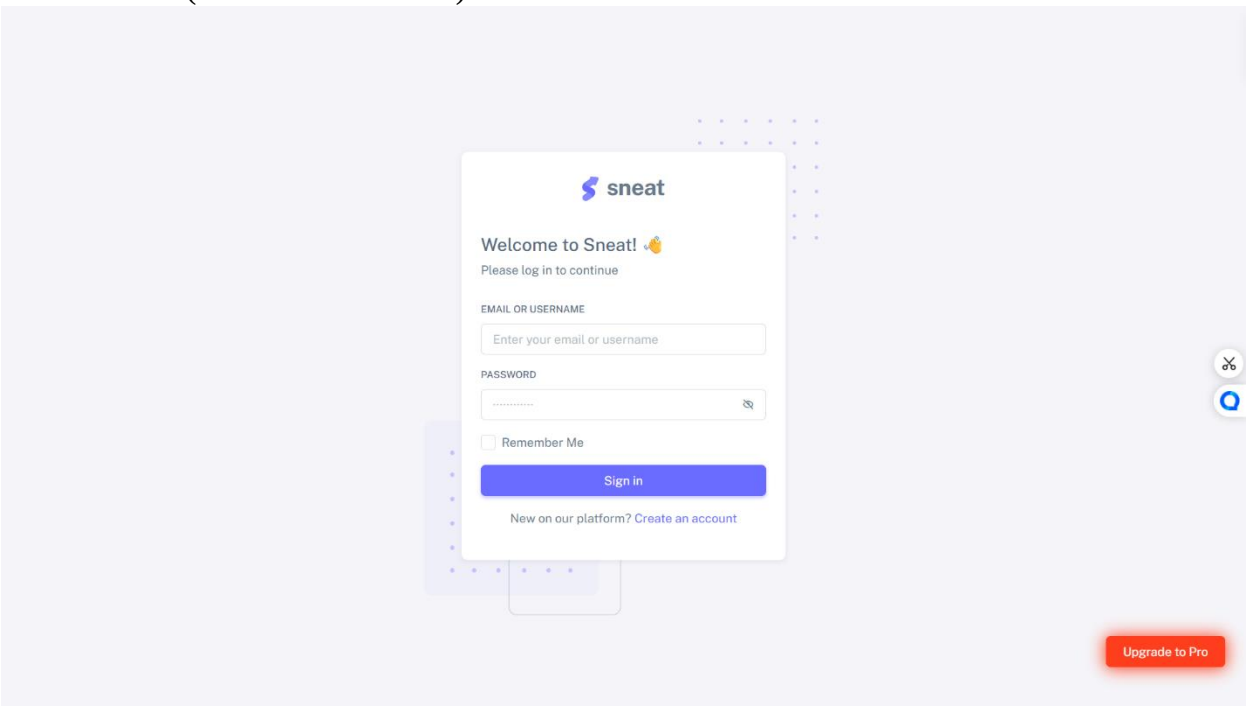
Dynamic Application Security Testing (DAST) Report

1. Project Overview

Application Name: Damn Vulnerable Node.js Express App
Base URL: http://localhost:5000
Testing Type: Dynamic Application Security Testing (DAST)
Testing Date: —

Tools Used:

- OWASP ZAP (Automated Scanning)
- Postman (Manual Testing & PoCs)
- Web Browser (Manual Verification)



2. Scope of Testing

The scope of this assessment includes all publicly accessible endpoints exposed by the application running on the local environment. The testing focused on identifying common web application vulnerabilities based on the OWASP Top 10.

Out of scope:

- Source code review (covered in Phase B – SAST)
- Infrastructure and OS-level hardening

---

3. Methodology

The assessment was conducted using the following approach:

1. **Automated DAST** using OWASP ZAP to enumerate endpoints and identify potential vulnerabilities.
2. **Manual Verification** using Postman and browser-based testing to validate findings and create Proof of Concepts (PoCs).
3. **OWASP Top 10 Mapping** for each confirmed vulnerability.

---

4. Phase A – Automated Testing (OWASP ZAP)

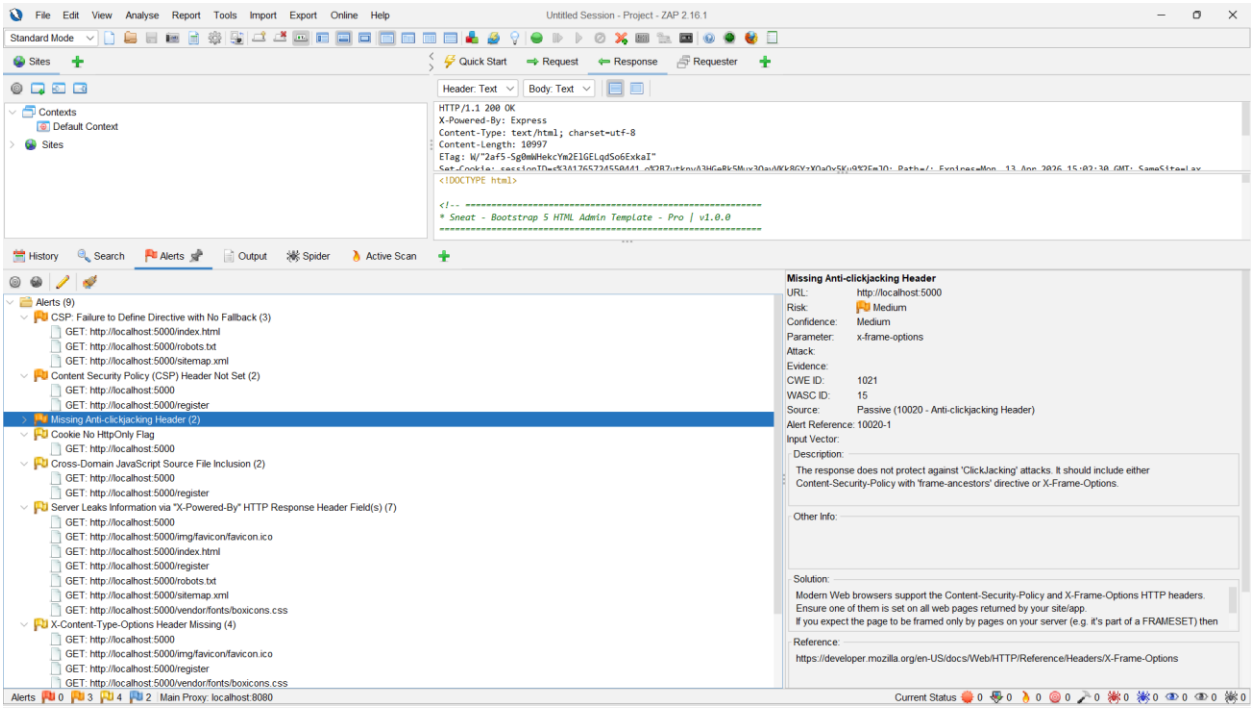
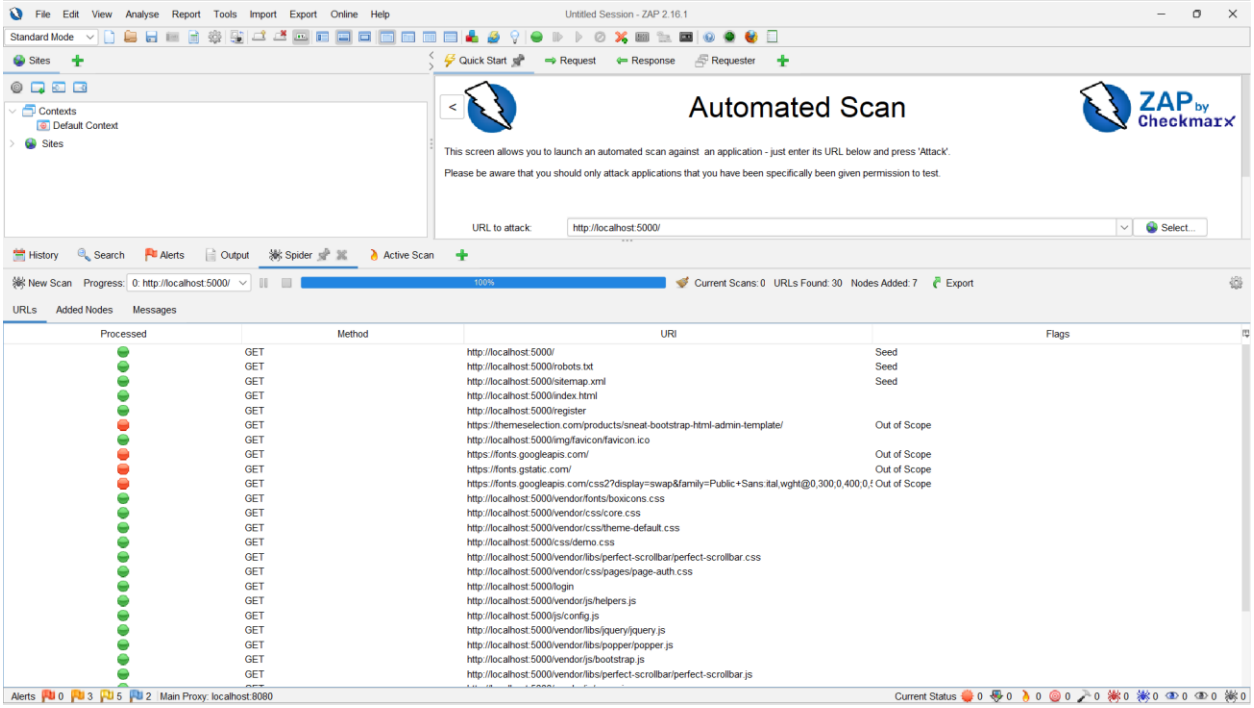
4.1 Discovered Endpoints (Examples)

- /api-docs
- /v1/admin/promote/{id}
- /v1/search/{filter}/{query}
- /v1/beer-pic/
- /v1/test/
- /v1/redirect/
- /v1/status/{brand}
- /
- /v1/user/{id}

4.2 ZAP Findings Summary

OWASP ZAP reported multiple high-risk issues, including:

- Possible SQL Injection

- Reflected Cross-Site Scripting (XSS)
- Path Traversal
- Server-Side Request Forgery (SSRF)
- Broken Access Control
- 📸 *Screenshot:*
  - 
  - 

5. Phase A – Manual Testing & Proof of Concepts (PoCs)

Note: Vulnerabilities are reordered for clarity based on OWASP Top 10 categories. Technical details remain unchanged.

🔴 V1 – Vertical Privilege Escalation (Admin Promotion)

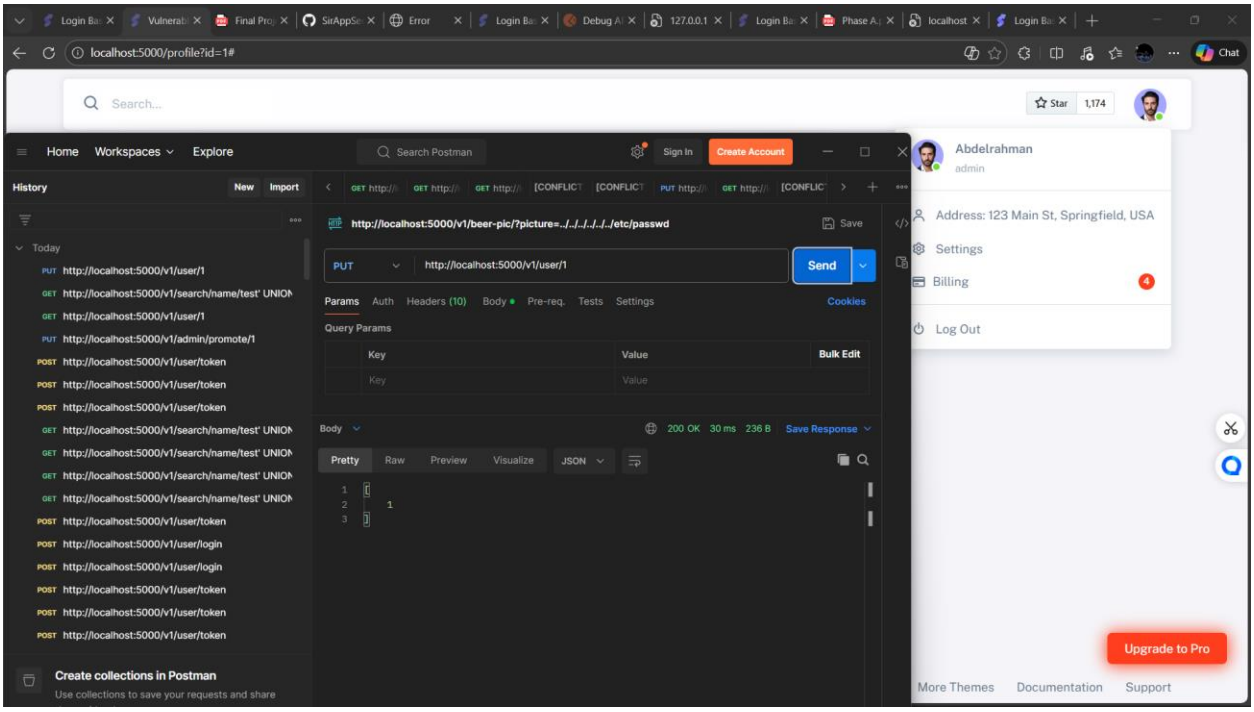
- **Endpoint:** PUT /v1/admin/promote/{user_id}
- **Full URL:** http://localhost:5000/v1/admin/promote/1
- **OWASP Category:** A01 – Broken Access Control

Description:
Any authenticated or unauthenticated user can promote themselves to admin.

Impact:
Full administrative privileges gained.

Expected Result:
User role changes to admin.

📸 *Screenshot:*



🔴 V2 – Insecure Direct Object Reference (IDOR)

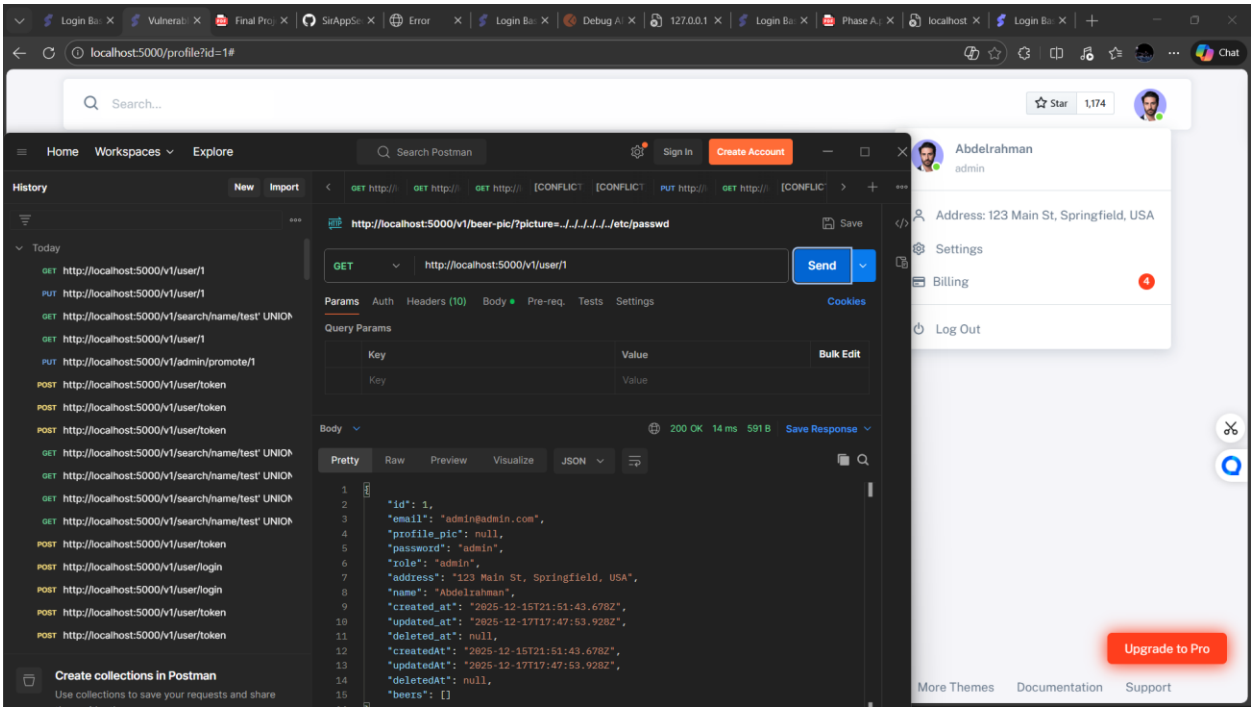
- **Endpoint:** GET /v1/user/{id}
- **Full URL:** http://localhost:5000/v1/user/1
- **OWASP Category:** A01 – Broken Access Control

Description:
User IDs can be modified to access other users' data without authorization.

