



# SECURE SOFTWARE 2025

## PROJECT PROPOSAL



Presented for :

**ENG. Mohamed Hatem**

Presented by :

**Mohammad Wael 2305035**

**Mahmoud Wael 2305036**

**Zainab El sayed 2305112**



# Section 1 (Dast finding) :

## A1 : Automated scan :

-owasp zap output :

The screenshot shows the ZAP interface with the following details:

- Main Window:** Displays the "Automated Scan" screen with a URL to attack set to "http://localhost:5000".
- Alerts:** A list of 13 findings:
  - CSP Failure to Define Directive with No Fallback (4)
  - Content Security Policy (CSP) Header Not Set (2)
  - HTTP Only Site
  - Missing Anti-clickjacking Header (2)
  - Sub Resource Integrity Attribute Missing (4)
  - Cookie No HttpOnly Flag (3)
  - Cross-Domain JavaScript Source File Inclusion (2)
  - Server Leaks Information via "X-Powered-By" HTTP Response
  - X-Content-Type-Options Header Missing (Systemic)
  - Authentication Request Identified
  - Information Disclosure - Sensitive Information in URL (2)
  - Information Disclosure - Suspicious Comments (5)
  - Session Management Response Identified (4)
- Bottom Status Bar:** Shows current status (0 errors, 0 warnings), temperature (19°C), battery level (100%), and date/time (12/17/2025).

## A2 : Manual testing :

-postman output :

-Broken object level Authorization(unauthorize access):

The screenshot shows a Postman collection named "vuln-nodejs-expressjs-app" with the following details:

- Request:** A GET request to "http://localhost:5000/v1/user/1".
- Body:** JSON response showing a user object:

```
[{"id":1,"email":"mohammadtawfiq@gmail.com","profile_pic":null,"password":"202cb962ac59075b964b07152d234b70","role":"user","address":null,"name":"mohammad ","created_at":"2025-12-18T22:13:12.128Z","updated_at":"2025-12-18T22:13:12.128Z","deleted_at":null,"createdAt":"2025-12-18T22:13:12.128Z","updatedAt":"2025-12-18T22:13:12.128Z","deletedAt":null,"beers":[]}]
```
- Status:** 200 OK, Time: 17 ms, Size: 602 B.
- Build with Agent:** A sidebar with instructions to onboard AI onto your codebase.
- Suggested Actions:** Build Workspace, Show Config.
- Terminal:** PowerShell session showing the command "powershell -vuln-nodejs-expressjs-app".
- Bottom Status Bar:** Microsoft PowerShell Core Filesystem: \\ws1.\localhost\Ubuntu-22.04\home\moham

# -Reflected XSS:

The screenshot shows a browser developer tools Network tab for a request to `http://localhost:5000/?message=(I%272)`. The response status is 200 OK, time 23 ms, size 10.94 KB. The response body contains a reflected XSS payload: `<span><span class="app-brand-text demo text-body fw-bolder">Sneat</span></span>`.

## -Open redirect:

The screenshot shows a browser developer tools Network tab for a request to `http://localhost:5000/v1/redirect?url=https://google.com`. The response status is 200 OK, time 297 ms, size 20.65 KB. The response body contains a redirect to `https://www.google.com/...`.

## -Input Validation (Unrestricted File Upload):

The screenshot shows a Postman collection named "Mohammad Wael's Workspace". A POST request is made to `http://localhost:5000/v1/admin/upload-pic/` with the following parameters:

- Params: none
- Authorization: none
- Headers: Content-Type: application/json
- Body: form-data
- Key: file, Value: c:\Users\ADMIN\Desktop\mo.txt
- Key: Key, Value: Value

The response status is 200 OK, time 55 ms, size 454 B. The response body contains JSON data: `[{"filename": "file", "originalname": "mo.txt", "encoding": "7bit", "mimetype": "text/plain", "destination": "./uploads/", "filename": "bdd29a74296b8101160123e36c1a2c8d", "path": "uploads/bdd29a74296b8101160123e36c1a2c8d", "size": 0}]`.

