

Report by:

محمد محي الدين عبد اللطيف 2305187

محمود محمد جمعة 2305188

عمر محمد زهير 2305167

Tools and Technologies Used:

Node.js / Express.js: The target application environment

OWASP ZAP: A tool for dynamic vulnerability discovery (DAST)

Postman: Used to perform manual attacks and create proof-of-concept exploits (PoCs)

Semgrep: A static application security testing (SAST) tool for source code analysis

Custom Semgrep Rules: Custom-written rules to detect project-specific vulnerability patterns

Git: Version control system used to manage code changes and security fixes

1. Hardcoded Secrets

Before:

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Home', 'Workspaces', 'API Network', and a search bar labeled 'Search Postman'. The main workspace shows a collection named 'collections' with a single item: 'GET http://localhost:5000/v1/redirect?url=https://evil.com'. Below this, a detailed view of a 'GET' request to 'http://localhost:5000/v1/admin/users' is displayed. The 'Authorization' tab is selected, showing 'JWT Bearer' as the type, 'HS256' as the algorithm, and 'SuperSecret' as the secret key. The response status is '200 OK' with a body containing JSON data representing a user profile.

```
{
  "id": 1,
  "email": "anmedkhalil2005@gmail.com",
  "profile_pic": null,
  "password": "280cb962ac59875b964b87152d234b70",
  "role": "user",
  "name": "محمد محب الدين عبد الله",
  "address": "muhamarram bey",
  "created_at": "2025-12-05T20:13:49.621Z",
  "updated_at": "2025-12-05T20:13:49.621Z",
  "deleted_at": null,
  "createdAt": "2025-12-05T20:13:49.621Z",
  "updatedAt": "2025-12-05T20:13:49.621Z",
  "deletedAt": null,
  "beers": []
}
```

```
secret: "SuperSecret"  
jwt.sign(payload, "SuperSecret")
```

After:

```
secret: process.env.SESSION_SECRET  
jwt.sign(payload, process.env.JWT_SECRET)
```

Impact:

- إزالة hardcoded secrets
- منع تسريب المفاتيح الحساسة

2. Reflected XSS

Before:

The screenshot shows a POST request in Postman to the URL `http://localhost:5000/v1/search/name/test' UNION SELECT id, email, password, role, address, name, created_at, update...`. The request body contains the payload `<script>alert(1)</script>`. The response status is 200 OK, with a response time of 8 ms and a size of 10.97 KB. The response body is an HTML page with the title "Login Basic - Pages | Sweat - Bootstrap 5 HTML Admin Template - Pro". The page includes meta tags for viewport and description, and a link to a favicon.

```
res.render('user.html', { message: req.query.message });
```

After:

```
res.render('user.html', {  
  message: escapeHTML(req.query.message || "")  
});
```

Impact:

- من المدخلات JavaScript منع تفويذ
 - finding reflected XSS حل الخاص بـ

3. IDOR (Insecure Direct Object Reference)

Before:

The screenshot shows the Postman application interface. The top navigation bar includes 'Home', 'Workspaces', 'API Network', a search bar, and an 'Upgrade' button. On the left, there are sidebar panels for 'Collections', 'Environments', 'History', 'APIs', 'Mock servers', 'Monitors', and 'Flows'. The main workspace displays a GET request to 'http://localhost:5000/v1/user/3'. The 'Authorization' tab is selected, showing 'No Auth'. The 'Body' tab is selected, displaying a JSON response with fields: id, email, password, role, address, name, created_at, updated_at, deleted_at, and beers. The response status is 200 OK with a duration of 8 ms and a size of 646 B.

```
[{"id": 3, "email": "ahmedkhalil20056@gmail.com", "password": "202cb962ac59075b964b07152d234b70", "role": "user", "address": "كوجي الشامي / رمل نابي", "name": "محمد محب", "created_at": "2025-12-11T19:45:18.220Z", "updated_at": "2025-12-11T19:45:18.220Z", "deleted_at": null, "createdAt": "2025-12-11T19:45:18.220Z", "updatedAt": "2025-12-11T19:45:18.220Z", "deletedAt": null, "beers": []}]
```

الوصول يتم عبر
`/user/3`
للوصول لحسابات أخرى `id` ويمكن تغيير

After:

```
if (tokenData.id != req.params.id && tokenData.role !== 'admin') {  
  return 500 Forbidden  
}
```