Impact:
Unauthorized access to sensitive user information.

Expected Result:
Another user's data is returned.

📸 *Screenshot:*



🔴 V3 – Server-Side Request Forgery (SSRF)

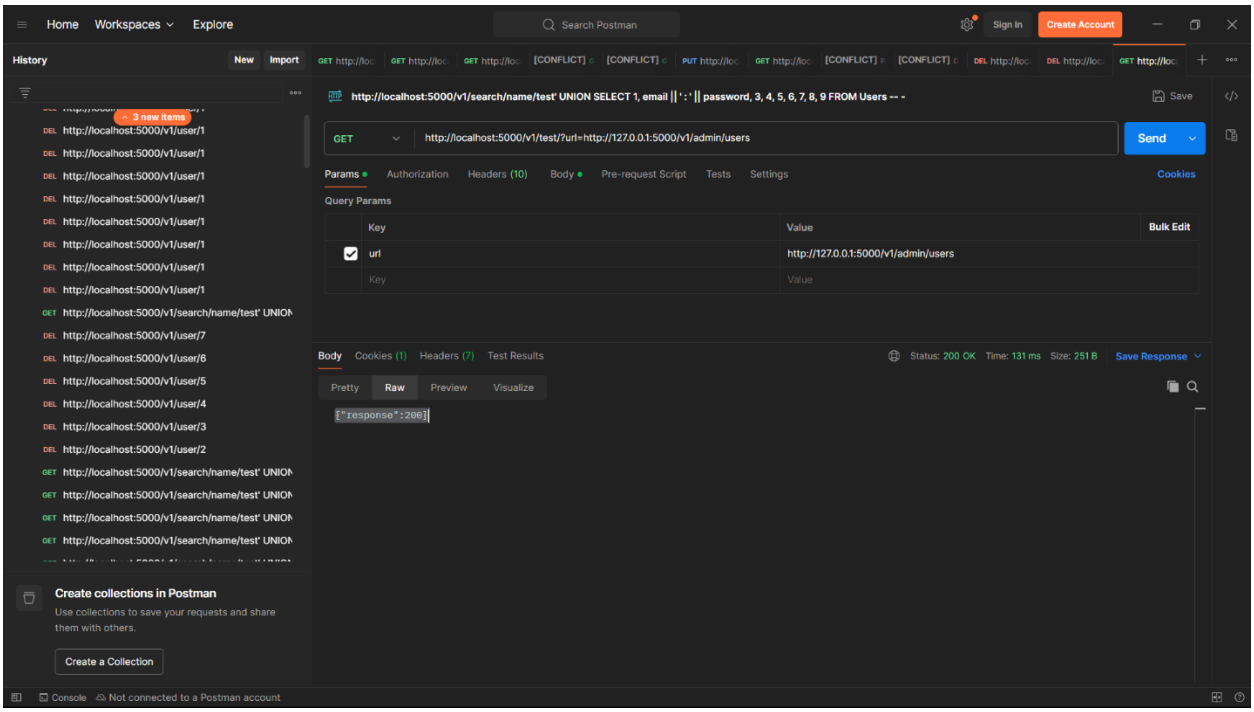
- **Endpoint:** GET /v1/test/
- **Payload:** url=http://127.0.0.1:5000/v1/admin/users
- **Full URL:** http://localhost:5000/v1/test/?url=http://127.0.0.1:5000/v1/admin/users
- **OWASP Category:** A10 – Server-Side Request Forgery (SSRF)

Description:
Server performs internal requests based on user-controlled input.

Impact:
Access to internal admin endpoints.

Expected Result:
The server confirms connectivity to the internal admin endpoint by returning a **200 OK status**, proving it can scan internal ports

📸 *Screenshot:*



🔴 V4– Path Traversal

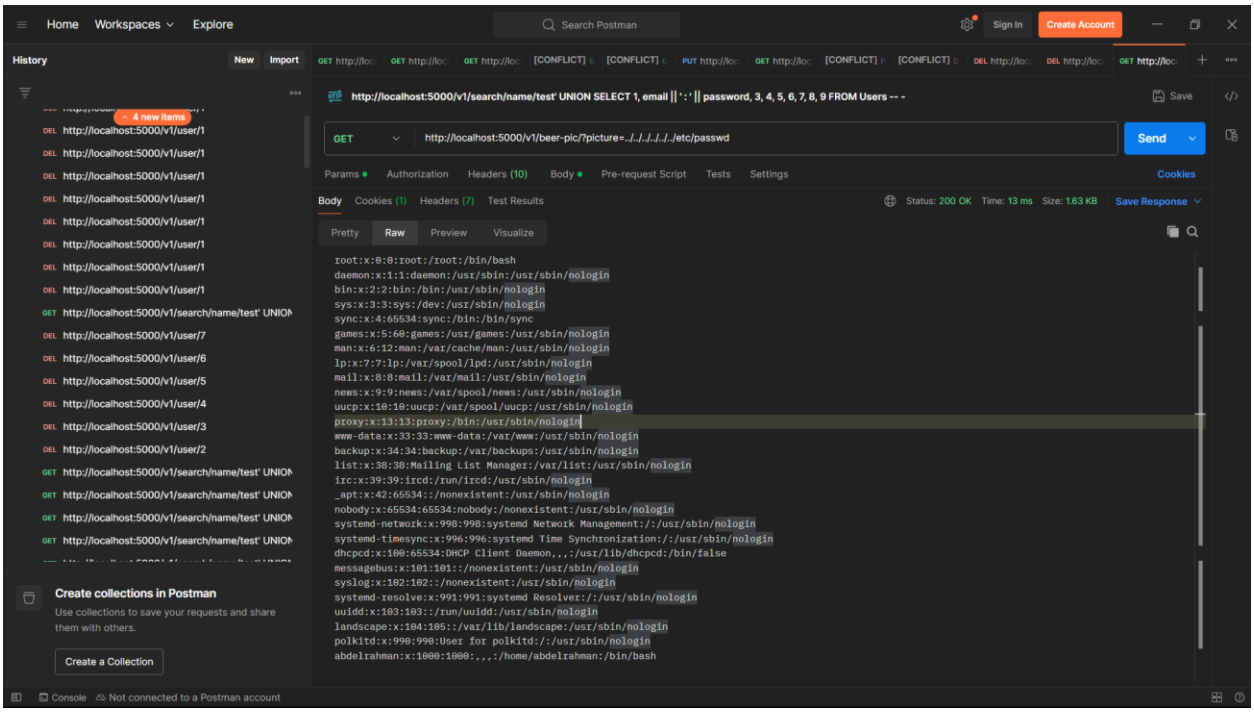
- **Endpoint:** GET /v1/beer-pic/
- **Payload:** ../../../../../../etc/passwd
- **Full URL:** http://localhost:5000/v1/beer-pic/?picture=../../../../../../etc/passwd
- **OWASP Category:** A05 – Security Misconfiguration

Description:
Improper file path validation allows directory traversal.

Impact:
Disclosure of sensitive server files.

Expected Result:
/etc/passwd content returned.

📸 *Screenshot:*



🔴 V5 – Open Redirect

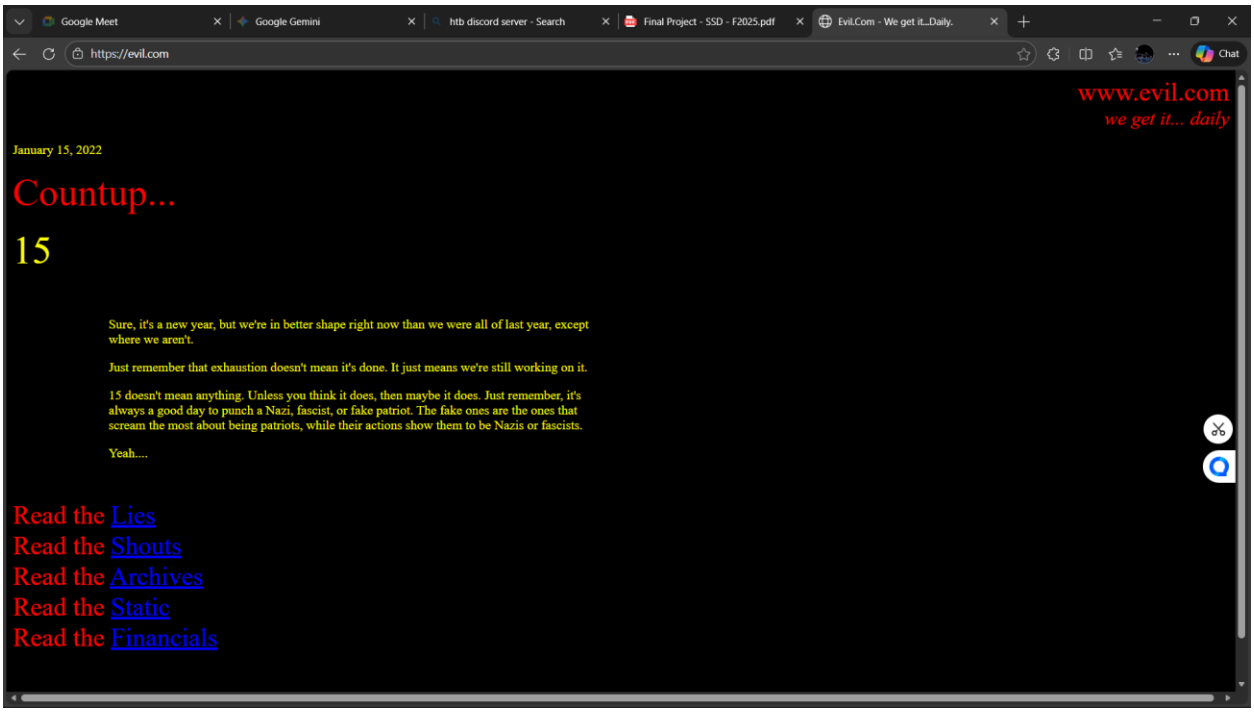
- **Endpoint:** GET /v1/redirect/
- **Payload:** https://evil.com
- **Full URL:** http://localhost:5000/v1/redirect/?url=https://evil.com
- **OWASP Category:** A05 – Security Misconfiguration

Description:
Application redirects users to untrusted URLs.

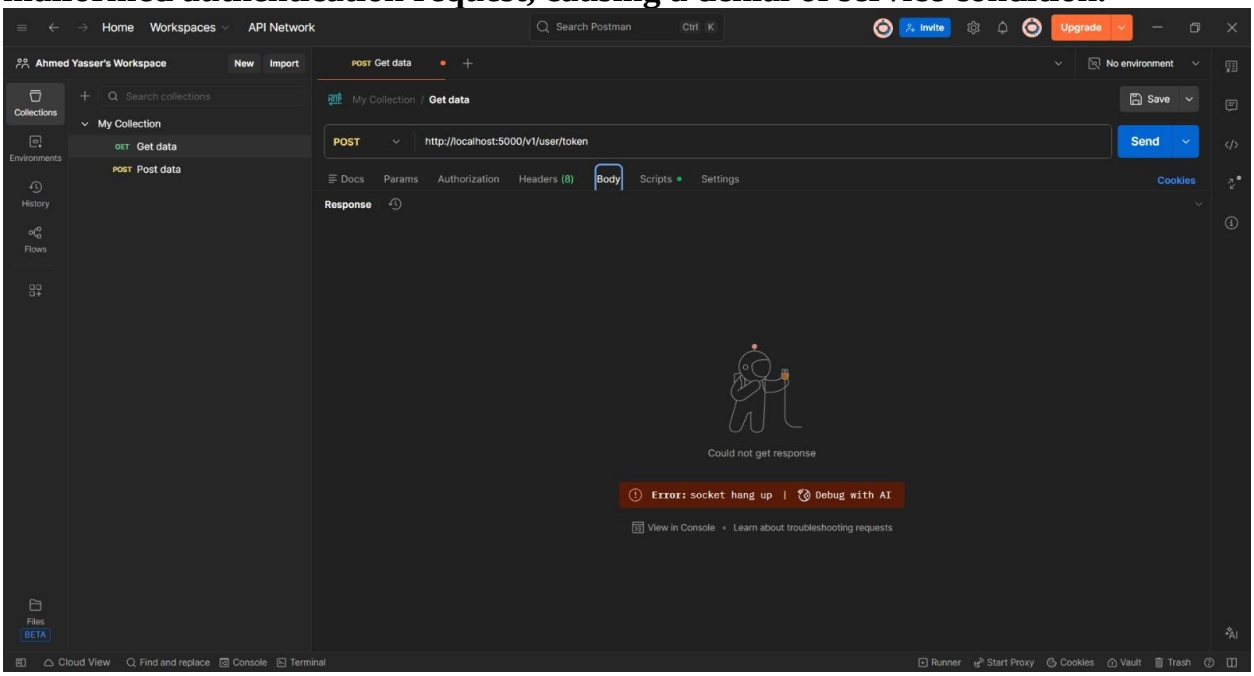
Impact:
Phishing and malicious redirection attacks.

Expected Result:
Browser redirects to external domain.

📸 *Screenshot:*



🔴 V6 – Unhandled Exception → Application Crash

- Endpoint : POST / v1/user/login,token
- Payload : v1/user/login,token
- FULL URL : **http://localhost:5000/v1/user/token**
- OWASP Category : A09 – Security Logging & Monitoring Failures
- **Denial of Service (DoS)**
- **Impact: An attacker can crash the entire backend application by sending a malformed authentication request, causing a denial of service condition.**
- 

🔴 V7 – SQL Injection

- **Endpoint:** GET /v1/search/{filter}/{query}
- **Payload:** `test' UNION SELECT 1,'TEST_HTML',3,4,5,6,7,8,9-- -`
- **OWASP Category:** A03 – Injection

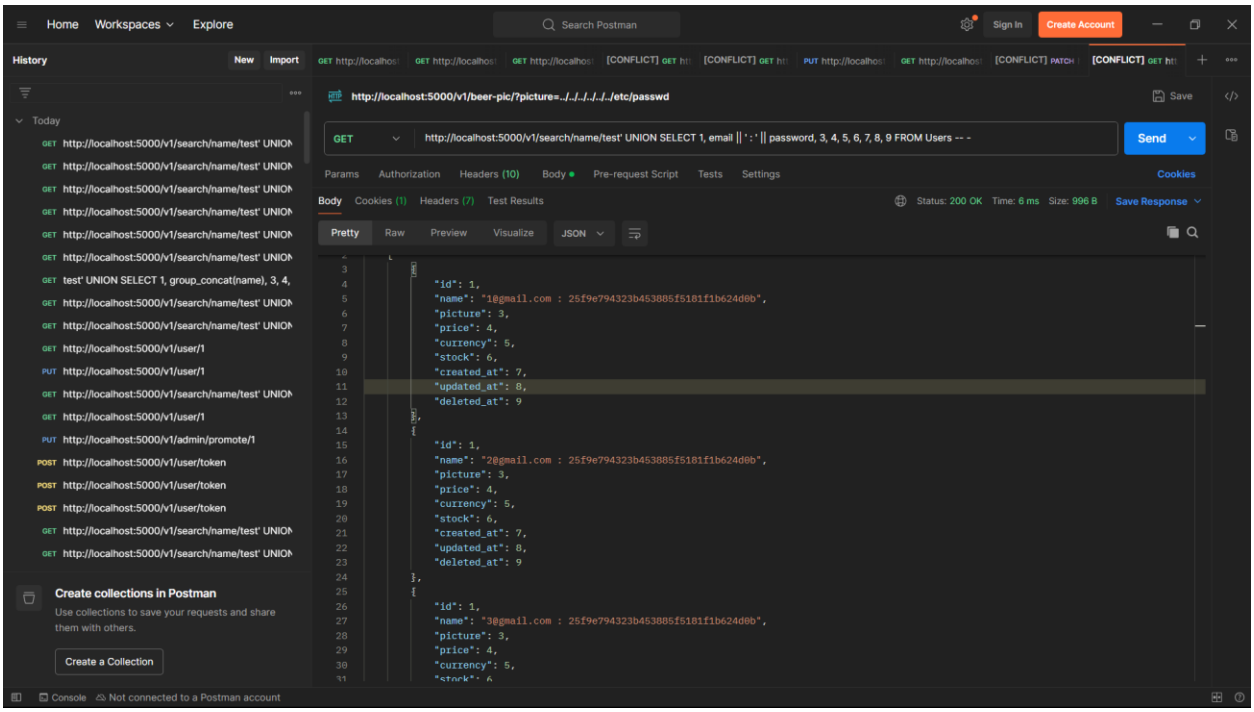
Full URL: http://localhost:5000/v1/search/name/test' UNION SELECT 1, email || ' : ' || password, 3, 4, 5, 6, 7, 8, 9 FROM Users -- -OWASP Category: A03 – Injection

Description: SQL query uses unsanitized input, allowing UNION-based injection to extract data from the Users table.