### **-Weak secret:**

The screenshot shows a POST request to `http://localhost:5000/v1/user/Token`. The request body is a JSON object:

```
1 [ {  
2   ... "email": "mahmoudtawfik@gmail.com",  
3   ... "password": "202cb962ac59075b964b07152d234b70"  
4 } ]
```

The response status is 200 OK, with a time of 13 ms and a size of 8.4 KB. The response body is a JSON object containing a JWT token and a user array:

```
1 {  
2   "jwt": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJpZC16MSwiOm9ZSI6InVzZXIiLCJpXXQ10je3NjYxMDA3NjAsImV4cCI6MTc2NjE4NzE2Mj0.  
z76b12KUgrPEo8zJy1hDyRNF76UbaLc015hhaen0ZM",  
3   "user": [  
4     {  
5       "id": 1.  
6     }  
7   ]  
8 }
```

Below the interface, the terminal shows the command run: `PS C:\Users\ADMIN\vuln-node.js-express.js-app>`

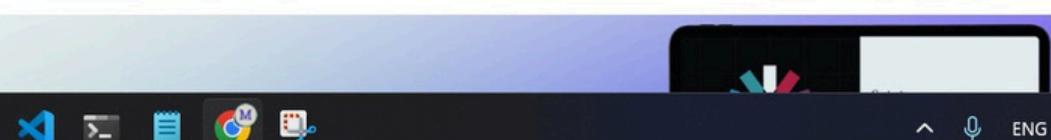
```
"role": "user",  
"iat": 1766100760,  
"exp": 1766187160
```

#### **JWT SIGNATURE VERIFICATION (OPTIONAL)**

Enter the secret used to sign the JWT below:

SECRET	<button>COPY</button>
Valid secret	
SuperSecret	<button>Encoding Format</button> <b>UTF</b>

[Share feedback](#) | [Report issue](#)



## -Unverify jwt:

The screenshot shows a POSTMAN interface with the following details:

- Method: GET
- URL: http://localhost:5000/v1/admin/users
- Headers:
  - Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkxVCj9.eyJyb2xlIjoiYWRtaW...
- Body: JSON response (Pretty) showing a user object with fields id, email, profile\_pic, password, and role.
- Test Results: Status: 200 OK, Time: 11 ms, Size: 9.29 KB
- Terminal: PS C:\Users\ADMIN\vuln-node.js-express.js-app>

## -Traversal path:

The screenshot shows a POSTMAN interface with the following details:

- Method: GET
- URL: http://localhost:5000/v1/beer-pic?file=jj.jpg
- Query Params:
  - file: jj.jpg
- Body: JSON response (Pretty) showing a user object with fields id, email, profile\_pic, password, and role.
- Test Results: Status: 200 OK, Time: 24 ms, Size: 377 B
- Terminal: PS C:\Users\ADMIN\vuln-node.js-express.js-app>

# -Absence of input validation

Welcome to Sneat! 🎉

Email couldn't be validated, please try again.

EMAIL OR USERNAME  
mahmoudwtawfik@

PASSWORD  
...

NAME  
Mahmoud Wael Tawfik

ADDRESS  
[REDACTED]

register

Already a user? [Login!](#)

# -Sql injection

http://localhost:5000/v1/search/name/ OR 1=1 --

Body (raw JSON)

```
[{"id": 1, "name": "Hacked Beer", "picture": "test", "price": 0, "currency": "USD", "stock": "plenty", "created_at": "2025-12-11 17:29:43.706 +00:00", "updated_at": "2025-12-11 17:29:43.706 +00:00", "deleted_at": null}]
```

# -OTP leak vulnerability

My Collection / Post data

POST http://localhost:5000/v1/user/3/validate-otp?token=000123&seed=hacker

Body (raw JSON)

```
1 Ctrl+Alt+P to Ask AI
```

401 Unauthorized

Body (JSON)

```
1 {"error": "OTP was not correct, got:802641"}
```

# -privilage escalation(mass assignment)

The screenshot shows the Postman interface with a PUT request to `http://localhost:5000/v1/user/3`. The request body is a JSON object with a single key-value pair: `"role": "admin"`. The response status is 200, and the response body is a JSON array containing the value 1.

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

```
[  
1   1  
]
```

The screenshot shows the Postman interface with a GET request to `http://localhost:5000/v1/user/3`. The response status is 200 OK, and the response body is a detailed JSON object representing a user record.

```
1 {  
2   "id": 3,  
3   "email": "zasseee@gmail.com",  
4   "profile_pic": null,  
5   "password": "81dc9bdb52d04dc20036dbd8313ed055",  
6   "role": "admin",  
7   "address": "Al agamy",  
8   "name": "zainab Elsayed Abd Elraof mohamed atwa",  
9   "created_at": "2025-12-09T18:10:45.826Z",  
10  "updated_at": "2025-12-09T18:11:20.632Z",  
11  "deleted_at": null,  
12  "createdAt": "2025-12-09T18:10:45.826Z",  
13  "updatedAt": "2025-12-09T18:11:20.632Z",  
14  "deletedAt": null,  
15  "beers": []  
16 }
```