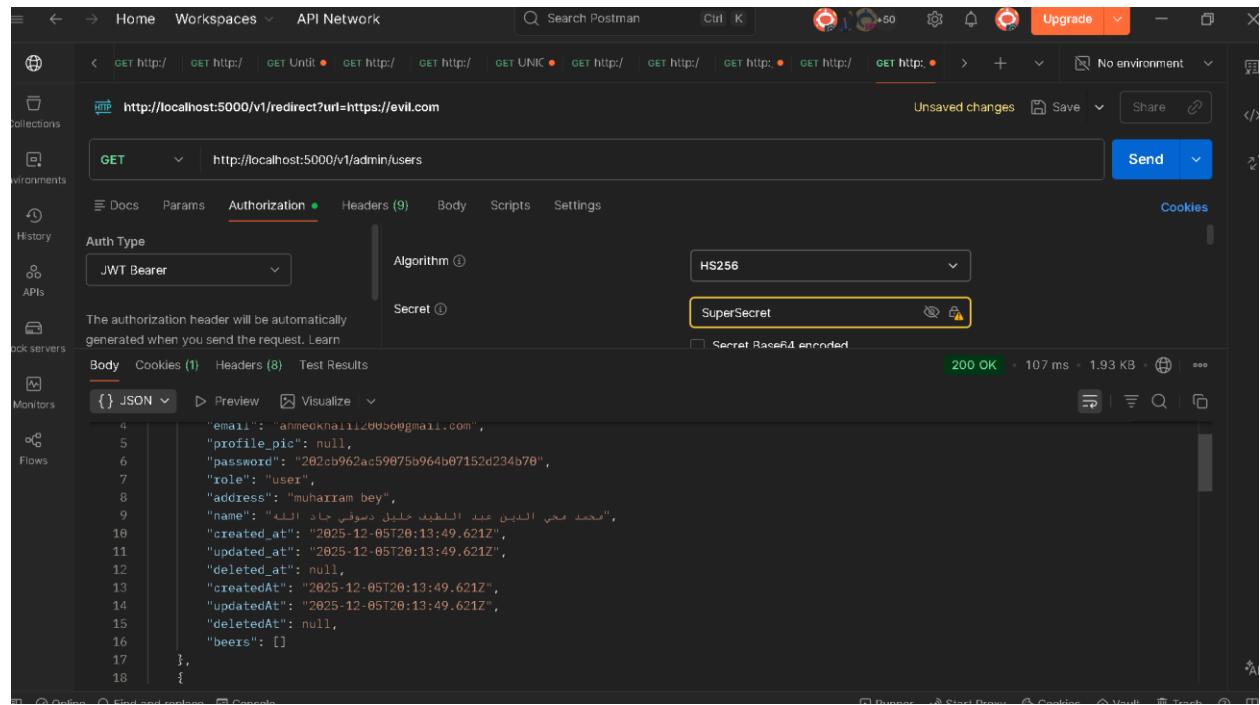
Impact:

- المستخدم يرى بياناته فقط
- منع الوصول غير المصرح به

4. JWT Misconfiguration

4.1 No Expiration

Before:



The screenshot shows a Postman collection named "Collections" containing a single item. The item has a "GET" method pointing to "http://localhost:5000/v1/admin/users". In the "Authorization" tab, the "Auth Type" is set to "JWT Bearer" and the "Secret" is "SuperSecret". The response status is "200 OK" with a response time of 107 ms and a size of 1.93 KB. The response body is a JSON object:

```
4 {"email": "ahmedkhalil2000@gmail.com", "profile_pic": null, "password": "202cb962ac59875b964b07152d234b70", "role": "user", "name": "محمد عاصي الدين عبد اللطيف جليل دسوقي جاد الله", "address": "muhairim bey", "created_at": "2025-12-05T20:13:49.621Z", "updated_at": "2025-12-05T20:13:49.621Z", "deleted_at": null, "createdAt": "2025-12-05T20:13:49.621Z", "updatedAt": "2025-12-05T20:13:49.621Z", "deletedAt": null, "beers": []}, {
```

jwt.sign(payload, secret)

After:

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:5000/v1/admin/users`. The response status is `500 Internal Server Error`. The response body contains an HTML error page with the title "Error" and a stack trace:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8">
5 <title>Error</title>
6 </head>
7 <body>
8 <pre>Error: secret option required for sessions<br>at session (C:\Users\BAB AL SAFA\Desktop\vuln-node.js-express.js-app-main\vuln-node.js-express.js-app-main\node_modules\express-session\index.js:200:12)<br>&ampnbspat Layer.handle [as handle_request] (C:\Users\BAB AL SAFA\Desktop\vuln-node.js-express.js-app-main\vuln-node.js-express.js-app-main\node_modules\express\lib\router\layer.js:95:5)<br>&ampnbspat trim_prefix (C:\Users\BAB AL SAFA\Desktop\vuln-node.js-express.js-app-main\vuln-node.js-express.js-app-main\node_modules\express\lib\router\index.js:328:13)<br>&ampnbspat C:\Users\BAB AL SAFA\Desktop\vuln-node.js-express.js-app-main\vuln-node.js-express.js-app-main\node_modules\express\lib\router\index.js:286:9<br>&ampnbspat Function.process_params (C:\Users\BAB AL SAFA\Desktop\vuln-node.js-express.js-app-main\vuln-node.js-express.js-app-main\node_modules\express\lib\router\index.js:286:17)<br>Error: Error: jwt sign failed: jwt.SigningError: Missing secret option. Did you mean to use 'secret' or 'key'?