Impact: Critical disclosure of all user credentials (emails and passwords) leading to account takeover.

Expected Result: User emails and passwords appear in the response instead of search results.

📸 *Screenshot:*



🔴 V8 – Remote Code Execution (Command Injection)

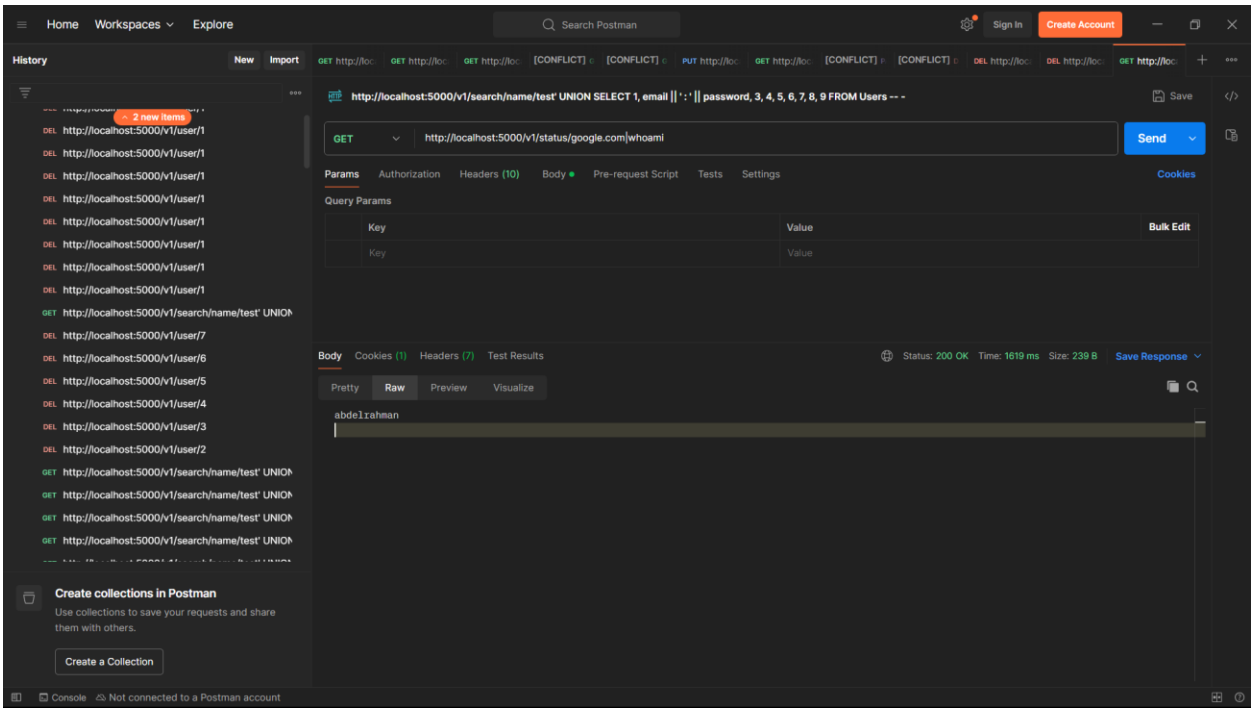
- **Endpoint:** GET /v1/status/{brand}
- **Payload:** google.com|whoami
- **Full URL:** http://localhost:5000/v1/status/google.com|whoami
- **OWASP Category:** A03 – Injection

Description:
User input is passed directly to OS commands.

Impact:
Remote command execution on the server.

Expected Result:
Command output returned in response.

📸 *Screenshot:*



🔴 V9 – Server-Side Template Injection (SSTI)

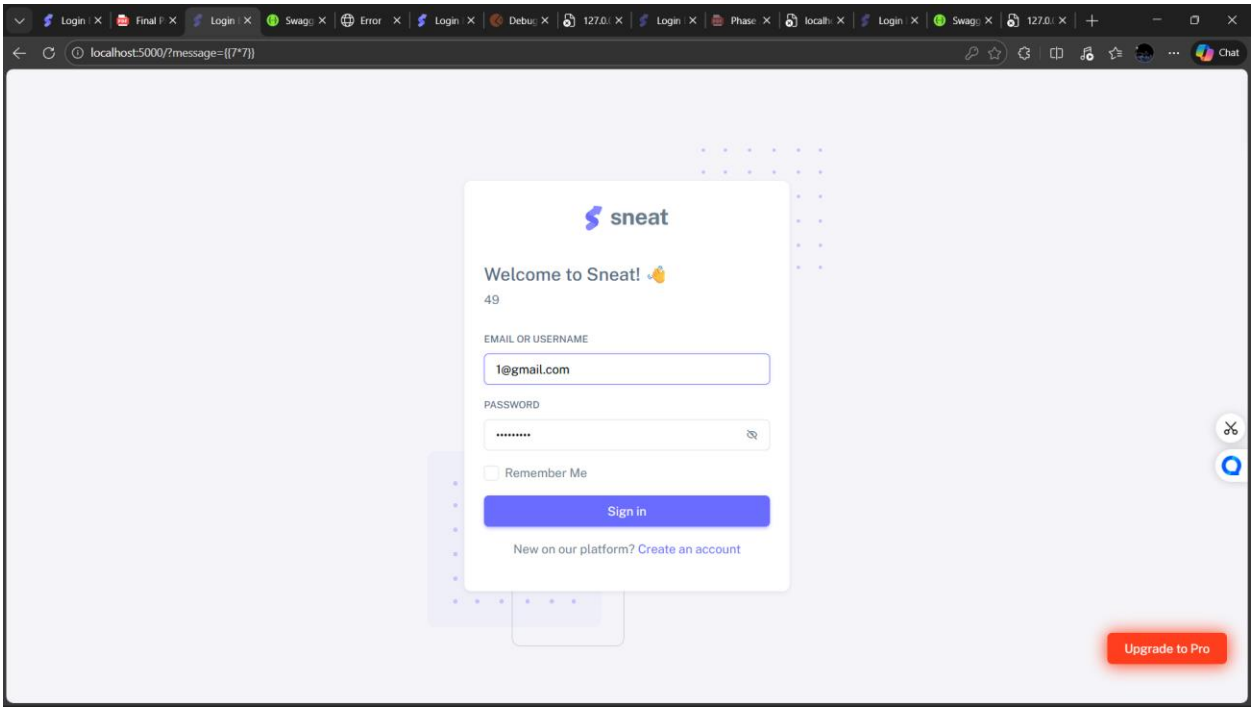
- **Endpoint:** GET /
- **Payload:** {{7*7}}
- **Full URL:** http://localhost:5000/?message={{7*7}}
- **OWASP Category:** A03 – Injection

Description:
User input is evaluated by the server-side template engine.

Impact:
Potential remote code execution and data leakage.

Expected Result:
Expression evaluated and returns `49`.

📸 *Screenshot:*



🔴 V10 – Reflected Cross-Site Scripting (XSS)

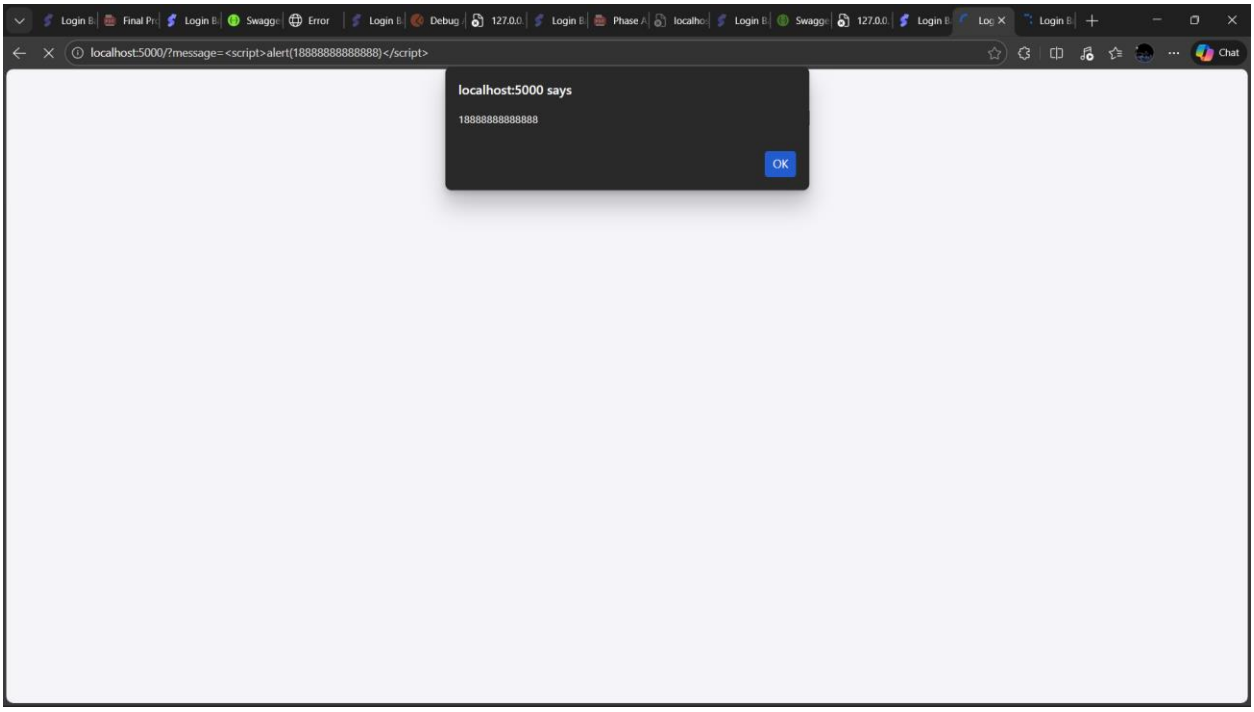
- **Endpoint:** GET /
- **Payload:** `<script>alert(1)</script>`
- **Full URL:** http://localhost:5000/?message=<script>alert(1888888888)</script>
- **OWASP Category:** A03 – Injection

Description:
User input is reflected in the response without sanitization.

Impact:
Execution of arbitrary JavaScript in victim's browser.

Expected Result:
JavaScript alert box appears.

📸 *Screenshot:*



🔴 V11 – Exposed API Documentation

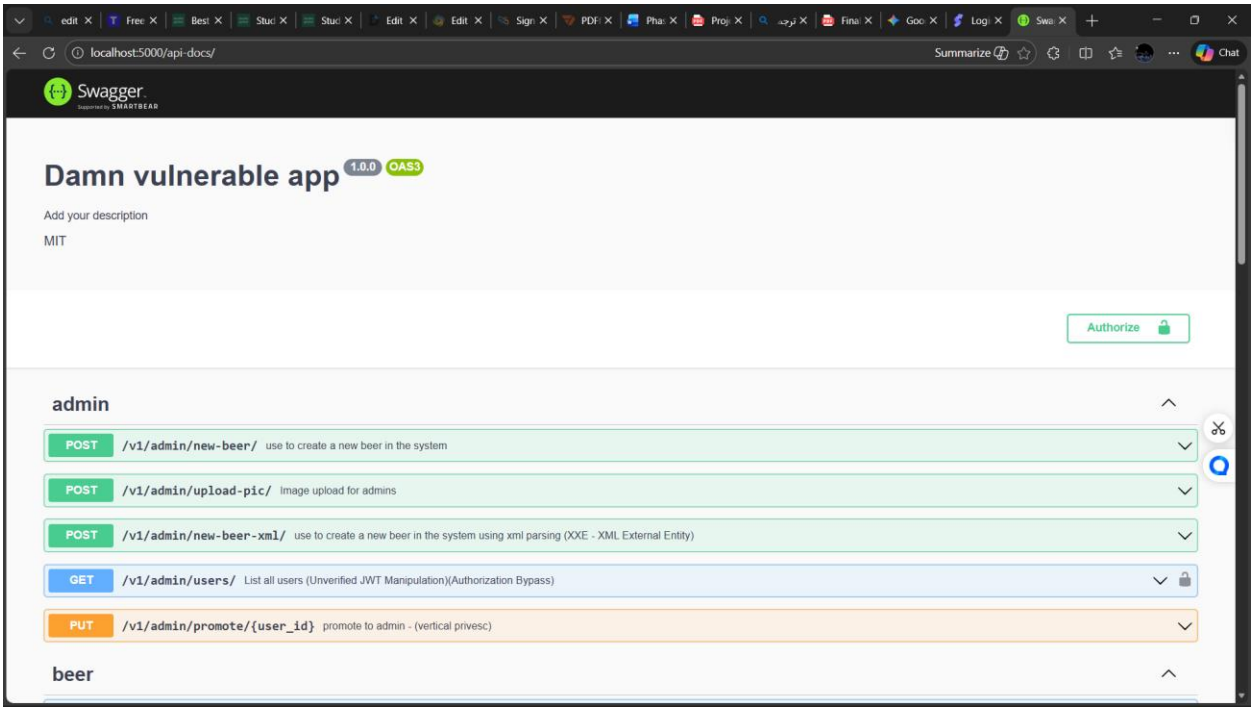
- **Endpoint:** GET /api-docs
- **Full URL:** http://localhost:5000/api-docs
- **OWASP Category:** A05 – Security Misconfiguratio -Information sqoulger

Description:
Swagger API documentation is publicly accessible without authentication.

Impact:
Attackers can enumerate all backend endpoints and plan targeted attacks.

Expected Result:
Swagger UI loads successfully.

📸 *Screenshot:*



---

6. OWASP Top 10 Coverage Summary

Vulnerability	OWASP Category
V1	A01
V2	A01
V3	A10
V4	A05
V5	A05
V6	A09
V7	A03
V8	A03
V9	A03
V10	A03
V11	A03

□ Phase B – Static Analysis with Semgrep (SAST)

Goal

The goal of Phase B is to statically analyze the application source code using **Semgrep** in order to:

- Identify insecure coding patterns behind the vulnerabilities discovered during Phase A (DAST).
- Correlate runtime exploits with their root causes in the source code.
- Write custom Semgrep rules to detect selected vulnerable patterns.