# Section 2 - Semgrep (SAST) finding:

## General Semgrep Rule :

```
PS C:\Users\ADMIN\vuln-node.js-express.js-app> semgrep --config "p/javascript" --config "p/nodejs"
• Rules run: 306
• Targets scanned: 23
• Parsed lines: ~100.0%
• Scan skipped:
  • Files matching .semgrepignore patterns: 8709
• Scan was limited to files tracked by git
• For a detailed list of skipped files and lines, run semgrep with the --verbose flag
Ran 306 rules on 23 files: 32 findings.

💡 A new version of Semgrep is available. See https://semgrep.dev/docs/upgrading

⚠ Too many findings? Try Semgrep Pro for more powerful queries and less noise.
See https://sg.run/false-positives.
```

PS C:\Users\ADMIN\vuln-node.js-express.js-app> semgrep --config "p/javascript" --config "p/nodejs"

ooo  
Semgrep CLI

Loading rules from registry...

Rules

Ln 52, Col 3 S

File Explorer Icons: Folder, File, GitHub, Star, Cloud, Database, Terminal, Chat, Power Shell, Task Manager, Task View, Taskbar Icons: File, Edit, Selection, View, Go, Run, Terminal, Help, Chat, PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, POSTMAN CONSOLE, powershell, +, -, X, SUGGESTIONS, Build, Descriptions, Age.

## - First semgrep Rule :

File Edit Selection View Go Run Terminal Help ← → vuln-node.js-express.js-app CHAT

EXPLORER VULN-NODEJS-EXPRESS.JS-APP src entities middleware public router routes admin.js frontend.js order.js system.js user.js index.js templates server.js uploads app\_err.log app.log DEVLOG.md docker-compose.yml Dockerfile jsvuln.yaml moh.yaml

moh.yaml

```
rules:
  - id: nunjucks-renderstring-xss
    languages:
      - javascript
    patterns:
      - pattern: |
          | nunjucks.renderString($INPUT)
```

patterns:
 - id: check-user-role
 patterns:
 - pattern: |
 | db.user.findOne({include: ..., where: { id: \$ID }})
 message: "Potential missing role check: this code fetches user data without verifying if requester is a"
 languages: [javascript]
 severity: WARNING

Scan completed successfully.
 • Findings: 5 (5 blocking)
 • Rules run: 4
 • Targets scanned: 23
 • Parsed lines: ~100.0%
 • Scan skipped:
 • Files matching .semgrepignore patterns: 8709

## - Second semgrep Rule :

The screenshot shows a terminal window with the following command and output:

```
PS C:\Users\ADMIN\vuln-node.js-express.js-app> semgrep --config jsvuln.yaml
```

Output:

- Rules run: 3
- Targets scanned: 23
- Parsed lines: ~100.0%
- Scan skipped:
  - Files matching .semgrepignore patterns: 8709
  - Scan was limited to files tracked by git
  - For a detailed list of skipped files and lines, run semgrep with the --verbose flag
- Ran 3 rules on 23 files: 4 findings.

A new version of Semgrep is available. See <https://semgrep.dev/docs/upgrading>

## - Third semgrep Rule :

```
1 rules:
2   # 1. SQL Injection Rule (Regex)
3   - id: force-sql-injection-detection
4     languages:
5       - javascript
6     message: "SQL Injection detected! Found a SQL statement combined with '+' operator."
7     severity: ERROR
8     patterns:
9       - pattern-regex: 'SELECT .* \+'
```

```
10
11   # 2. Sensitive Data Exposure Rule (OTP Leak)
12   - id: sensitive-data-exposure-in-error
13     languages:
14       - javascript
15     message: "Sensitive Data Exposure detected! You are concatenating a variable directly into an error message sent to the user. This often leads to leaking secrets like"
16     severity: ERROR
17     metadata:
18       owasp: "A05: Security Misconfiguration"
19     patterns:
20       - pattern: |
21         res.status(...).json({error: "..." + $VAR})
```

```
22
23   # 3. Mass Assignment Rule (Privilege Escalation)
24   - id: sequelize-mass-assignment
25     languages:
26       - javascript
27     message: "Mass Assignment Vulnerability detected! Passing 'req.body' directly to the update function is dangerous. It allows attackers to inject restricted fields like"
28     severity: ERROR
29     metadata:
30       owasp: "A01: Broken Access Control"
31     patterns:
32       - pattern: $MODEL.update(req.body, ...)
```