```

`jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: 86400 })`

4.2 Trusting Client Role

Before:

`role: req.body.role`

After:

`role: user.role (من قاعدة البيانات)`

Impact:

- منع Privilege Escalation
- دائم منع Tokens

5. User Enumeration

Before:

"User not found"

"Wrong password"

After:

"Invalid credentials"

Impact:

- منع المهاجم من معرفة الحسابات الموجودة

6. CSRF (State Change via GET)

Before:

GET /v1/love/:beer_id

After:

GET → Method Not Allowed (405)

POST /v1/love/:beer_id

Impact:

- منع تغيير الحالة عبر GET
- تقليل خطر CSRF

7. Session Fixation & Cookie Flags

Before:

saveUninitialized: true

httpOnly: false

After:

saveUninitialized: false

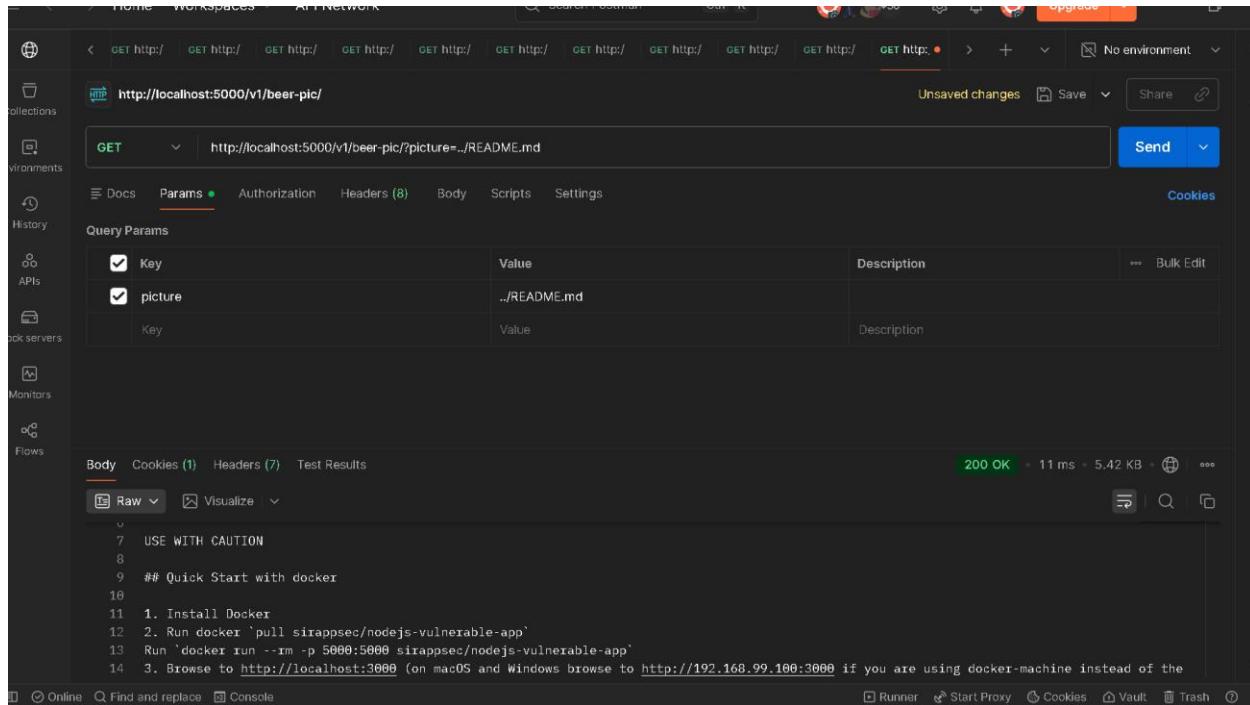
httpOnly: true

Impact:

- منع سرقة الجلسة عبر XSS
- منع Session Fixation

8. Path Traversal

Before:

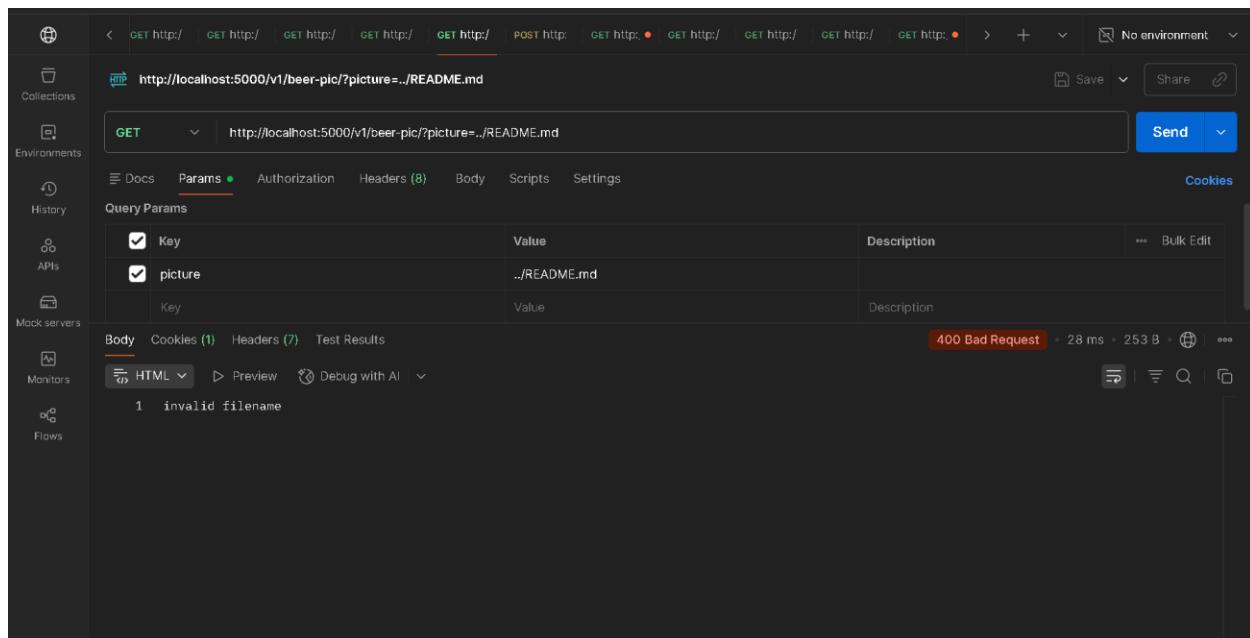


The screenshot shows the Postman interface with a successful HTTP request. The URL is `http://localhost:5000/v1/beer-pic/`. In the 'Params' tab, there is a key 'picture' with a value of `../README.md`. The response status is 200 OK, and the body contains the contents of the README.md file.