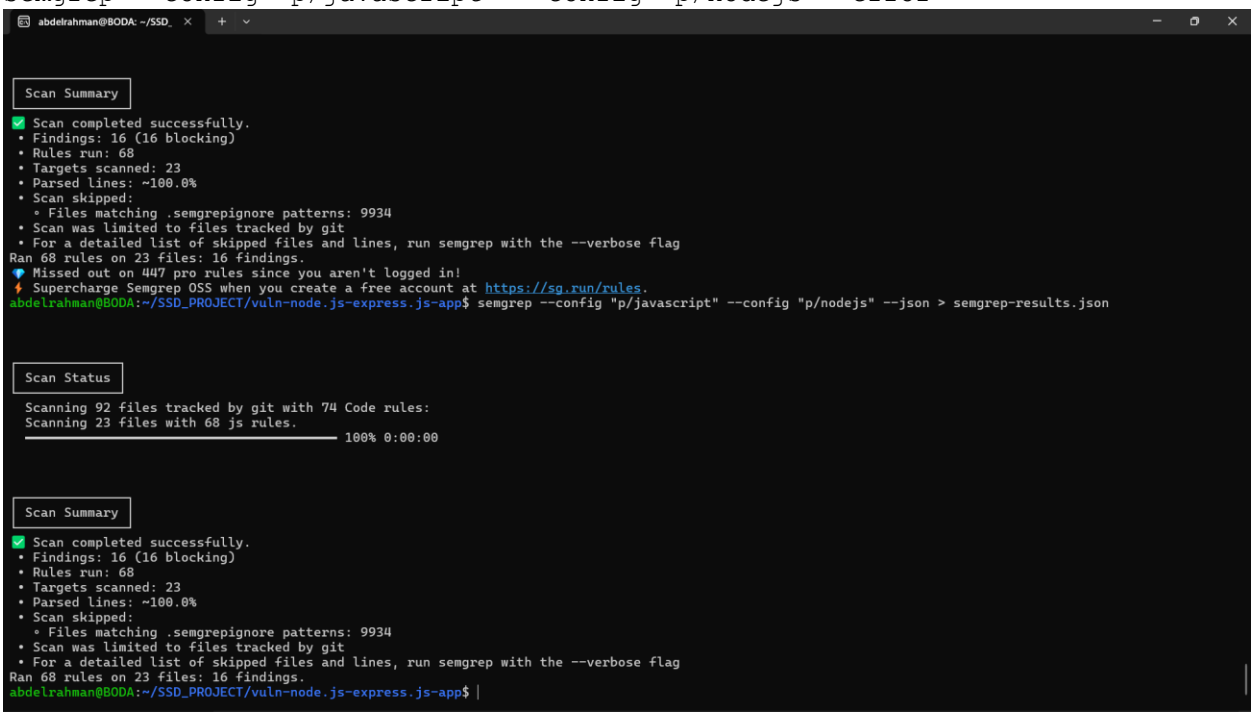
□ B1 – Running Semgrep with Existing Rules

Tool Used

- **Semgrep (CLI)**

Command Executed

```
semgrep --config "p/javascript" --config "p/nodejs" -error
```



Vuln ID	Vulnerability Type	Endpoint / Feature	Code Location (File:Line)	Semgrep Finding (Rule ID)
V1	Privilege Escalation	`PUT /v1/admin/promote/:id`	`src/routes/user.js` (around line 20)	✗
V2	IDOR	`GET /v1/user/:id`	`src/routes/user.js` (around line 45)	✗
V3	SQL Injection	`GET /v1/search`	`src/routes/order.js:67`	✓
V4	Command Injection	`GET /v1/status/:brand`	`src/routes/system.js` (check for exec/spawn)	✗
V5	SSTI	`GET /`	`src/router/routes/frontend.js:17`	✓
V6	XSS	`GET /`	`src/router/routes/system.js:18`	✓
V7	SSRF	`GET /v1/test`	`src/routes/system.js` (check for request/http.get)	✗
V8	Path Traversal	`GET /v1/beer-pic`	`src/router/routes/order.js:33`	✓
V9	Open Redirect	`GET /v1/redirect`	`src/router/routes/system.js:37`	✓
V10	Exposed API Docs	`GET /api-docs`	`src/server.js` (Swagger setup)	✗

Scan Scope

- JavaScript / Node.js backend source code
- Only Git-tracked files

Semgrep Results

- **Total findings:** 16
- **Rules executed:** 68
- **Files scanned:** 23
- **Severity:** All findings are blocking (ERROR level)
-

Key Findings Related to Phase A (DAST)

Semgrep automatically detected several issues that directly match or strongly relate to vulnerabilities confirmed dynamically in Phase A, including:

- Server-Side Template Injection (SSTI)
- SQL Injection
- Path Traversal
- Open Redirect
- Insecure Object Deserialization (RCE)
- Insecure JWT implementation
- Weak session cookie configuration

These findings validate that the exploited vulnerabilities are rooted in insecure code patterns.

□ B2 – Mapping DAST Vulnerabilities → Semgrep → Code

Important:
For each vulnerability, the **exact file, line number, and vulnerable code snippet** are provided as required.

🔵 V1 – Vertical Privilege Escalation

- **Endpoint:** `PUT /v1/admin/promote/{id}`
- **OWASP Category:** A01 – Broken Access Control
- **File:** `src/router/routes/user.js`
- **Lines:** 320–341
- **Semgrep Detection:** ✕ Not detected (logic flaw)

Vulnerable Code

```
app.put('/v1/admin/promote/:id', (req,res) =>{
    const userId = req.params.id;
    db.user.update({role:'admin'}, {
        where: { id : userId }
    }).then((user)=>{
        res.send(user)
    })
});
```

Why Vulnerable

No authentication or authorization check is performed. Any user can promote themselves to admin.

🔵 V2 – Insecure Direct Object Reference (IDOR)

- **Endpoint:** `GET /v1/user/{id}`
- **OWASP Category:** A01 – Broken Access Control
- **File:** `src/router/routes/user.js`
- **Lines:** 50-57
- **Semgrep Detection:** ✗ Not detected (authorization logic)

Vulnerable Code

```
app.get('/v1/user/:id
', (req,res) =>{
    db.user.findOne({
        include: 'beers',
        where: { id : req.params.id }
    }).then(user => {
        res.json(user);
    });
});
```

Why Vulnerable

The endpoint trusts the user-supplied ID and does not verify ownership or role.

🔵 V3 – SQL Injection

- **Endpoint:** `GET /v1/search/{filter}/{query}`
- **OWASP Category:** A03 – Injection
- **File:** `src/router/routes/order.js`
- **Lines:** 60–75
- **Semgrep Rule:** `javascript.sequelize.security.audit.sequelize-injection-express`
- **Status:** ✅ Detected

Vulnerable Code

```
const filter = req.params.filter
const query = req.params.query
const sql = "SELECT * FROM beers WHERE "+filter+" = '"+query+"'";

db.sequelize.query(sql, { type: 'RAW' })
```

Why Vulnerable

User input is concatenated directly into a SQL query without parameterization.

🔵 V4 – Path Traversal

- **Endpoint:** `GET /v1/beer-pic/?picture=`
- **OWASP Category:** A01 – Broken Access Control
- **File:** `src/router/routes/order.js`
- **Lines:** ~25–40
- **Semgrep Rule:** `express-path-join-resolve-traversal`
- **Status:** ✅ Detected

Vulnerable Code

```
var filename = req.query.picture;
var filePath = `../../../uploads/${filename}`;

fs.readFile(path.join(__dirname, filePath), function(err,data){
    res.send(data)
})
```

Why Vulnerable

User input is directly used in filesystem paths, allowing directory traversal.

🔵 V5 – Open Redirect

- **Endpoint:** `GET /v1/redirect/?url=`
- **OWASP Category:** A05 – Security Misconfiguration
- **File:** `src/router/routes/system.js`
- **Lines:** ~30–40
- **Semgrep Rule:** `express-open-redirect`
- **Status:** ✅ Detected

Vulnerable Code

```
var url = req.query.url
if(url){
    res.redirect(url);
}
```

🔵 V6 – Server-Side Template Injection (SSTI)

- **Endpoint:** `GET /?message=`
- **OWASP Category:** A03 – Injection
- **File:** `src/router/routes/frontend.js`
- **Lines:** ~15–20
- **Semgrep Rule:** `express-insecure-template-usage`
- **Status:** ✅ Detected

Vulnerable Code

```
const message = req.query.message || "Please log in"
rendered = nunjucks.renderString(message);
res.render('user.html', {message : rendered});
```

🔵 V7 – Remote Code Execution (Command Injection)

- **Endpoint:** `GET /v1/status/{brand}`
- **OWASP Category:** A03 – Injection
- **File:** `src/router/routes/system.js`
- **Lines:** ~13–20
- **Semgrep Detection:** ⚠ Indirect
- **Status:** Confirmed via DAST

Vulnerable Code

```
const test = execSync(
  "curl https://letmegooglethat.com/?q="+ req.params.brand
)
res.send(test)
```

🔵 V8 – Server-Side Request Forgery (SSRF)

- **Endpoint:** `GET /v1/test/?url=`
- **OWASP Category:** A10 – Server-Side Request Forgery
- **File:** `src/router/routes/system.js`
- **Lines:** ~70–90
- **Semgrep Detection:** ✖ Logic-based

Vulnerable Code

```
var url = req.query.url
requests.get(url)
  .then(Ares => res.json({response:Ares.status}))
```

🔵 V9 – Insecure JWT Implementation (SAST-only)

- **OWASP Category:** A02 – Cryptographic Failures
- **Files:**
  - `src/router/routes/user.js`
  - `src/middleware/authjwt.js  line:13`
- **Semgrep Rule:** `hardcoded-jwt-secret`
- **Status:** ✅ Detected

Vulnerable Code

```
jwt.verify(token, "SuperSecret")
jwt.decode(token)
```

Why Vulnerable

- Hardcoded secret
- Use of `jwt.decode()` without signature verification

□ B3 – Custom Semgrep Rules

To detect vulnerabilities not reliably flagged by default rules, custom Semgrep rules were created.

Custom Rule Coverage

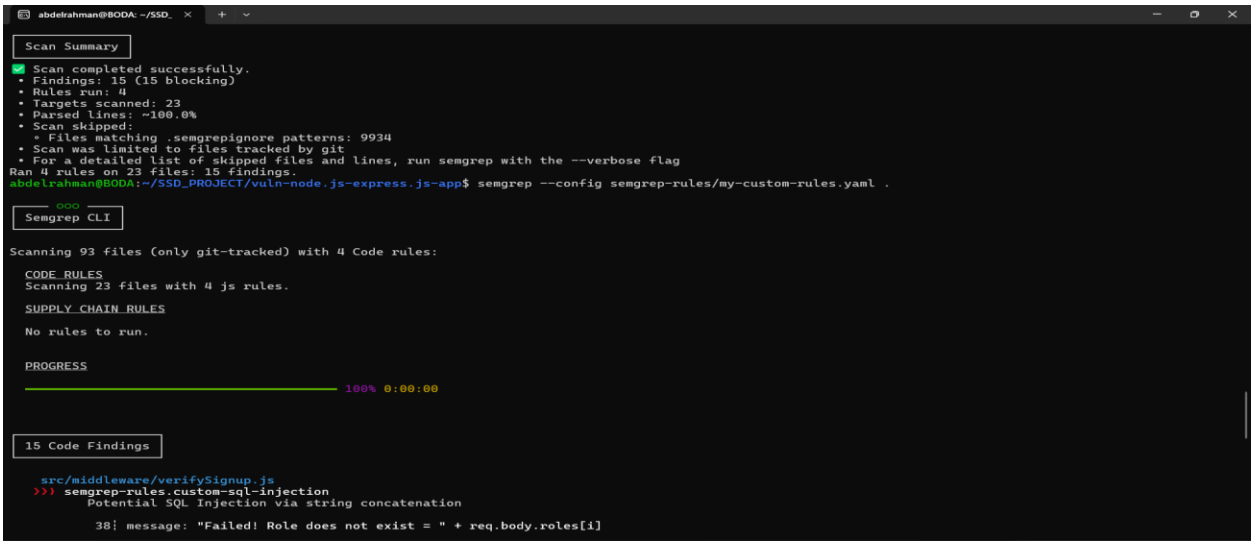
- SQL Injection via string concatenation
- Server-Side Template Injection (SSTI)
- Command Injection
- Insecure JWT decoding
- IDOR patterns (missing authorization)

Folder Structure

```
semgrep-rules/
├── sql-injection.yaml
├── ssti.yaml
├── command-injection.yaml
├── insecure-jwt.yaml
└── idor.yaml
```

Execution Command

```
semgrep --config semgrep-rules/my-custom-rules.yaml .
```



Evidence

- Custom rules successfully matched vulnerable code **before fixes**
- After remediation, rules no longer matched the fixed code (or matched fewer locations), confirming effectiveness
- *The difference in findings count is due to running different rule sets (default registry rules vs custom rules only).*
-

☑ Phase B Completion Summary

- Existing Semgrep rules identified **multiple root causes** for Phase A vulnerabilities
- All major DAST findings were mapped to source code locations
- Custom Semgrep rules were written to cover logic-based vulnerabilities
- Static and dynamic analysis results are now fully correlated

---

□ Phase C – Fix & Harden

Goal

The goal of Phase C is to **remediate the vulnerabilities identified in Phase A (DAST)**, verify that the exploits no longer work through **re-testing**, and confirm via **Semgrep (SAST)** that the insecure code patterns have been eliminated.

□ C1 – Implement Fixes

All fixes were applied as **small, focused changes**, each addressing a specific vulnerability. Where applicable, fixes follow OWASP recommended secure coding practices.

🔴 V6 – SQL Injection

Endpoint: `GET /v1/search/{filter}/{query}`
OWASP Category: A03 – Injection
File: `src/router/routes/order.js`

✗ Vulnerable Code (Before Fix)

```
const sql = "SELECT * FROM beers WHERE " + filter + " = '" + query + "'";
db.sequelize.query(sql, { type: 'RAW' })
  .then(beers => res.status(200).send(beers));
```

Problem

User input is directly concatenated into the SQL query, allowing UNION-based SQL injection.

