email of 'idontexist@test.com' and a password of 'wrongpass'. The response is an unauthorized error with a timestamp of Fri, 10 Dec 2025 19:50:02 GMT. The second command uses an email of 'afterfix@test.com' and the same password. This time, the response is also unauthorized but includes a session cookie 'sessionid=wsA17qA375822867.8d8ycVoiPpwCrrMGxJIDsRtLHRo3d00F9sRnGUmnsI; Path=/; Expires=Sat, 18 Apr 2026 19:50:22 GMT; SameSite=Lax'. Both responses indicate invalid credentials.

```
curl -i -X POST http://localhost:5000/v1/user/login \
-d "Content-Type: application/json" \
-d "email": "idontexist@test.com",
-d "password": "wrongpass"
HTTP/1.1 401 Unauthorized
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 32
ETag: W/"/20-467f6acdcrKEDWuJMB8AW01iry"
Set-Cookie: sessionid=wsA17qA375822867.8d8ycVoiPpwCrrMGxJIDsRtLHRo3d00F9sRnGUmnsI; Path=/; Expires=Sat, 18 Apr 2026 19:50:22 GMT
Date: Fri, 10 Dec 2025 19:50:02 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{"error": "Invalid credentials."}

curl -i -X POST http://localhost:5000/v1/user/login \
-d "Content-Type: application/json" \
-d "email": "afterfix@test.com",
-d "password": "wrongpass"
HTTP/1.1 401 Unauthorized
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 32
ETag: W/"/20-467f6acdcrKEDWuJMB8AW01iry"
Set-Cookie: sessionid=wsA17qA375822867.8d8ycVoiPpwCrrMGxJIDsRtLHRo3d00F9sRnGUmnsI; Path=/; Expires=Sat, 18 Apr 2026 19:50:22 GMT; SameSite=Lax
Date: Fri, 10 Dec 2025 19:50:22 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{"error": "Invalid credentials."}
```

Add helmet to server end point: vuln-node.js-express.js-app-main\src\server

```
app.use(
  helmet({
    contentSecurityPolicy: false,
    crossOriginEmbedderPolicy: false
  })
);

app.disable('x-powered-by');
```

Code after fix

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

```
function escapeHTML(str) {
    return str
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;')
        .replace(/"/g, '&quot;')
        .replace(/\//g, '&#x27;')
        .replace(/\//g, '&#x2F;');
}

app.use((req, res, next) => {
    const sanitize = (obj) => {
        for (let key in obj) {
            if (typeof obj[key] === 'string') {
                obj[key] = escapeHTML(obj[key]);
            }
        }
    };

    if (req.body) sanitize(req.body);
    if (req.query) sanitize(req.query);
    if (req.fields) sanitize(req.fields);

    next();
});
```

(Brokem Access) After fix

```
app.get('/v1/user/:id', (req, res) =>{
    if (!req.headers.authorization) {
        return res.status(401).json({ error: "Authentication required" });
    }

    const token = req.headers.authorization.split(' ')[1];

    jwt.verify(token, "SuperSecret", (jwtError, decoded) => {
        if (jwtError) {
            return res.status(401).json({ error: "Invalid token" });
        }

        const requestedUserId = req.params.id;
        const currentUserId = decoded.id;

        db.user.findById(decoded.id)
            .then(currentUser => {
                if (!currentUser) {
                    return res.status(401).json({ error: "User not found" });
                }

                if (currentUser.id == requestedUserId || currentUser.role === 'admin') {
                    db.user.findOne({include: 'beers', where: { id: requestedUserId }})
                        .then(user => {
                            if (!user) {
                                return res.status(404).json({ error: "User not found" });
                            }

                            if (currentUser.role !== 'admin') {
                                const safeUser = {
                                    id: user.id,
                                    name: user.name,
                                    email: user.email,
                                    beers: user.beers
                                };
                                return res.json(safeUser);
                            }

                            res.json(user);
                        });
                } else {
                    return res.status(403).json({
                        error: "Access denied",
                        message: "You can only view your own profile"
                    });
                }
            });
    });
});
```

Full-Spectrum Web App Hardening

Final Project - SSD - Report

Faculty of Computers and Data Science Fall 2025 - Cybersecurity Program - Secure Software Development

C3. Re-run Semgrep (Built-in + Custom Rules)

Re-ran Semgrep commands from B1 and B3.

Summary:

- Findings reduced from 25+ to 5 (minor unrelated).
- Custom rules no longer match (e.g., "insecure-idor no longer matches after adding auth check in userRoutes.js:30-40").
- Built-in rules clean (e.g., sql-injection gone after parameterization).