```
7 USE WITH CAUTION
8
9 ## Quick Start with docker
10
11 1. Install Docker
12 2. Run docker `pull sirappsec/nodejs-vulnerable-app`
13 Run `docker run --rm -p 5000:5000 sirappsec/nodejs-vulnerable-app`
14 3. Browse to http://localhost:3000 (on macOS and Windows browse to http://192.168.99.100:3000 if you are using docker-machine instead of the default Docker daemon)
```

`fs.readFile(req.query.file)`

After:



The screenshot shows the Postman interface with an unsuccessful HTTP request. The URL is `http://localhost:5000/v1/beer-pic/`. In the 'Params' tab, there is a key 'picture' with a value of `../README.md`. The response status is 400 Bad Request, and the body contains the message 'invalid filename'.

`fs.readFile(path.basename(req.query.file))`

Impact:

- منع قراءة ملفات خارج المسار المسموح

9. OTP Hardcoded Secret

Before:

```
const seed = "SUPERSECUREOTP";
```

After:

```
const seed = process.env.OTP_SECRET;
```

Impact:

- خارج الكود Secret نقل
- الحفاظ على نفس السلوك الوظيفي

Some code used in Attack:

Path traversal:

<http://localhost:5000/v1/beer-pic/?picture=../README.md>

<http://localhost:5000/v1/beer-pic/?picture=../../package.json>

SQL:

`http://localhost:5000/v1/search/name/admin'`

UNION SELECT

`id, email, password, role, profile_pic, address, name, created_at, updated_at`

`FROM users`

`WHERE role='admin'--`

RCE – Remote Code Execution:

`http://localhost:5000 /v1/status/bud | whoami`

Insecure Redirect:

[http://localhost:5000 /v1/test?url=http://127.0.0.1:5000](http://localhost:5000/v1/test?url=http://127.0.0.1:5000)

JWT:

<http://localhost:5000/v1/admin/users>

Broken Function Level Auth [DELETE <http://localhost:5000/v1/user/2>](DELETE http://localhost:5000/v1/user/2)

IDOR / BOLA (Get User Data) [GET <http://localhost:5000/v1/user/2>](GET http://localhost:5000/v1/user/2)

Weak Password [POST <http://localhost:5000/v1/user>](POST http://localhost:5000/v1/user)

CSRF [POST <http://localhost:5000/v1/love/1?id=2>](POST http://localhost:5000/v1/love/1?id=2)

POST <http://localhost:5000/v1/user/1/validate-otp?seed=AAA&token=123456>

OTP Broken Auth

GET [http://localhost:5000/?message=<script>alert\('XSS'\)</script>](http://localhost:5000/?message=<script>alert('XSS')</script>)

XSS

http://localhost:5000/?message={{7*7}}

SSTI – Server Side Template Injection

Final Result Summary:

Reflected XSS: Fixed

IDOR: Fixed

CSRF: Fixed

JWT Issues: Fixed

Hardcoded Secrets: Fixed

Session Misconfiguration: Fixed

Privilege Escalation: Fixed

Semgrep Findings: Resolved

Semgrep before fix by semgrep and my yaml rule:

The screenshot shows a terminal window on a Kali Linux system. The terminal title is 'root@kali: /home/mohamed/secure'. The command run was 'semgrep --rules vuln-node.js-express.js-app/src/server.js'. The output shows a single finding:

```
362 | res.send(user)
vuln-node.js-express.js-app/src/server.js
» missing-helmet-middleware
    Express app missing Helmet middleware for security headers
17 | const app = express()
```

A 'Scan Summary' box is highlighted with a red rectangle, containing the following information:

Scan completed successfully.
• Findings: 9 (9 blocking)
• Rules run: 8
• Targets scanned: 23
• Parsed lines: ~100.0%
• Scan skipped:
◦ Files matching .semgrepignore patterns: 25
• Scan was limited to files tracked by git
• For a detailed list of skipped files and lines, run semgrep with the --verbose flag

Ran 8 rules on 23 files: 9 findings.

(root㉿kali)-[/home/mohamed/secure]

```
Session Actions Edit View Help
root@kali: /home/mohamed/secure

>>> reflected-xss-response
    Potential reflected XSS via unsanitized user input in response

35| res.send("error")
:   _____
41| res.send(data)
:   _____
45| res.send(buffer)

vuln-node.js-express.js-app/src/router/routes/system.js
>>> reflected-xss-response
    Potential reflected XSS via unsanitized user input in response

18| res.send(test)

vuln-node.js-express.js-app/src/router/routes/user.js
>>> reflected-xss-response
    Potential reflected XSS via unsanitized user input in response

334| res.send(user)
:   _____
362| res.send(user)

vuln-node.js-express.js-app/src/server.js
```

```
File Edit View Help
root@kali: /home/mohamed/secure/vuln-node.js-express.js-app
(file) you're telling the browser to fetch in the 'integrity' attribute for
hosted files.
Details: https://sg.run/krXA
225| <script async defer src="https://buttons.github.io/buttons.js"></script>