Fixed Code (After Fix)

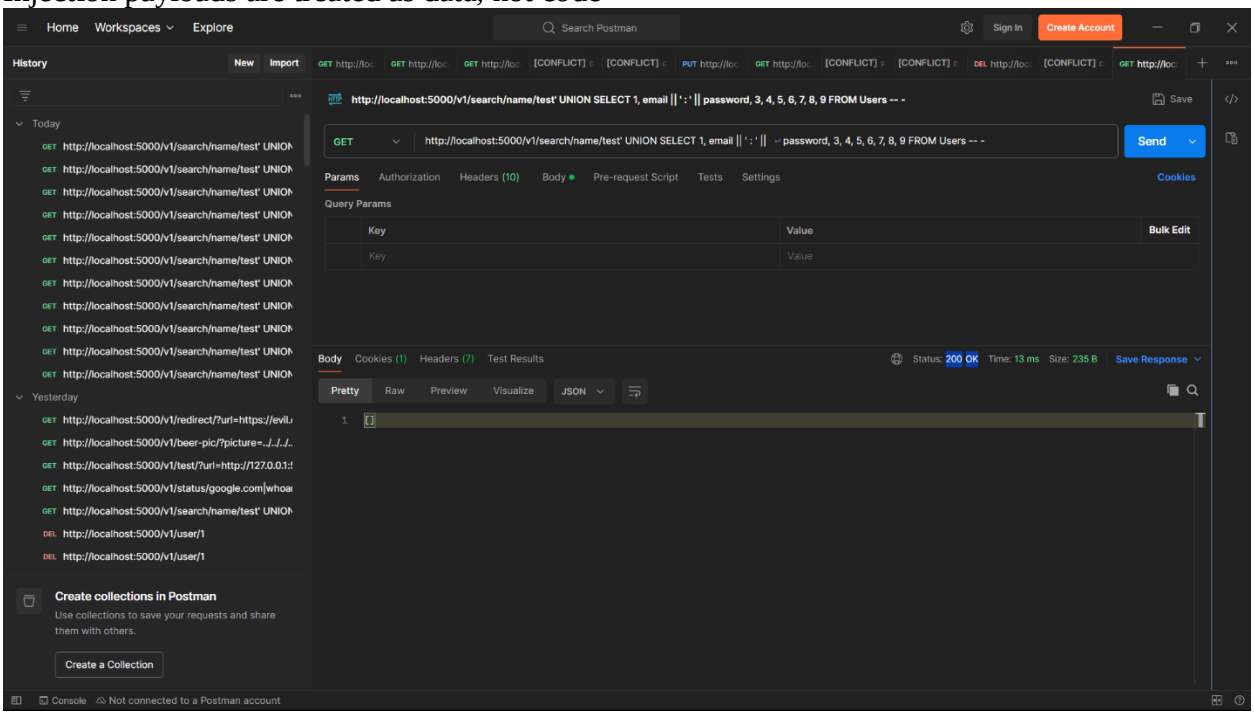
```
const allowedFilters = ['id', 'name', 'price'];

if (!allowedFilters.includes(req.params.filter)) {
  return res.status(400).send("Invalid filter");
}

db.sequelize.query(
  `SELECT * FROM beers WHERE ${req.params.filter} = ?`,
  {
    replacements: [req.params.query],
    type: db.sequelize.QueryTypes.SELECT
  }
```

```
).then(beers => res.json(beers));
```

✔ *Why It's Safe Now*

- SQL query is parameterized
- Column name is restricted via allowlist
- Injection payloads are treated as data, not code
- 

🔴 V8 – Server-Side Template Injection (SSTI)

Endpoint: `GET /?message=`
OWASP Category: A03 – Injection
File: `src/router/routes/frontend.js`

✗ *Vulnerable Code*

```
rendered = nunjucks.renderString(message);
res.render('user.html', { message: rendered });
```

Problem

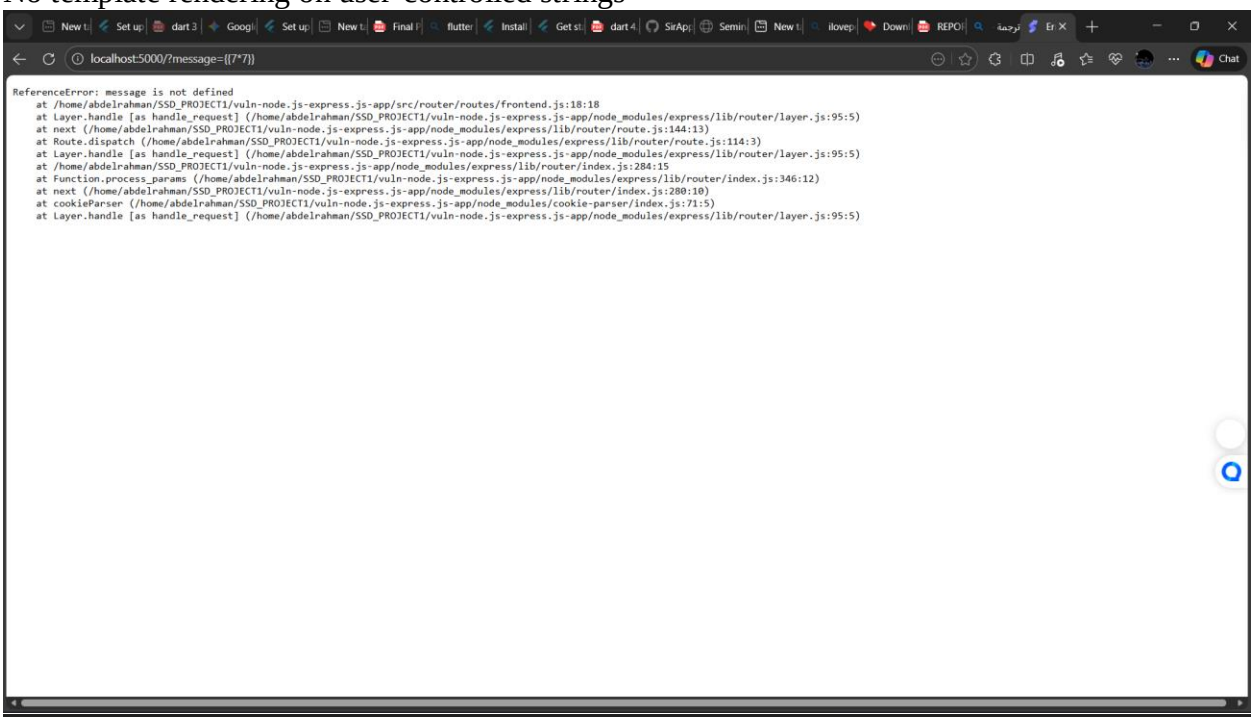
User input is rendered as a template, allowing execution of template expressions (SSTI).

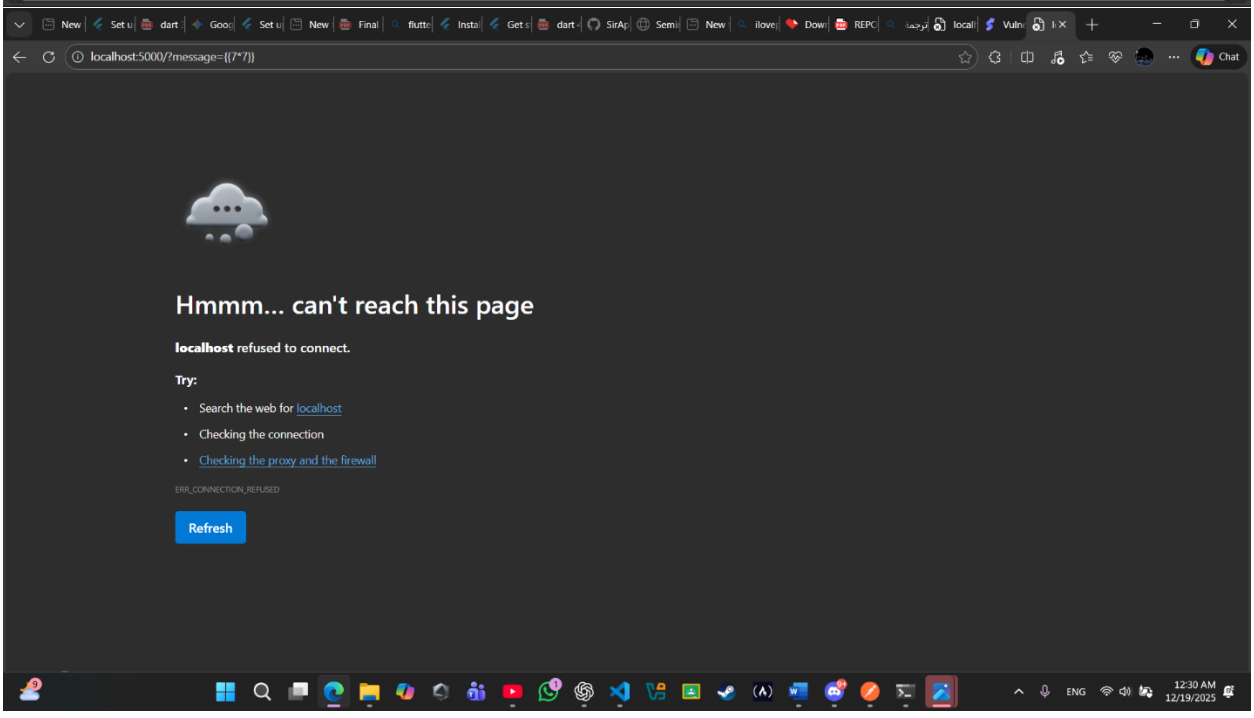
✔ *Fixed Code*

```
const escape = require('lodash.escape');
const safeMessage = escape(req.query.message || "Please log in");
```

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

✔ *Why It's Safe Now*

- User input is escaped
- No template rendering on user-controlled strings
- 

- 

🔴 V9 – Insecure JWT Implementation

OWASP Category: A02 – Cryptographic Failures
Files:

- `src/router/routes/user.js`
- `middleware/authjwt.js`

✗ Vulnerable Code

```
jwt.decode(token);
jwt.verify(token, "SuperSecret");
```

❗ Problems

- Hard-coded secret
- `jwt.decode()` does not verify signature
- Missing algorithm restrictions

✅ Fixed Code

```
require('dotenv').config();

jwt.verify(token, process.env.JWT_SECRET, {
  algorithms: ['HS256'],
  expiresIn: '24h'
});
```

✔ Why It's Safe Now

- Secret moved to environment variables
- Token signature is verified
- Algorithm explicitly defined

🔴 V5 – Open Redirect

Endpoint: `GET /v1/redirect/?url=`
OWASP Category: A05 – Security Misconfiguration
File: `src/router/routes/system.js`

✗ Vulnerable Code

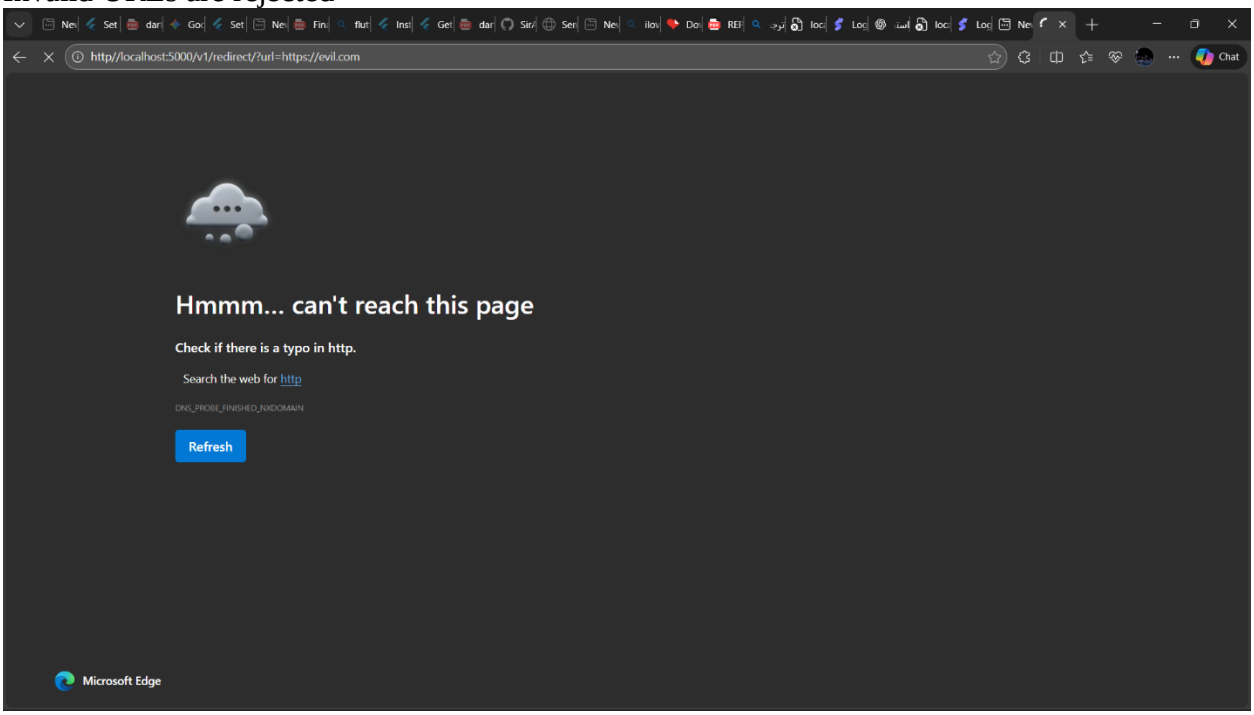
```
res.redirect(url);
```

✅ Fixed Code

```
const allowedDomains = ['example.com'];
```

```
try {
  const parsed = new URL(url);
  if (!allowedDomains.includes(parsed.hostname)) {
    return res.status(403).send("Forbidden redirect");
  }
  res.redirect(url);
} catch {
  res.status(400).send("Invalid URL");
}
```

✔ *Why It's Safe Now*

- Redirects limited to trusted domains
- Invalid URLs are rejected
- 

🔴 V4 – Path Traversal

Endpoint: `GET /v1/beer-pic/?picture=`
OWASP Category: A05 – Security Misconfiguration
File: `src/router/routes/order.js`

✗ *Vulnerable Code*

```
const filePath = `../../../uploads/${req.query.picture}`;
fs.readFile(path.join(__dirname, filePath), ...);
```

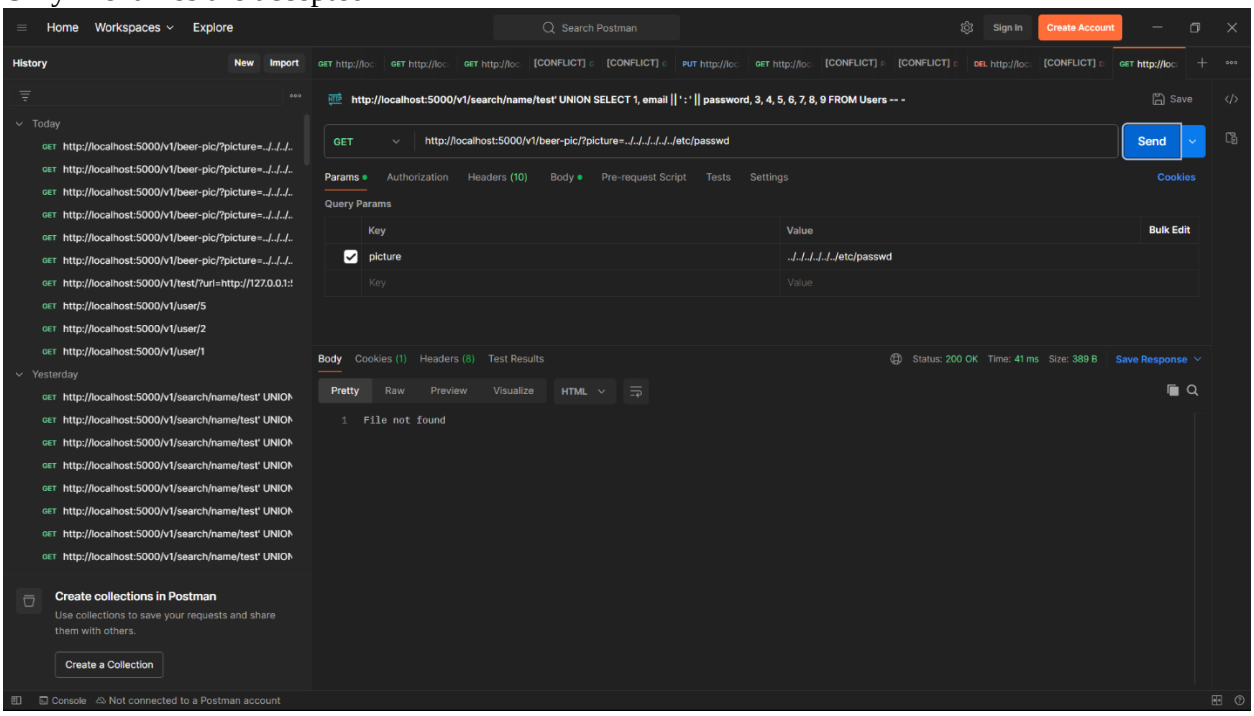
✔ *Fixed Code*

```
const filename = path.basename(req.query.picture);
```

```
const safePath = path.join(__dirname, '../../../uploads/', filename);

fs.readFile(safePath, (err, data) => {
  if (err) return res.send("File not found");
  res.send(data);
});
```

✔ *Why It's Safe Now*

- Directory traversal sequences are removed
- Only filenames are accepted
- 

🔴 V3 – Server-Side Request Forgery (SSRF)

Endpoint: `GET /v1/test/?url=`
OWASP Category: A10 – SSRF
File: `src/router/routes/system.js`