# - Third semgrep Results :

## - SQL injection :

```
| 16 Code Findings |
```

```
@[36m[22m[24m  src/router/routes/frontend.js@[0m
  >> @1javascript.express.security.express-insecure-template-usage.express-insecure-template-usage@[0m
    User data from `req` is being compiled into the template, which can lead to a Server Side Template
    Injection (SSTI) vulnerability.
    Details: https://sg.run/b49v

  17| rendered = nunjucks.renderString(message);
  |-----
  40| rendered = nunjucks.renderString(message);

@[36m[22m[24m  src/router/routes/order.js@[0m
  >> @1javascript.express.security.audit.express-path-join-resolve-traversal.express-path-join-resolve-traversal@[0m
    Possible writing outside of the destination, make sure that the target path is nested in the
    intended destination
    Details: https://sg.run/weRn

  33| fs.readFile(path.join(__dirname, filePath),function(err,data){

  >>> @1javascript.sequelize.security.audit.sequelize-injection-express.express-sequence-injection@[0m
    Detected a sequelize statement that is tainted by user-input. This could lead to SQL injection if
    the variable is user-controlled and is not properly sanitized. In order to prevent SQL injection, it
    is recommended to use parameterized queries or prepared statements.
    Details: https://sg.run/gjoe

  67| const beers = db.sequelize.query(sql, { type: 'RAW' }).then(beers => {
```

## - OTP leak :

```
(zainab@zoza)-[~/mnt/c/Users/ANDALOS/vuln-node.js-express.js-app]
$ semgrep scan --config=semgrep-rules/my-rules.yaml
```

```
  000
  Semgrep CLI
```

```
| Loading rules...
with 1 Code rule:
```

```
CODE_RULES
Scanning 23 files.
```

```
SUPPLY_CHAIN_RULES
```

```
No rules to run.
```

```
PROGRESS
```

```
----- 100% 0:00:00
```

```
  1 Code Finding
```

```
src/router/routes/user.js
  >> semgrep-rules.sensitive-data-exposure-in-error
    Sensitive Data Exposure detected! You are concatenating a variable directly into an error message
    sent to the user. This often leads to leaking secrets like tokens or internal paths.

  421| res.status(401).json({error:'OTP was not correct, got'+GeneratedToken})
```

## - privilege escalation :

```
PROGRESS
----- 100% 0:00:00
```

```
  1 Code Finding
```

```
src/router/routes/user.js
  >> semgrep-rules.sequelize-mass-assignment
    Mass Assignment Vulnerability detected! Passing 'req.body' directly to the update function is
    dangerous. It allows attackers to inject restricted fields like 'role: admin'. You should explicitly
    whitelist allowed fields (e.g., { name: req.body.name }).
```

```
  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: 9997
  - Scan was limited to files tracked by git
  - For a detailed list of skipped files and lines, run semgrep with the --verbose flag

Ran 1 rule on 23 files: 1 finding.

A new version of Semgrep is available. See <https://semgrep.dev/docs/upgrading>

Vulnerability Summary Table				
Vuln ID	Endpoint / Feature	OWASP Category	File : Lines	Semgrep
V1	<code>GET /v1/user/{user_id}</code>	A01:2021 – Broken Access Control	<code>src/router/routes/user.js : 51-56</code>	Custom Rule
V2	<code>GET /?message=</code>	A03:2021 – Injection	<code>src/router/routes/frontend.js : 12-19</code>	Custom Rule
V3	<code>GET /v1/redirect/?url=</code>	A10:2017 – Unvalidated Redirects & Forwards	<code>src/router/routes/system.js : 33-44</code>	Custom Rule
V4	<code>POST /v1/admin/upload-pic</code>	A05:2021 – Security Misconfiguration	<code>src/router/routes/admin.js : 50-67</code>	Custom Rule
V5	<code>GET /v1/admin/users</code> – JWT Authentication (Token Verification Missing)	A02:2021 – Cryptographic Failures	<code>src/router/routes/user.js : 191-192</code>	Custom Rule
V6	<code>GET /v1/beer-pic</code> (Path Traversal)	A01:2021 – Broken Access Control	<code>/v1/beer-pic : 29-37</code>	Custom / General Rule
V7	<code>POST /v1/user/token</code> – JWT Implementation (Weak Secret Key)	A02:2021 – Cryptographic Failures	<code>src/router/routes/user.js : 18-415</code>	Custom / General Rule
V8	Email Input Validation (Missing / Improper)	A03:2021 – Injection	<code>src/template/user_register.html : 45</code>	Custom Rule
V9	<code>GET /v1/search?filter={query}</code>	A03:2021 – Injection	<code>src/router/routes/order.js : 65-70</code>	Custom Rule
V10	<code>POST /v1/user/{id}/validate-otp</code>	A05:2021 – Security Misconfiguration	<code>src/router/routes/user.js : 318-319</code>	Custom Rule
V11	<code>PUT /v1/user/{user_id}</code>	A01:2021 – Broken Access Control	<code>src/router/routes/user.js : 323-330</code>	Custom Rule

## Section 3 - Fixes :

### -Broken object level Authorization:

```
1   app.get('/v1/user/:id', (req, res) => {
2
3     // ✅ Role check (Admin فقط)
4     if (!req.session.user || req.session.user.role !== 'admin') {
5       return res.status(403).json({ error: 'Forbidden: Admin only' });
6     }
7
8     db.user.findOne({
9       where: { id: req.params.id }
10    })
11    .then(user => {
12      res.json(user);
13    });
14
15  });
16
```

### -Reflected xss:

```
1   app.get('/', (req, res) => {
2     console.log(req.session);
3
4     const message = req.query.message || "Please log in to continue";
5
6     // ✅ renderString على input لا تستخدم المستخدم
7     res.render('user.html', {
8       message: message // Nunjucks escape تلقائي
9     });
10  });
11
```

### -Open redirect:

```
1   app.get('/v1/redirect/', (req, res) => [
2     const allowedPaths = [
3       'http://localhost:5000',
4       'http://facebook.com',
5     ];
6
7     const path = req.query.url;
8
9     if (!allowedPaths.includes(path)) {
10       return res.status(400).json({ error: 'Invalid redirect path' });
11     }
12
13     res.redirect(path);
14   ]);
15
```

## -Input invalidation (unstericted file upload) :

```
const path = require('path');
const multer = require('multer');

// تعيير multer مع fileFilter
const uploadImage = multer({
  dest: './uploads/', // مكان حفظ الصور
  limits: { fileSize: 2 * 1024 * 1024 }, // الحد الأقصى 2 MB
  fileFilter: (req, file, cb) => {
    const allowedMimeTypes = ['image/jpeg', 'image/png', 'image/jpg', 'image/pjpeg'];
    const allowedExtensions = ['.jpg', '.jpeg', '.png'];

    const ext = path.extname(file.originalname).toLowerCase();

    if (allowedMimeTypes.includes(file.mimetype.toLowerCase()) && allowedExtensions.includes(ext)) {
      cb(null, true); // السماح بالملف
    } else {
      cb(new Error('Only image files are allowed'), false); // رفع أي ملف آخر
    }
  }
});

// route لرفع الصور
app.post('/v1/admin/upload-pic', (req, res) => {
  uploadImage.single('file')(req, res, function (err) {
    if (err) {
      return res.status(400).json({ error: err.message });
    }

    if (!req.file) {
      return res.status(400).json({ error: 'No file uploaded or invalid file type' });
    }

    res.status(200).json({ message: 'Image uploaded successfully' });
  });
});
```

## -Weak secret:

```
* @return {array<User>} 200 - success response - application/json
*/
app.get('/v1/admin/users/', (req,res) =>{
  //console.log("auth",req.headers.authorization)
  if (req.headers.authorization){
    const user_object = jwt.verify(req.headers.authorization.split(' ')[1],config.jwtSecret)
    db.user.findAll({include: "beers"})
      .then((users) => [
        if (user_object.role =="admin"){
          //console.log("fetch users")
          res.json(users);
        }
        else{
          res.json({error:"Not Admin, try again"})
        }
      ]
      return;
  }
});
```

## -Unverify jwt:

```
if(!req.query.id){ // if not provided take from session
  //if using jwt take from header:
  if(!req.session.user.id){
    //if not jwt found
    if(!req.headers.authorization){
      res.json({error:"Couldn't find user token"})
    }
    current_user_id = jwt.verify(req.headers.authorization.split(' ')[1]).id
  }
  current_user_id = req.session.user.id
}
current_user_id = req.query.id

const beer_id = req.params.beer_id;