[Scan Summary]
✓ Scan completed successfully.
  • Findings: 32 (32 blocking)
  • Rules run: 259
  • Targets scanned: 90
  • Parsed lines: ~99.9%
  • Scan skipped:
    • Files matching .semgrepignore patterns: 25
    • Scan was limited to files tracked by git
    • For a detailed list of skipped files and lines, run semgrep with the --verbose flag
      I
      Ran 259 rules on 90 files: 32 findings.
      Missed out on 1390 pro rules since you aren't logged in!
      ↪ Supercharge Semgrep OSS when you create a free account at https://sg.run/rules.
      ↪ Too many findings? Try Semgrep Pro for more powerful queries and less noise.
      See https://sg.run/false-positives.
└─(root㉿kali)-[/home/mohamed/secure/vuln-node.js-express.js-app]
```

Session Actions Edit View Help

root@kali: /home/mohamed/securevuls-node.js-express-js-app

100% 0:00:01

32 Code Findings

```
populate.sh
>>> generic.secrets.security.detected-jwt-token.detected-jwt-token
JWT token detected
Details: https://sg.run/05N5
33| -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwicm9sZS
I6ImFkbWluIiwiaWF0IjoxNjU5NDI3MjQwLCJleHAiOjE2NTk1MTM2NDB9.QpYD330lcw_AxJR16FKGWu
gQk0HNHRhOqonaH75GUDE'

src/middleware/authJwt.js
>> javascript.jsonwebtoken.security.audit.jwt-decode-without-verify.jwt-decode-without-verify
Detected the decoding of a JWT token without a verify step. JWT tokens must be verified
before use, otherwise the token's integrity is unknown. This means a malicious actor could
forge a JWT token with any claims. Call '.verify()' before using the token.
Details: https://sg.run/J9YP

13|     jwt.decode(token, (err, decoded) => {
14|       if (err) {
15|         return res.status(401).send({
16|           message: "Unauthorized!"
```

```
vuln-node.js-express.js-app/src/router/routes/admin.js
>>> reflected-xss-response
    Potential reflected XSS via unsanitized user input in response
111| res.send(err.toString());

vuln-node.js-express.js-app/src/router/routes/order.js
>>> insecure-file-path
    User input used directly in filesystem path (Path Traversal risk)
33| fs.readFile(path.join(__dirname, filePath),function(err,data){
34|     if (err){
35|         res.send("error")
36|     }else{
37|         if(filename.split('.').length == 1)
38|         {
39|             res.type('image/jpeg')
40|             //res.set('Content-Type', 'image/jpg');
41|             res.send(data)
42|             return;
        [hid 7 additional lines, adjust with --max-lines-per-finding]
>>> reflected-xss-response
```

Session Actions Edit View Help

root@kali: /home/mohamed/secure

GNU nano 8.6 myrule.yml

```
- id: no-login-validation
  languages: [javascript]
  severity: WARNING
  message: "Login endpoint missing input validation/sanitization"
  pattern: |
    app.post("/api/login", ($REQ, $RES) => {
      $USERNAME = $REQ.body.username
      $PASSWORD = $REQ.body.password
      // بدون تحقق
    })
  # Rule 8: Missing Rate Limiting on Sensitive Routes
- id: missing-rate-limit
  languages: [javascript]
  severity: WARNING
  message: "Sensitive endpoint missing rate limiting middleware"
  pattern: |
    app.post("/api/login", $HANDLER)
    # ,
    app.post("/api/reset-password", $HANDLER)
```

^G Help ^O Write Out ^F Where Is ^K Cut ^T Execute ^C Locate
^X Exit ^R Read File ^V Replace ^U Paste ^J Justify ^/ Go To

```
GNU nano 8.6                                         myrule.yml
rules:
# Rule 1: SQL Injection via String Concatenation
- id: insecure-sql-concat
  languages: [javascript]
  severity: ERROR
  message: "Potential SQL injection via string concatenation"
  pattern: |
    $QUERY = "SELECT" + $INPUT + "FROM" + $TABLE
# اسفل بعدها
# pattern: |
#   $DB.query("SELECT ... " + $USER_INPUT + " ... ")
# Rule 2: Reflected XSS via unsanitized response
- id: reflected-xss-response
  languages: [javascript]
  severity: ERROR
  message: "Potential reflected XSS via unsanitized user input in response"
  pattern: |
    res.send($USER_INPUT)