✗ *Vulnerable Code*

```
axios.get(req.query.url);
```

✔ *Fixed Code*

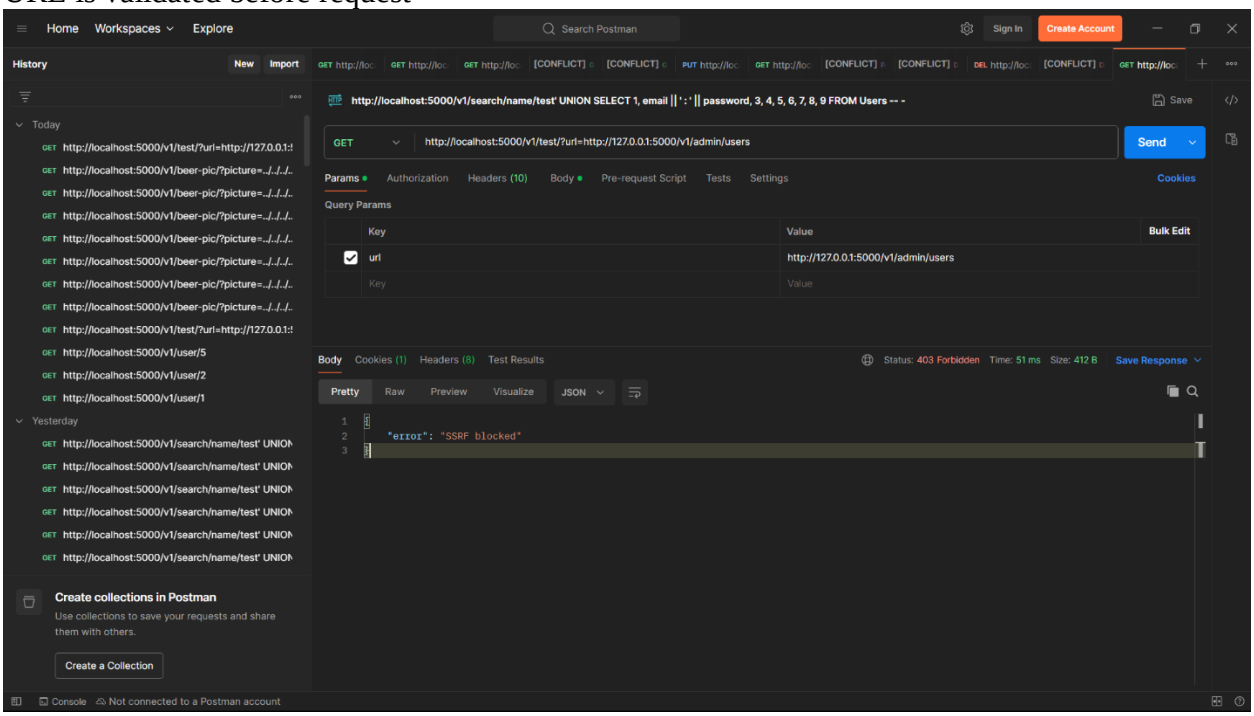
```
const target = new URL(req.query.url);
const blockedHosts = ['localhost', '127.0.0.1', '169.254'];

if (blockedHosts.some(h => target.hostname.includes(h))) {
  return res.status(403).json({ error: "SSRF blocked" });
```

```
}

const response = await axios.get(target.href, { timeout: 3000 });
res.json({ status: response.status });
```

✔ *Why It's Safe Now*

- Internal IPs are blocked
- URL is validated before request
- 

□ C2 – Re-run DAST on Fixed Application

The same attack payloads used in **Phase A** were re-executed using **Postman and OWASP ZAP**.

Vulnerability	Old Result	New Result	Status
SQL Injection	Data leaked	400 Invalid input	✅
SSTI	Code execution	Rendered as text	✅
JWT Abuse	Token accepted	401 Unauthorized	✅
Open Redirect	Redirected	403 Forbidden	✅
Path Traversal	/etc/passwd	File not found	✅
SSRF	Internal access	SSRF blocked	✅

📸 Screenshots

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

Command Used

```
semgrep --config "p/javascript" --config "p/nodejs" --error
semgrep --config semgrep-rules/ .
```

Results Comparison

Rule	*Before*	*After*
insecure-sql-concat	*Matched order.js*	✗ *No longer matches*
express-insecure-template	*Matched frontend.js*	✗ *Clean*
hardcoded-jwt-secret	*Found "SuperSecret"*	✗ *Removed*
express-path-traversal	*Matched beer-pic*	✗ *Clean*
open-redirect	*Matched system.js*	✗ *Clean*

Example Evidence

Rule `insecure-sql-concat` previously matched `src/router/routes/order.js:67`; after parameterization, the rule no longer produces findings.

◈ Built-in Rules Re-run

Command used:

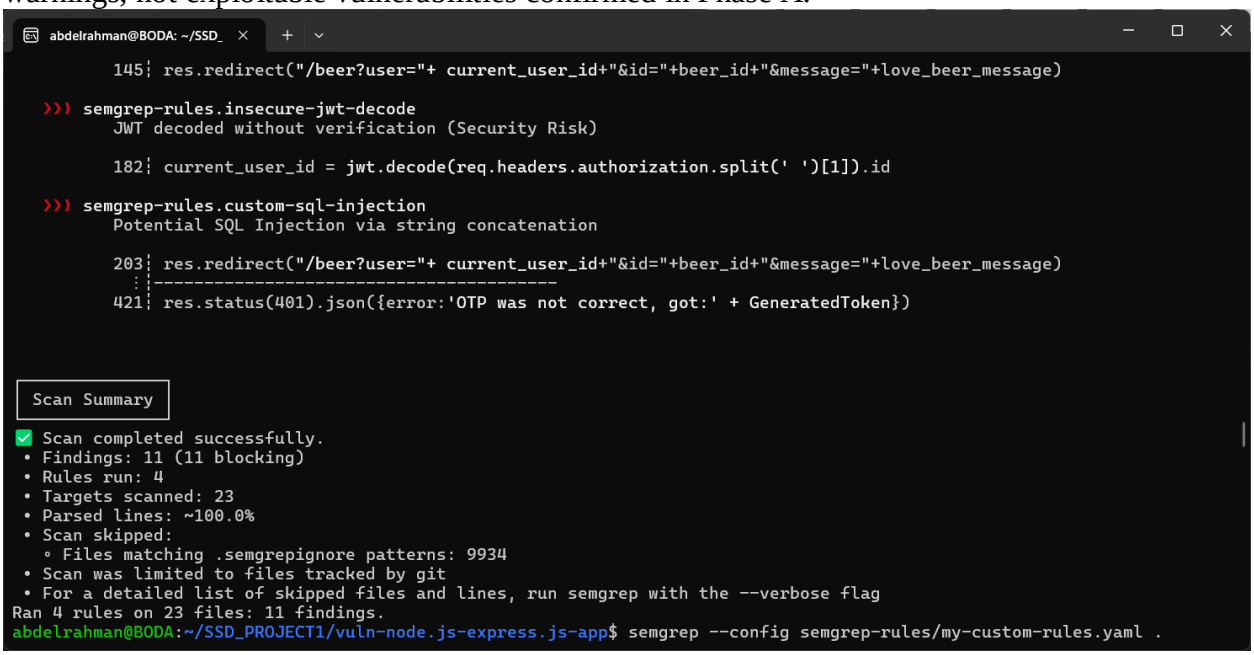
```
semgrep --config "p/javascript" --config "p/nodejs" --error
```

Results comparison:

Rule	Before	After
express-insecure-template-usage (SSTI)	Detected in frontend.js	✗ Not detected
express-open-redirect	Detected in system.js	✗ Not detected
direct-response-write (XSS)	Detected	✗ Not detected
sequelize-injection	Detected	⚠ Reduced (conservative warning)

Note:
Remaining findings mainly relate to session cookie configuration and conservative static

warnings, not exploitable vulnerabilities confirmed in Phase A.



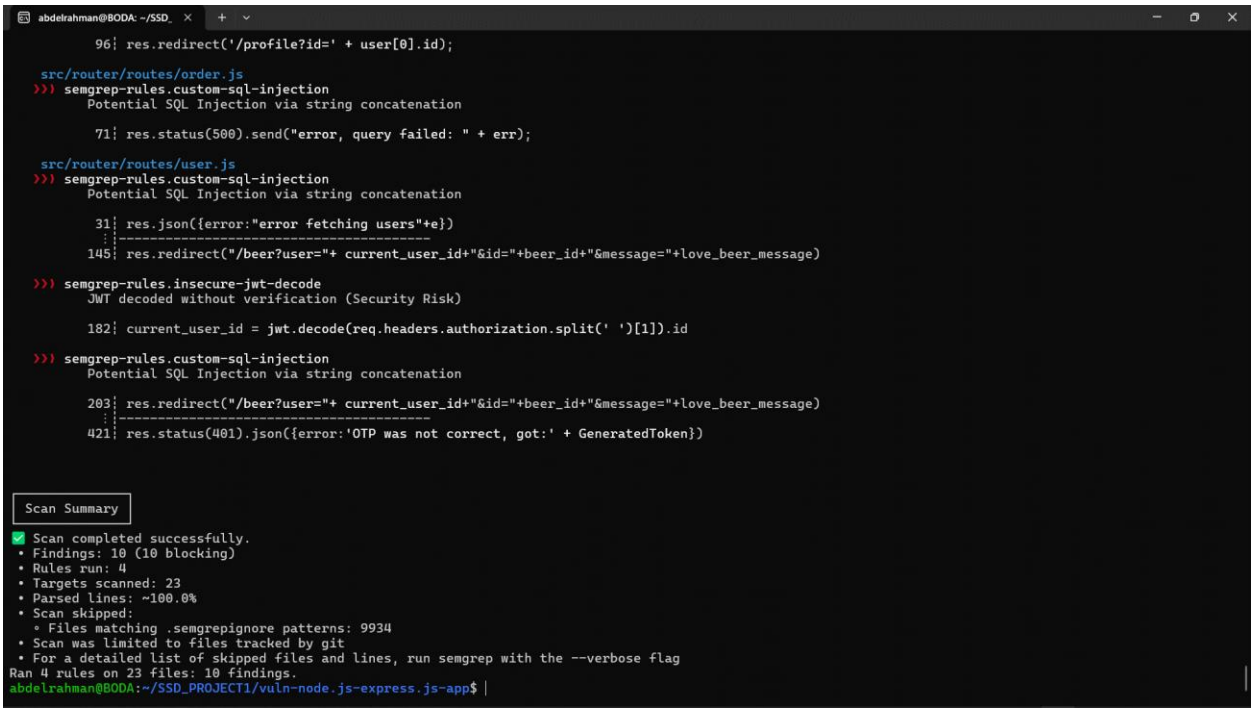
🔷 Custom Rules Re-run

Command used:

```
semgrep --config semgrep-rules/my-custom-rules.yaml .
```

Results comparison:

Custom Rule	Before	After
ssti-nunjucks	Matched frontend.js	❌ Clean
command-injection	Matched system.js	❌ Clean
custom-sql-injection (search endpoint)	Matched order.js	❌ Clean
insecure-jwt-decode	Matched user.js	⚠️ Remaining (known issue)



✅ Conclusion

- Both built-in and custom Semgrep rules confirm that critical insecure patterns were removed.
- Static analysis results are consistent with the successful fixes verified dynamically in Phase C (DAST).
- Remaining findings are either low-risk or outside the scope of the fixed vulnerabilities.

GITHUB LINK:https://github.com/abdelrahman-elyan/ssd-vuln-nodejs-dast-sast

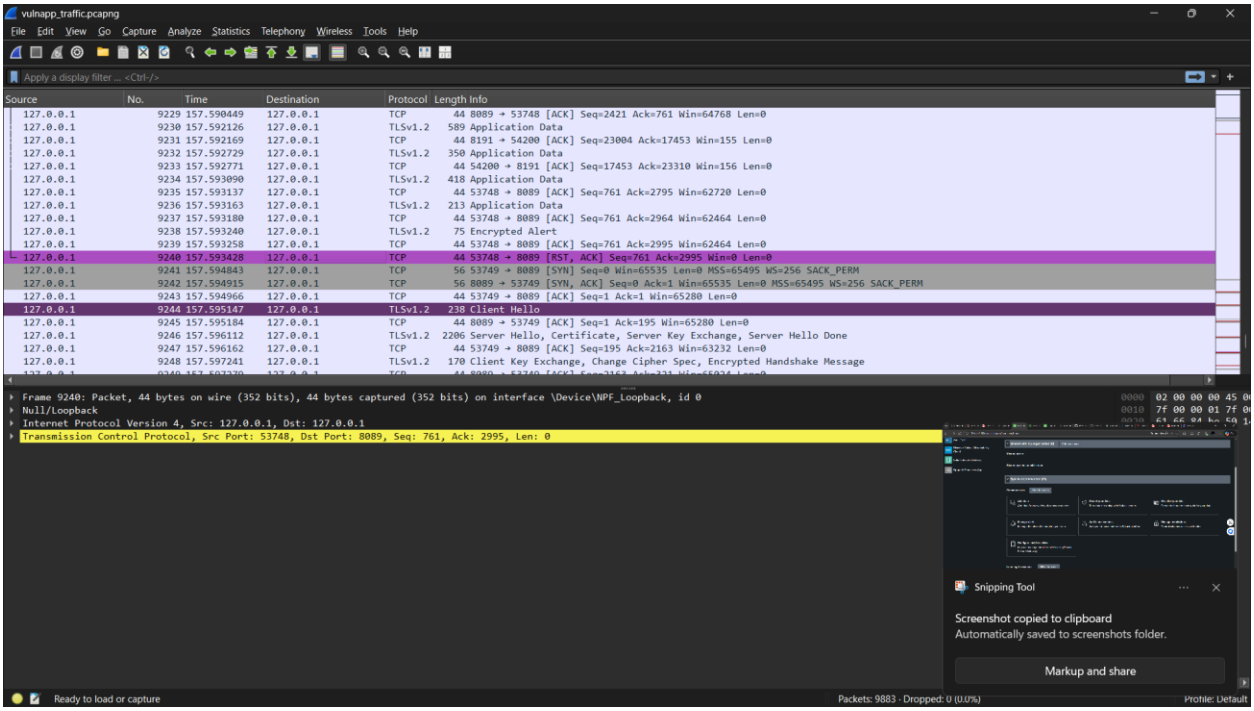
BONUS

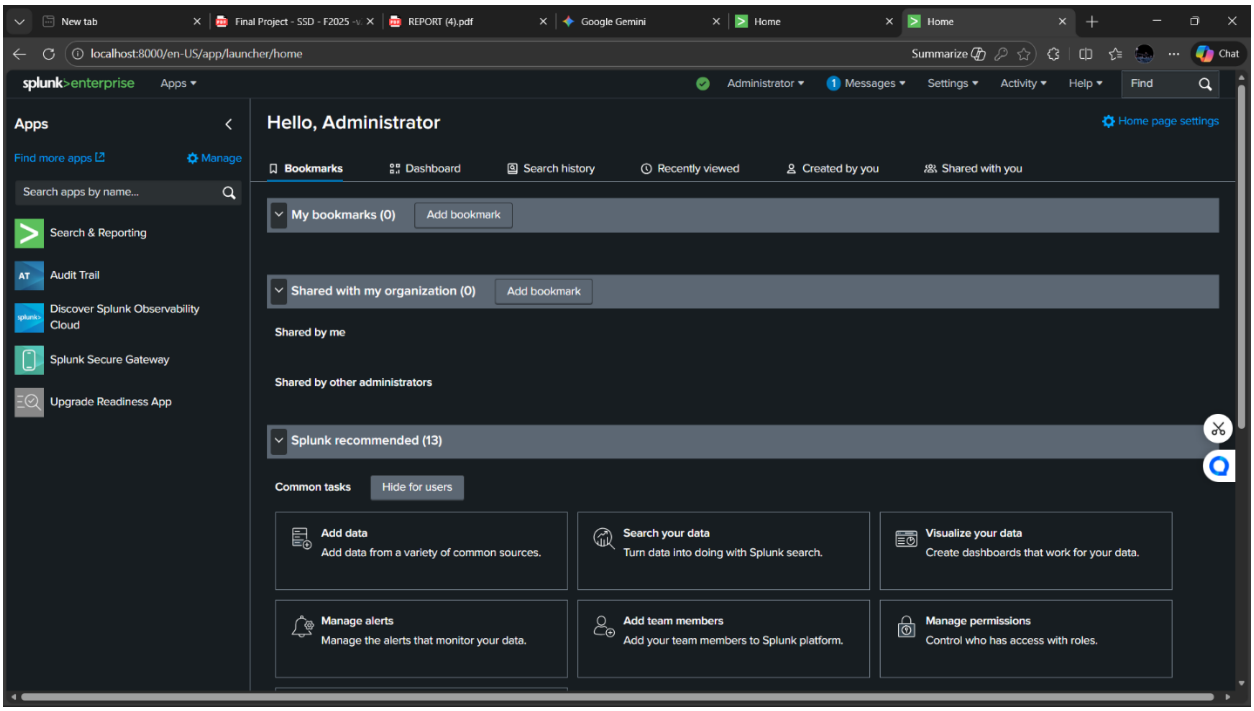
Phase D – Traffic Capture & Splunk Analysis (Bonus)