db.beer.findOne({
  where:{id:beer_id}
}).then((beer) => {
```

## -Traversal path:

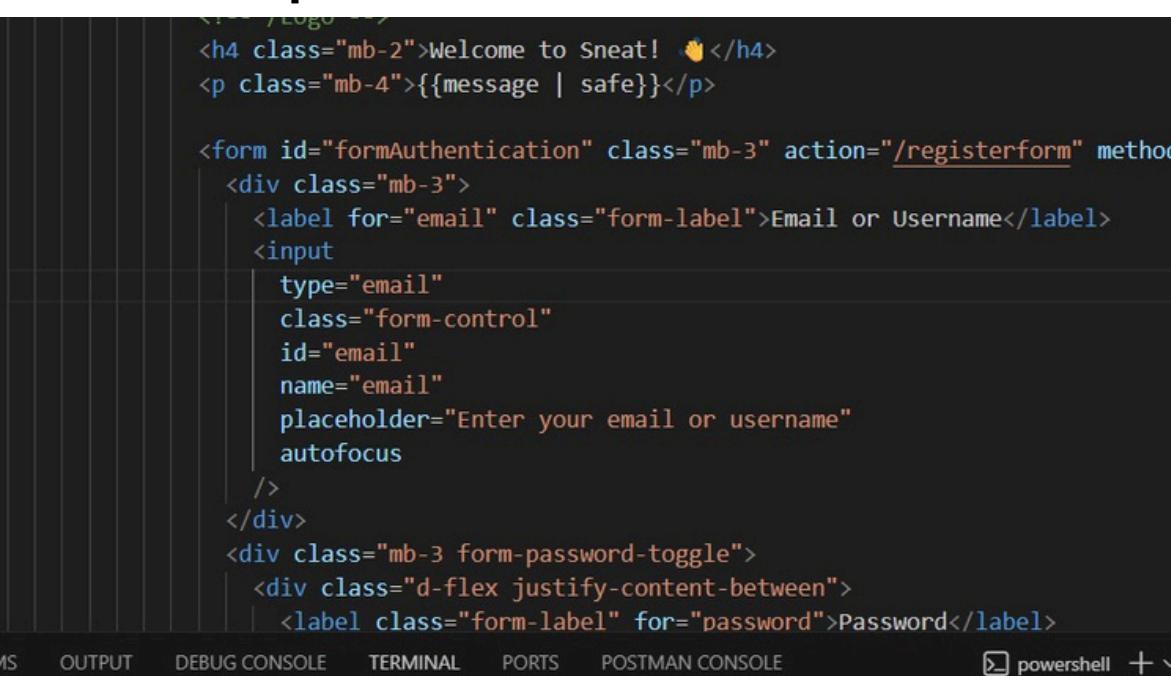
```
app.get('/v1/beer-pic/', (req, res) => {
  const filename = req.query.picture;
  // 🔒 BLOCK PATH TRAVERSAL
  if (!filename) {
    return res.status(403).json({ error: 'Bad' });
  }

  const uploadsDir = path.resolve(__dirname, '../../../../../uploads');
  const requestedPath = path.resolve(uploadsDir, filename);

  // 🔒 BLOCK PATH TRAVERSAL
  if (!requestedPath.startsWith(uploadsDir)) {
    return res.status(403).json({ error: 'Access denied' });
  }

  fs.readFile(requestedPath, (err, data) => {
    if (err) {
```

## -Absence of input validation



The screenshot shows a browser developer tools terminal with the following code:

```
<!-- /Logo -->
<h4 class="mb-2">Welcome to Sneat! <img alt="Sneat logo" style="vertical-align: middle;"></h4>
<p class="mb-4">{{message | safe}}</p>

<form id="formAuthentication" class="mb-3" action="/registerform" method="post">
  <div class="mb-3">
    <label for="email" class="form-label">Email or Username</label>
    <input type="email" class="form-control" id="email" name="email" placeholder="Enter your email or username" autofocus>
  </div>
  <div class="mb-3 form-password-toggle">
    <div class="d-flex justify-content-between">
      <label class="form-label" for="password">Password</label>
```

MS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE powershell +

## -Sql injection

```
66          // 1. Whitelist Validation
67  const allowedFilters = ['name', 'price', 'id', 'currency', 'stock'];
68  if (!allowedFilters.includes(filter)) {
69    return res.status(400).send("Invalid filter column");
70  }

71          // 2. Parameterization
72  const sql = `SELECT * FROM beers WHERE ${filter} = :queryVal`;