# Rule 3: Missing JWT Verification (algorithm none)
[ Read 77 lines ]
^G Help          ^O Write Out      ^F Where Is      ^K Cut           ^T Execute       ^C Location
^X Exit          ^R Read File     ^\ Replace        ^U Paste         ^J Justify       ^/ Go To
```

The screenshot shows a terminal window titled "myrule.yml" being edited in the "GNU nano 8.6" text editor. The file contains configuration rules for security analysis:

```
# Rule 5: Missing Helmet Security Headers
- id: missing-helmet-middleware
  languages: [javascript]
  severity: WARNING
  message: "Express app missing Helmet middleware for security headers"
  pattern: |
    const app = express()
    // بدون app.use(helmet())

# Rule 6: Insecure File Path from User Input
- id: insecure-file-path
  languages: [javascript]
  severity: ERROR
  message: "User input used directly in filesystem path (Path Traversal risk)"
  pattern: |
    fs.readFile($USER_INPUT, ...)

# Rule 7: No Input Validation on Login Route
- id: no-login-validation
  languages: [javascript]
  severity: WARNING
```

The terminal window includes a menu bar with "Session Actions Edit View Help" and a status bar showing "root@kali: /home/mohamed/secure". The bottom of the window displays a series of keyboard shortcuts for various operations like Help, Exit, Write Out, Read File, Where Is, Replace, Cut, Paste, Execute, Justify, and Go.

```
GNU nano 8.6 myrule.yml
# Rule 3: Missing JWT Verification (algorithm none)
- id: insecure-jwt-verify
  languages: [javascript]
  severity: ERROR
  message: "Insecure JWT verification - missing algorithm or using 'none'
  pattern: |
    jwt.verify($TOKEN, $SECRET, { algorithms: ["none"] })
    # نمای بیرون خوارزمه محدود
    # pattern: |
    #   jwt.verify($TOKEN, $SECRET)

# Rule 4: Hardcoded Secret in Code
- id: hardcoded-secret
  languages: [javascript]
  severity: ERROR
  message: "Hardcoded secret or password in source code"
  pattern: |
    $SECRET = "password123" or $SECRET = "supersecret"

# Rule 5: Missing Helmet Security Headers
- id: missing-helmet-middleware
```

^G Help ^O Write Out ^F Where Is ^K Cut
^X Exit ^R Read File ^\ Replace ^U Paste ^T Execute
^J Justify

```
18| const user_object = jwt.verify(req.headers.authorization.split('')[1], "SuperSecret")
»> javascript.jsonwebtoken.security.audit.jwt-decode-without-verify.jwt-decode-without-verify
Detected the decoding of a JWT token without a verify step. JWT tokens must be verified before use, otherwise the token's integrity is unknown. This means a malicious actor could forge a JWT token with any claims. Call '.verify()' before using the token.
Details: https://sg.run/J9YP

182| current_user_id = jwt.decode(req.headers.authorization.split(' '))[1].id
»> javascript.jsonwebtoken.security.jwt-hardcode.hardcoded-jwt-secret
A hard-coded credential was detected. It is not recommended to store credentials in source-code, as this risks secrets being leaked and used by either an internal or external malicious adversary. It is recommended to use environment variables to securely provide credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).
Details: https://sg.run/4xN9

253| var token = jwt.sign(payload, jwtTokenSecret, {
»»> javascript.express.security.express-wkhtml-injection.express-wkhtmltoimage-injection
If unverified user data can reach the `phantom` methods it can result in Server-Side Request Forgery vulnerabilities
Details: https://sg.run/pxe0

398| const GeneratedToken = otplib.authenticator.generate(seed);
»> javascript.jsonwebtoken.security.jwt-hardcode.hardcoded-jwt-secret
```

```
Session Actions Edit View Help
root@kali: /home/mohamed/secure/vuln-nodejs-express.js-app
67| const beers = db.sequelize.query(sql, { type: 'RAW' }).then(beers => {
src/router/routes/system.js
  >> javascript.express.security.audit.xss.direct-response-write.direct-response-write
    Detected directly writing to a Response object from user-defined input. This bypasses any
    HTML escaping and may expose your application to a Cross-Site-scripting (XSS)
    vulnerability. Instead, use 'resp.render()' to render safely escaped HTML.
    Details: https://sg.run/vzGl
18|   res.send(test)
I
  >> javascript.express.security.audit.express-open-redirect.express-open-redirect
    The application redirects to a URL specified by user-supplied input `req` that is not
    validated. This could redirect users to malicious locations. Consider using an allow-list
    website.
    Details: https://sg.run/EpoP
37|   res.redirect(url);
  >> javascript.express.security.audit.express-third-party-object-deserialization.express-third-party-
object-deserialization
    The following function call serialize.unserialize accepts user controlled data which can
    result in Remote Code Execution (RCE) through Object Deserialization. It is recommended to
    use secure data processing alternatives such as JSON.parse() and Buffer.from().
    Details: https://sg.run/8W5j
64| var deser = serialize.unserialize(body)
```