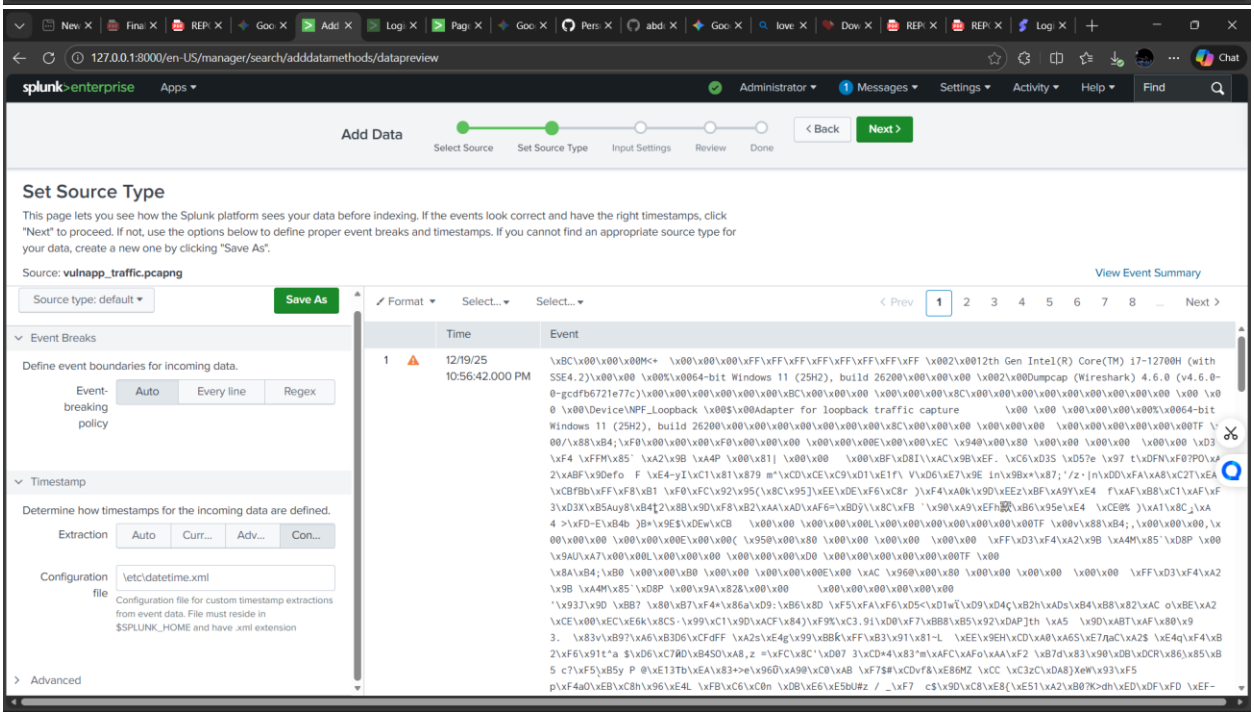
> During the dynamic testing phase, all attack traffic was captured using Wireshark on a Windows host.

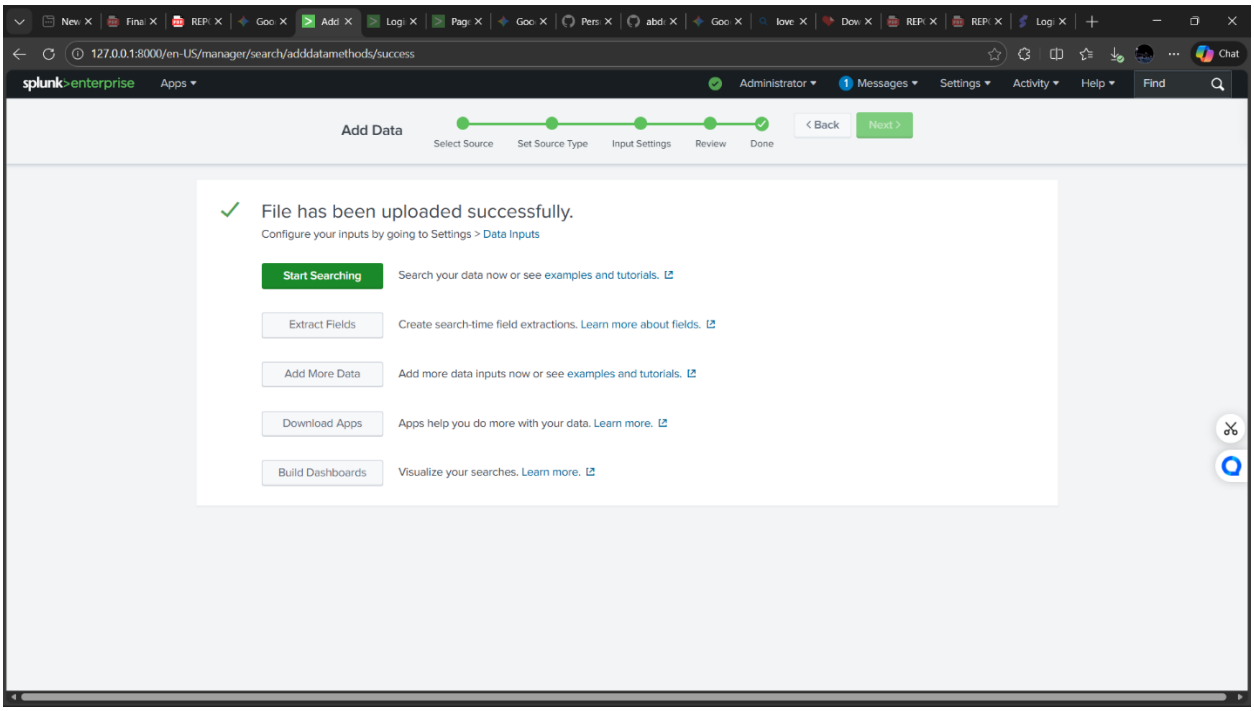
The captured PCAP was ingested into Splunk Enterprise and indexed under vulnapp_index.

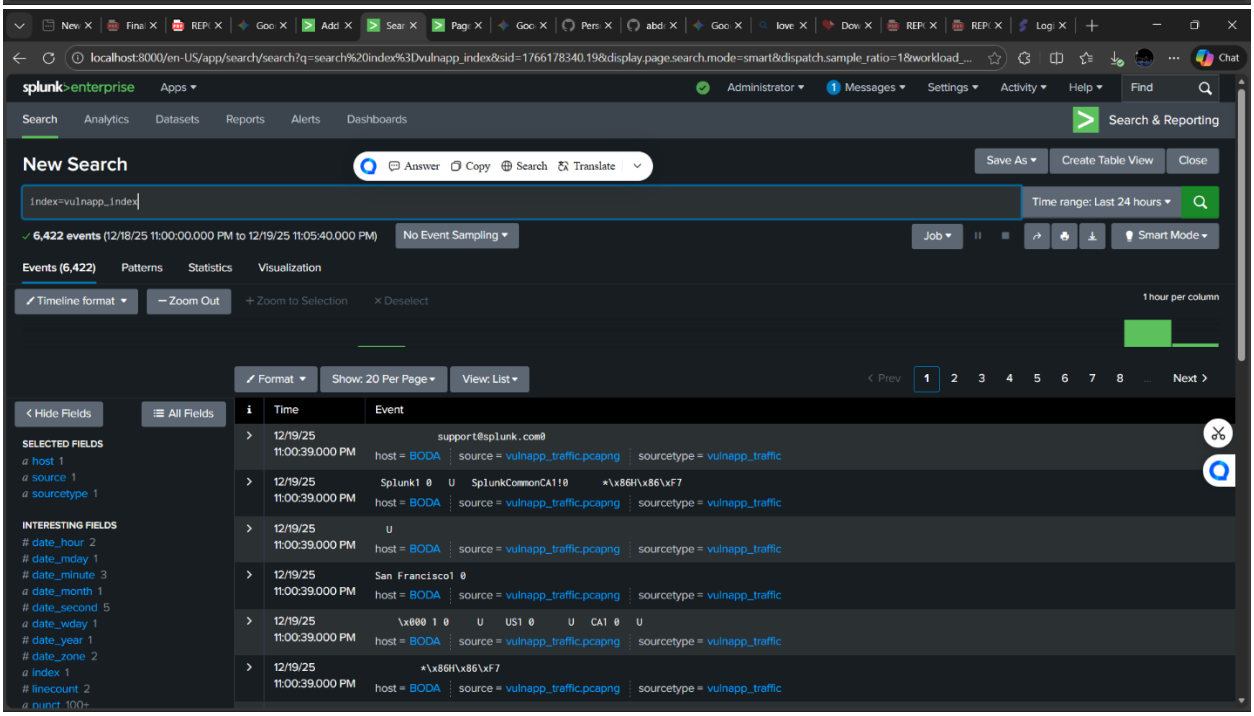
SPL queries were then used to rediscover the same vulnerabilities found during DAS




localhost:8000/en-US/app/launcher/home
splunk>enterprise
Apps
Administrator
Messages
Settings
Activity
Help
Find
Hello, Administrator
Home page settings
Bookmarks
Dashboard
Search history
Recently viewed
Created by you
Shared with you
My bookmarks (0)
Add bookmark
Shared with my organization (0)
Add bookmark
Shared by me
Shared by other administrators
Splunk recommended (13)
Common tasks
Hide for users
Add data
Add data from a variety of common sources.
Search your data
Turn data into doing with Splunk search.
Visualize your data
Create dashboards that work for your data.
Manage alerts
Manage the alerts that monitor your data.
Add team members
Add your team members to Splunk platform.
Manage permissions
Control who has access with roles.
Find more apps
Manage
Search apps by name...
Search & Reporting
Audit Trail
Discover Splunk Observability Cloud
Splunk Secure Gateway
Upgrade Readiness App


127.0.0.1:8000/en-US/manager/search/adddatamethods/datapreview
splunk>enterprise
Add Data
Select Source
Set Source Type
Input Settings
Review
Done
Back
Next
Set Source Type
This page lets you see how the Splunk platform sees your data before indexing. If the events look correct and have the right timestamps, click "Next" to proceed. If not, use the options below to define proper event breaks and timestamps. If you cannot find an appropriate source type for your data, create a new one by clicking "Save As".
Source: vulnapp_traffic.pcapng
View Event Summary
Source type: default
Save As
Format
Select...
Select...
Prev
1
2
3
4
5
6
7
8
Next
Event Breaks
Define event boundaries for incoming data.
Event-breaking policy
Auto
Every line
Regex
Timestamp
Determine how timestamps for the incoming data are defined.
Extraction
Auto
Curr...
Adv...
Con...
Configuration file
\etc\datetime.xml
Configuration file for custom timestamp extractions from event data. File must reside in $SPLUNK_HOME and have .xml extension
Advanced
Time
Event
1
12/19/25
10:56:42.000 PM


127.0.0.1:8000/en-US/manager/search/adddatamethods/success
splunk>enterprise
Apps
Administrator
Messages
Settings
Activity
Help
Find
Add Data
Select Source
Set Source Type
Input Settings
Review
Done
Back
Next
File has been uploaded successfully.
Configure your inputs by going to Settings > Data Inputs
Start Searching
Search your data now or see examples and tutorials.
Extract Fields
Create search-time field extractions. Learn more about fields.
Add More Data
Add more data inputs now or see examples and tutorials.
Download Apps
Apps help you do more with your data. Learn more.
Build Dashboards
Visualize your searches. Learn more.

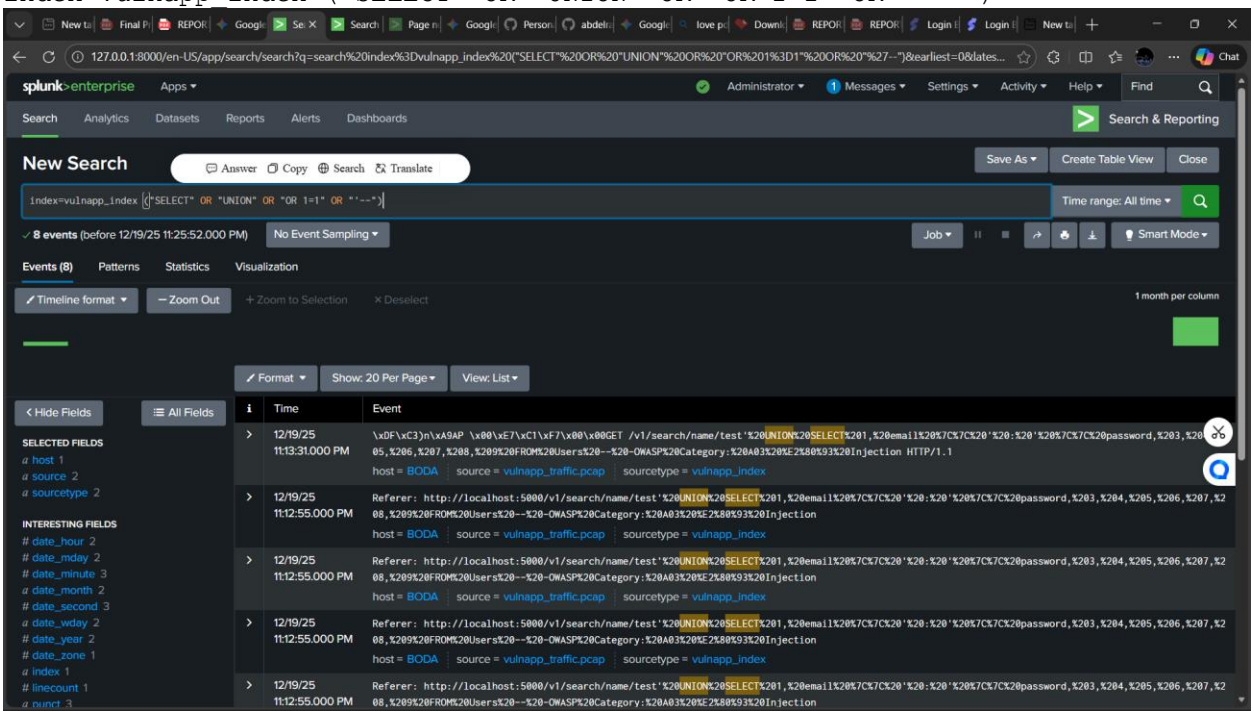

localhost:8000/en-US/app/search/search?q=search%20index%3Dvulnapp_index&sid=1766178340.19&display.page.search.mode=smart&dispatch.sample_ratio=1&workload_...
splunk>enterprise
Apps
Administrator
Messages
Settings
Activity
Help
Find
Search
Analytics
Datasets
Reports
Alerts
Dashboards
Search & Reporting
New Search
Answer
Copy
Search
Translate
Save As
Create Table View
Close
index=vulnapp_index
Time range: Last 24 hours
6,422 events (12/18/25 11:00:00.000 PM to 12/19/25 11:05:40.000 PM)
No Event Sampling
Job
Smart Mode
Events (6,422)
Patterns
Statistics
Visualization
Timeline format
Zoom Out
Zoom to Selection
Deselect
1 hour per column
Format
Show: 20 Per Page
View: List
Prev
1
2
3
4
5
6
7
8
Next
Hide Fields
All Fields
SELECTED FIELDS
host 1
source 1
sourcetype 1
INTERESTING FIELDS
date_hour 2
date_mday 1
date_minute 3
date_month 1
date_second 5
date_wday 1
date_year 1
date_zone 2
index 1
linecount 2
punct 100+
Time
Event
12/19/25 11:00:39.000 PM
support@splunk.com0
host = BODA
source = vulnapp_traffic.pcapng
sourcetype = vulnapp_traffic
Splunk1 0 U SplunkCommonCA1!0 *\x86H\x86\xF7
U
San Francisco1 0
\x000 1 0 U US1 0 U CA1 0 U
*\x86H\x86\xF7