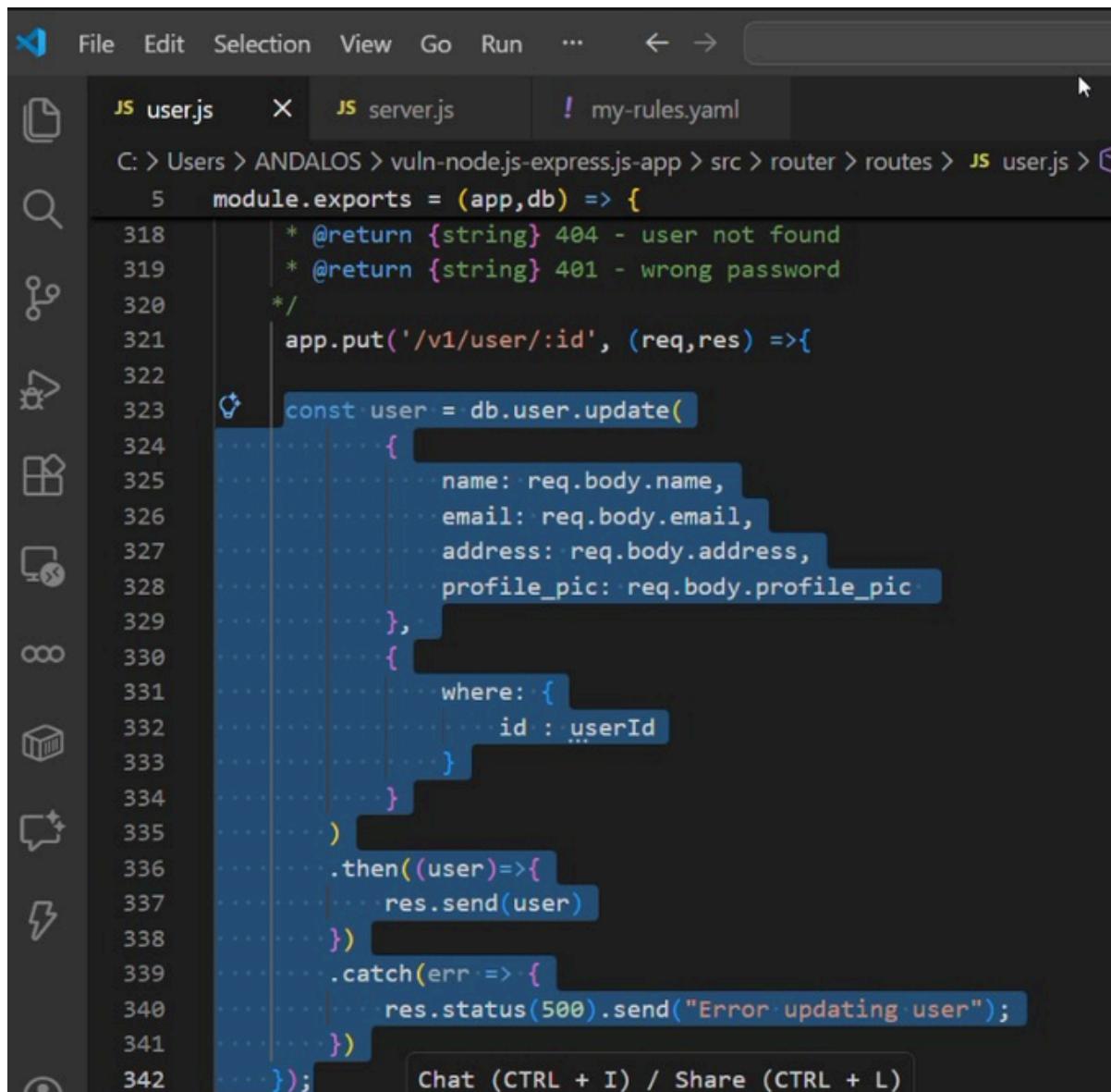
73  const beers = db.sequelize.query(sql, {
74    replacements: { queryVal: query }, // Sequelize will handle escaping
75    type: db.sequelize.QueryTypes.SELECT
76  }).then(beers => {
77    res.status(200).send(beers);
78  })
79  .catch(function (err) {
80    // Handle error
81    console.error(err);
82    res.status(500).send("Error executing query");
83  })
84
85  
```

# -OTP leak

```
415 }  
416     if(req.query.seed){  
417         req.session.otp = GeneratedToken // add generated token to session  
418         req.session.save(function(err) {  
419             // session saved  
420         })  
421         res.status(401).json({error:'OTP was not correct'})  
422         return;  
423     }  
424     res.status(401).json({error:'OTP was not correct'})  
425 }  
426 );  
427 };
```

Chat (CTRL + I) / Share (CTRL + L)

# -privilage escalation