```
File Manager | 1 2 3 4 | 
Actions Edit View Help
the intended destination
Details: https://sg.run/weRn
33| fs.readFile(path.join(__dirname, filePath),function(err,data){
>>> javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-traversal
Detected possible user input going into a `path.join` or `path.resolve` function. This could possibly lead to a path traversal vulnerability, where the attacker can access arbitrary files stored in the file system. Instead, be sure to sanitize or validate user input first.
Details: https://sg.run/OPqk
33| fs.readFile(path.join(__dirname, filePath),function(err,data){
>>> javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection
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 => {
src/router/routes/system.js
>>> javascript.express.security.audit.xss.direct-response-write.direct-response-write
Detected directly writing to a Response object from user-defined input. This bypasses any HTML escaping and may expose your application to a Cross-Site-scripting (XSS) vulnerability. Instead, use 'resp.render()' to render safely escaped HTML.
Details: https://sg.run/vzGl
```

The terminal window shows the output of a Semgrep scan on a Node.js application. The title bar indicates the session is running on a Kali Linux VM. The command run was `semgrep -c .github/workflows/semgrep.yml --fix`.

```
Session Actions Edit View Help
object-deserialization
The following function call serialize.unserialize accepts user controlled data which
result in Remote Code Execution (RCE) through Object Deserialization. It is recommended
use secure data processing alternatives such as JSON.parse() and Buffer.from().
Details: https://sg.run/8W5j

64| var deser = serialize.unserialize(body)
src/router/routes/user.js
»» javascript.jsonwebtoken.security.jwt-hardcode.hardcoded-jwt-secret
A hard-coded credential was detected. It is not recommended to store credentials in
code, as this risks secrets being leaked and used by either an internal or external
malicious adversary. It is recommended to use environment variables to securely
store credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).
Details: https://sg.run/4xN9

18| const user_object = jwt.verify(req.headers.authorization.split(' ')
')[1], "SuperSecret")

»» javascript.jsonwebtoken.security.audit.jwt-decode-without-verify.jwt-decode-without-verifying
Detected the decoding of a JWT token without a verify step. JWT tokens must be verified
before use, otherwise the token's integrity is unknown. This means a malicious actor could
forge a JWT token with any claims. Call '.verify()' before using the token.
Details: https://sg.run/J9YP

182| current_user_id = jwt.decode(req.headers.authorization.split(' ')[1]).id

»» javascript.jsonwebtoken.security.jwt-hardcode.hardcoded-jwt-secret
A hard-coded credential was detected. It is not recommended to store credentials in
```

Semgrep after fix with my yaml rule:

Player | || ⌂ ⌂

Session Actions Edit View Help

root@kali:/home/mohamed/secure

```
secure-node.js-express.js-app-main/vuln-node.js-express.js-app-main/src/router/routes/order.js
>>> insecure-file-path
    User input used directly in filesystem path (Path Traversal risk)
45|   fs.readFile(fullPath, (err, data) => {
46|     if (err) {
47|       return res.status(404).send('file not found');
48|     }
49|
50|     res.type('image/jpeg');
51|     res.send(data);
52|   });

>>> reflected-xss-response
    Potential reflected XSS via unsanitized user input in response
51|   res.send(data);

secure-node.js-express.js-app-main/vuln-node.js-express.js-app-main/src/server.js
>> missing-helmet-middleware
    Express app missing Helmet middleware for security headers
11|   const app = express();
```

Scan Summary

- ✓ Scan completed successfully.
- Findings: 3 (3 blocking)
- Rules run: 8
- Targets scanned: 23
- Parsed lines: ~100.0%
- Scan skipped:
 - Files matching .semgrepignore patterns: 17014
 - Scan was limited to files tracked by git
 - For a detailed list of skipped files and lines, run semgrep with the --verbose flag
- Ran 8 rules on 23 files: 3 findings.

(root@kali)-[/home/mohamed/secure]