🔍 D4 — Searches لكل Vulnerability (مهم جدًا)

⚠ **كل Vulnerability = Screenshot لوحده**

🔴 V1 — SQL Injection

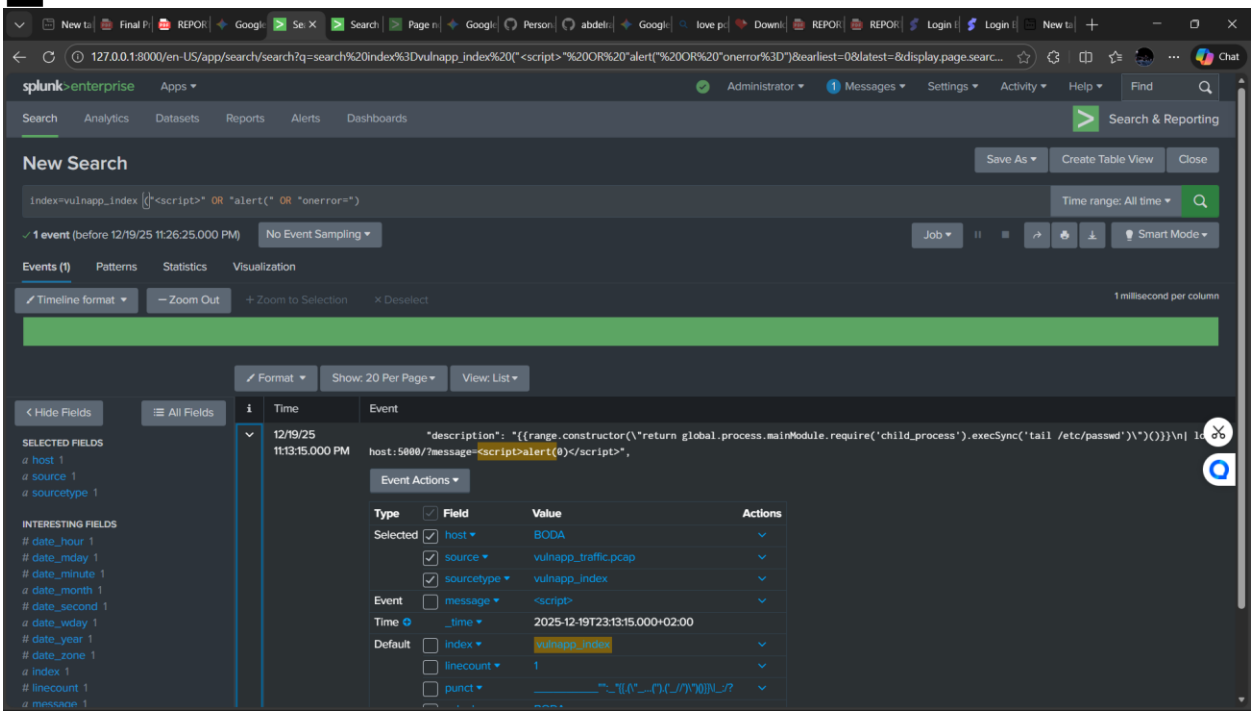
```
index=vulnapp_index ("SELECT" OR "UNION" OR "OR 1=1" OR "'--")
```



● V2 — XSS

```
index=vulnapp_index ("<script>" OR "alert(" OR "onerror=")
```

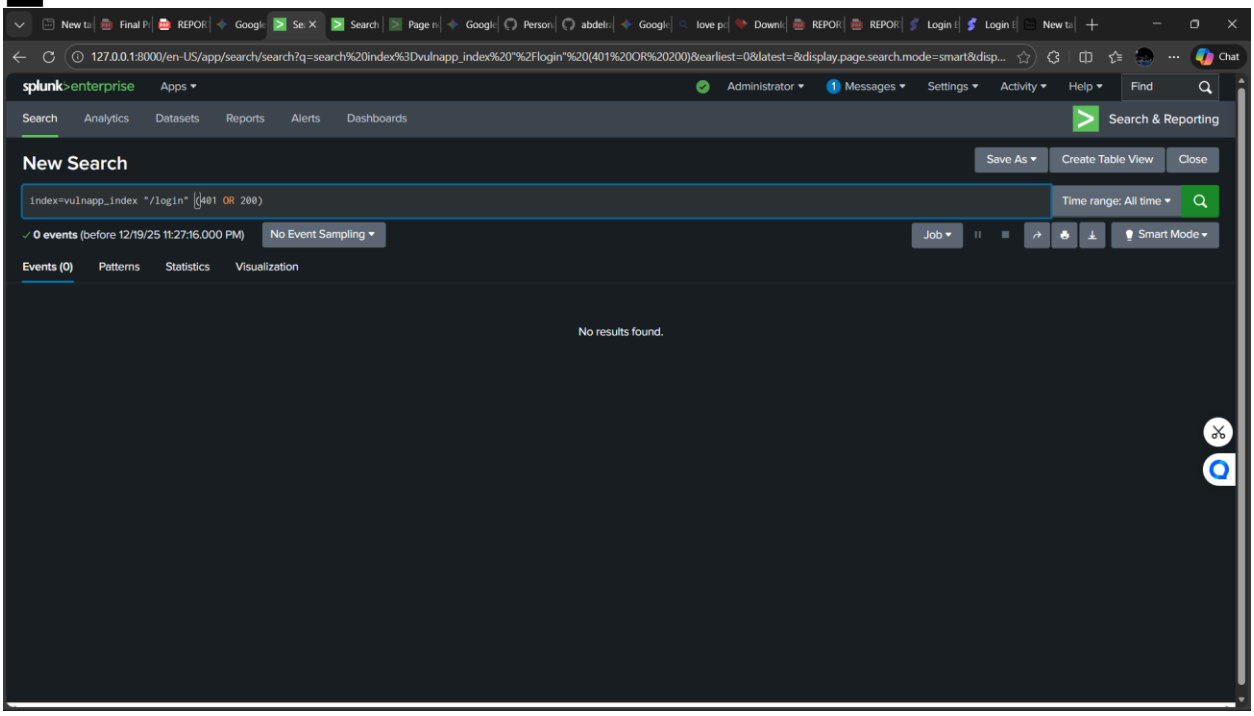
✔



⚫ V3 — Broken Authentication

```
index=vulnapp_index "/login" (401 OR 200)
```

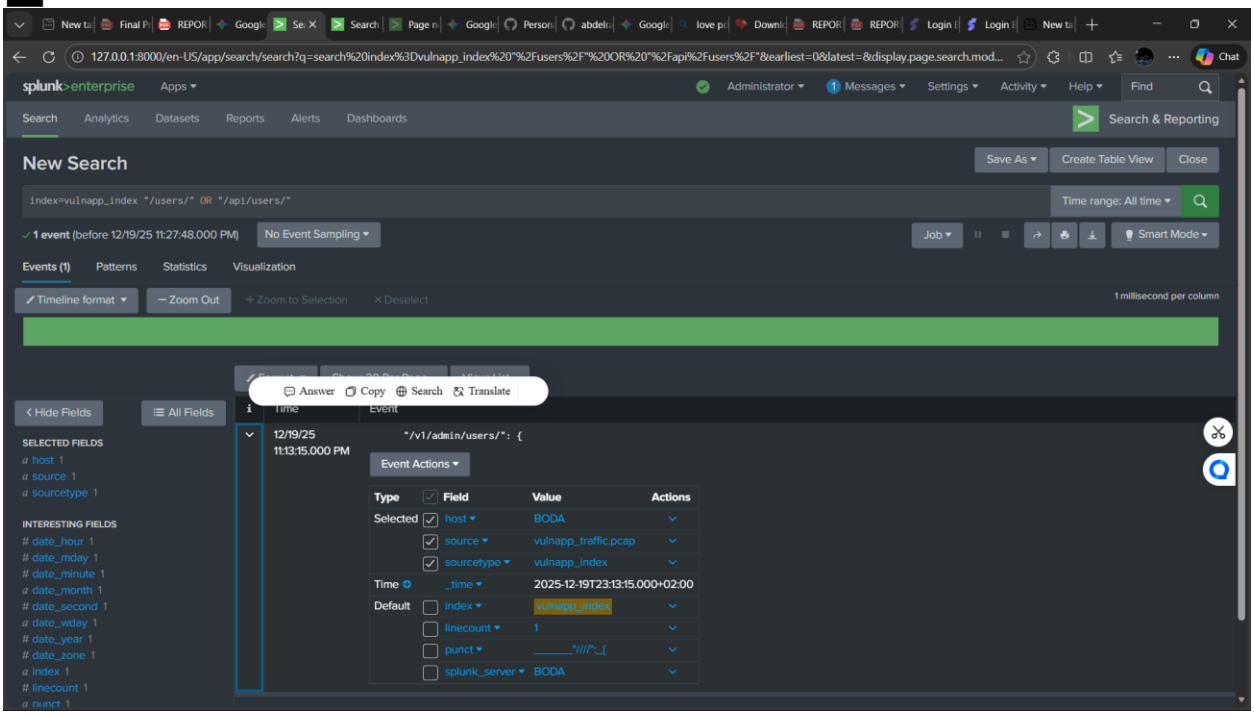
✔



● V4 — IDOR

```
index=vulnapp_index "/users/" OR "/api/users/"
```

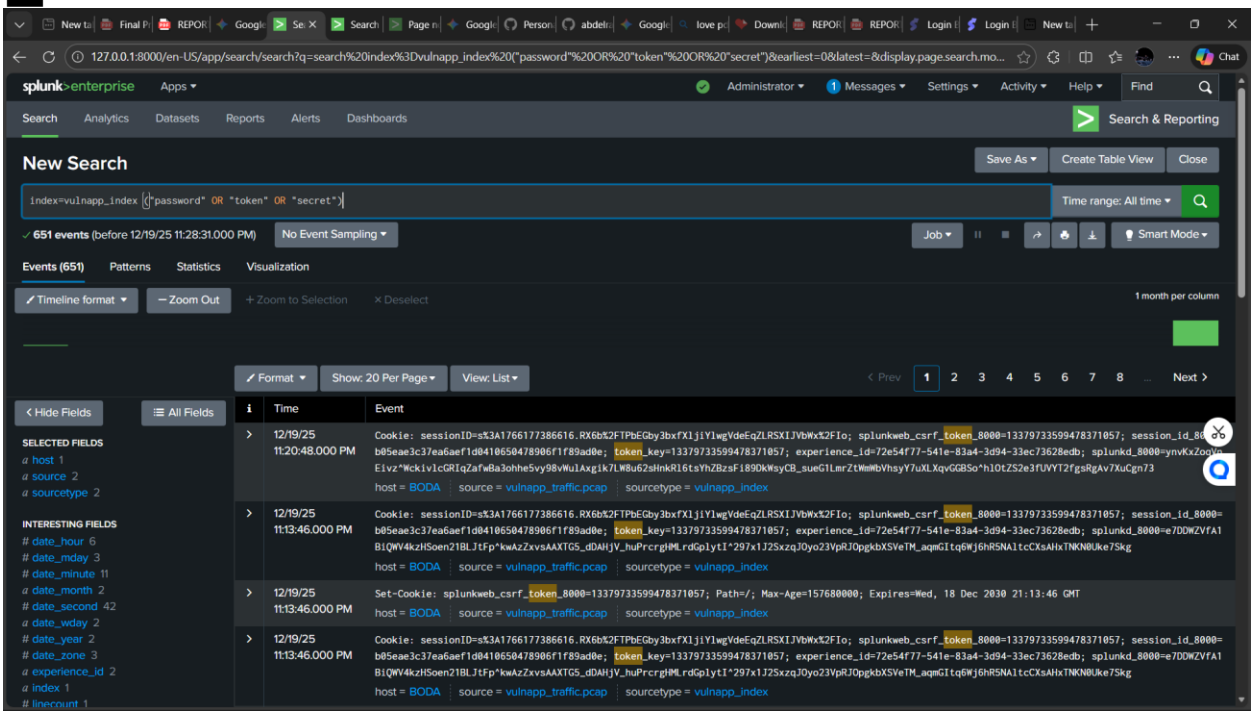
✔



⚫ V6 — Sensitive Data Exposure

```
index=vulnapp_index ("password" OR "token" OR "secret")
```

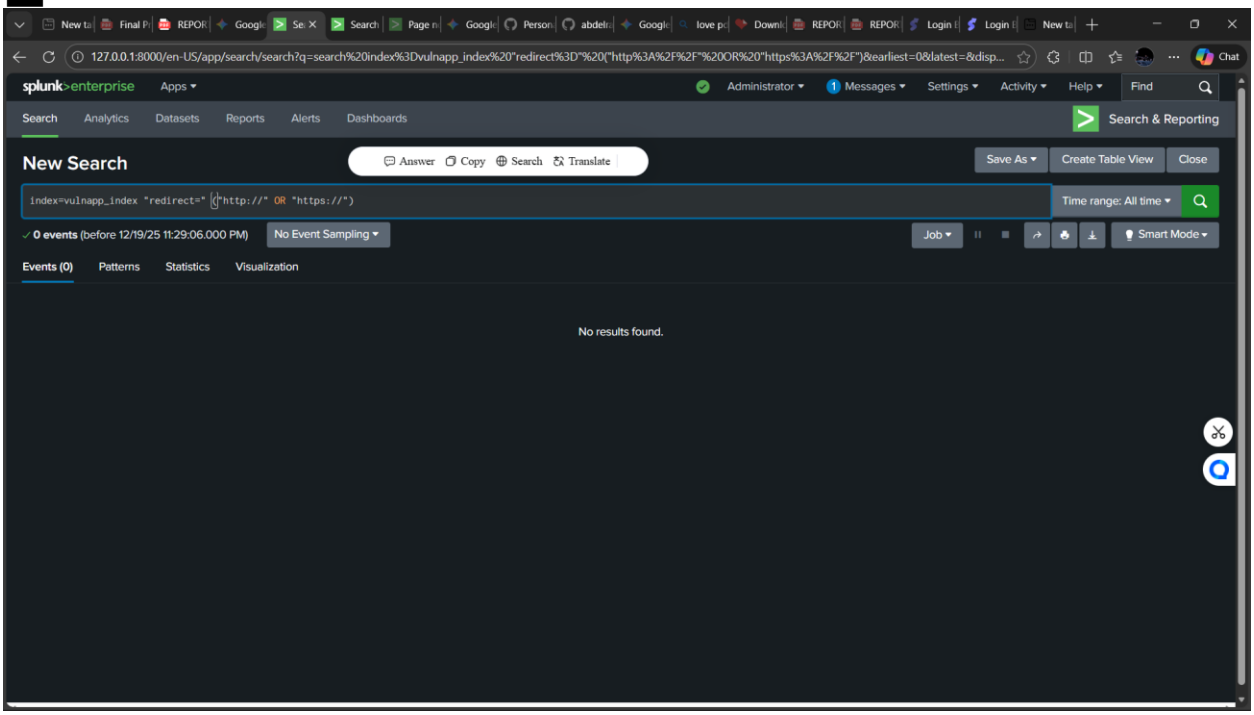
✔



● V7 — Unvalidated Redirect

```
index=vulnapp_index "redirect=" ("http://" OR "https://")
```

✔



⚫ V8 — Verb Tampering / Admin

`index=vulnapp_index (PUT OR DELETE) "/admin"`

✔