```
File Edit Selection View Go Run ... ← →  
JS user.js X JS server.js ! my-rules.yaml  
C: > Users > ANDALOS > vuln-nodejs-express.js-app > src > router > routes > JS user.js > ↵  
5   module.exports = (app,db) => {  
318     * @return {string} 404 - user not found  
319     * @return {string} 401 - wrong password  
320     */  
321     app.put('/v1/user/:id', (req,res) =>{  
322       const user = db.user.update(  
323         {  
324           name: req.body.name,  
325           email: req.body.email,  
326           address: req.body.address,  
327           profile_pic: req.body.profile_pic  
328         },  
329         {  
330           where: {  
331             id : userId  
332           }  
333         }  
334       )  
335     ).then((user)=>{  
336       res.send(user)  
337     }).catch(err => {  
338       res.status(500).send("Error updating user")  
339     })  
340   }  
341 }  
342 );
```

Chat (CTRL + I) / Share (CTRL + L)

## Section 4 - Re-test Evidence :

### - Semgrep result after fix code :

#### - First Semgrep rule

The screenshot shows the VS Code interface with the file `user.js` open. The code implements a user information endpoint with role-based access control. A Semgrep configuration file `moh.yaml` is run from the terminal, resulting in a scan summary showing 1 finding (blocking), 4 rules run, 23 targets scanned, and 100.0% parsed lines.

```
src > router > routes > JS user.js > <unknown> > exports > app.get('/v1/user/:id') callback
  5 module.exports = (app,db) => {
  6   /**
  7    * @tags user
  8    * @param {integer} user_id.path.required - user id to get information
  9    * @return {array<User>} 200 - success response - application/json
 10   */
 11   app.get('/v1/user/:id', (req, res) => {
 12
 13     // Role check (Admin check)
 14     if (!req.session.user || req.session.user.role !== 'admin') {
 15       return res.status(403).json({ error: 'Forbidden: Admin only' });
 16     }
 17
 18     db.user.findOne({
 19       where: { id: req.params.id }
 20     })
 21     .then(user => {
 22       res.json(user);
 23     });
 24   });
 25
 26   /**
 27    * DELETE /v1/user/{user_id}
 28    * @summary Delete a specific user (Broken Function Level Authentication)
 29    * @tags user
 30    * @param {integer} user_id.path.required - user id to delete (Broken Function Level)
 31  
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\Users\ADMIN\vuln-node.js-express.js-app> semgrep --config moh.yaml

Scan Summary

- Scan completed successfully.
- Findings: 1 (1 blocking)
- Rules run: 4
- Targets scanned: 23
- Parsed lines: ~100.0%
- Scan skipped:

#### - Second Semgrep rule

The screenshot shows the VS Code interface with the file `user.js` open. The code contains a path traversal vulnerability. A Semgrep configuration file `jsvuln.yaml` is run from the terminal, resulting in a scan summary showing 0 findings (0 blocking), 3 rules run, 23 targets scanned, and 100.0% parsed lines. The summary also lists skipped files and lines.

```
! jsvuln.yaml
1 rules:
2   - id: node-path-traversal-beer-pic
3     severity: ERROR
4     message: |
5       Path Traversal vulnerability:
6       User input from req.query.picture flows into fs.readFile.
7       This allows attackers to read arbitrary files using ../ sequences.
8     metadata:
9       category: security
10      cwe: "CWE-22"
11      owasp: "A01:2021 - Broken Access Control"
12      technology:
13        - nodejs
14        - express
15      mode: taint
16
17
18 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
19
20 PS C:\Users\ADMIN\vuln-node.js-express.js-app> semgrep --config jsvuln.yaml
21
22 Scan Summary
23
24 ✓ Scan completed successfully.
25 • Findings: 0 (0 blocking)
26 • Rules run: 3
27 • Targets scanned: 23
28 • Parsed lines: ~100.0%
29 • Scan skipped:
30   • Files matching .semgrepignore patterns: 8709
31   • Scan was limited to files tracked by git
32   • For a detailed list of skipped files and lines, run semgrep with the --verbose flag
33 Ran 3 rules on 23 files: 0 findings
```

## - Third Semgrep rule

The screenshot shows the Semgrep UI interface. At the top, there's a progress bar labeled "PROGRESS" with a green bar indicating "100% 0:00:00". Below the progress bar is a "Scan Summary" section. Inside this section, a green checkmark indicates "Scan completed successfully." followed by a detailed list of statistics: Findings: 0 (0 blocking), Rules run: 3, Targets scanned: 23, Parsed lines: ~100.0%, Scan skipped: Files matching .semgrepignore patterns: 9997, Scan was limited to files tracked by git, and For a detailed list of skipped files and lines, run semgrep with the --verbose flag. It also mentions Ran 3 rules on 23 files: 0 findings.

## - Summary:

**-The vulnerabilities have been closed :**

**1-Broken access control**

**2-Injection**

**3-Security misconfiguration**

**4-Cryptographic failures**

**5-Invalidate redirect forward**

**-Tools have been used:**

**1-owasp zap**

**2-postman**

**3-semgrep**

**4-www.jwt.io**