

Week 2 – Implementing Security Measures

Application: User Management System (Express + MongoDB)

Prepared by: Muhammad Raza

Date: February 2026

1. Executive Summary

During the second week the team applied the corrective actions outlined in *Week 1 – Vulnerability Assessment Report*. All critical findings (XSS, NoSQL injection, plaintext passwords, missing headers) were addressed by introducing input validation, password hashing, token-based login and secure HTTP headers. Each section below includes a brief implementation note, supporting screenshots pulled from the [docs/screenshots](#) directory, and a description of the security impact. A remediation summary table and references follow.

 **Note:** This is a follow-up report; consult [Week1_Report.md](#) for the original risk analysis and proof-of-concept evidence.

2. Fixing Vulnerabilities

The following subsections correspond to the high-risk findings reported in Week 1; each includes implementation details, proof-of-concept screenshots and impact statements.

2.1 Input Validation & Sanitization

Implementation

The [validator](#) npm package was added and used on both signup and login routes to enforce sane formats and strip malicious content.

```
const validator = require('validator');

if (!validator.isEmail(email)) {
  return res.status(400).send('Invalid email format');
}
if (!validator.isAlphanumeric(name)) {
  return res.status(400).send('Invalid name');
}
if (!validator.minLength(password, { min: 8 })) {
  return res.status(400).send('Password must be at least 8 characters');
}
```

Proof of Concept

```
(razajavaid@Kali)-[~/Desktop/User-Management-System-CRUD]
$ npm install validator

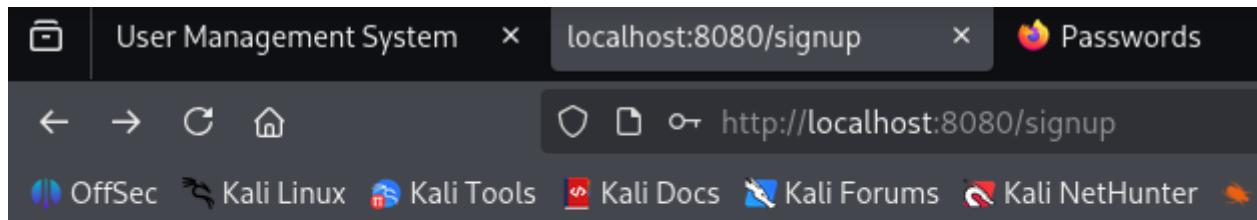
added 1 package, and audited 118 packages in 12s

19 packages are looking for funding
  run `npm fund` for details
25 high severity vulnerabilities
To address all issues, run:
  npm audit fix
  9 apply helmet globally for secure HTTP headers
Run `npm audit` for details.
```

Above: terminal output showing the `npm install validator` command completing successfully.

```
try {
  // Input validation
  if (!validator.isEmail(email)) {
    return res.status(400).send('Invalid email format');
  }
  if (!validator.isAlphanumeric(name)) {
    return res.status(400).send('Invalid name');
  }
  if (!validator.minLength(password, { min: 8 })) {
    return res.status(400).send('Password must be at least 8 characters');
}
```

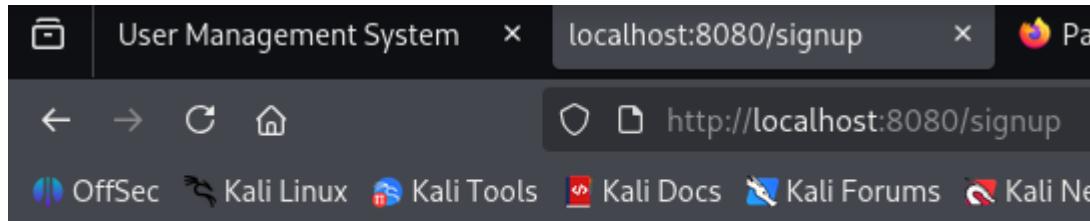
Code snippet from the signup route where input is checked using validator methods.



Signup

<script>alert('XSS')</script>	test@gmail.com	Signup
-------------------------------	----------------	-------	--------

The user registration form accepts properly formatted data.



Invalid name

Application returns an error message when the name field contains invalid characters or a script payload.

Impact

Rejects malformed or malicious payloads before they reach the database, eliminating the stored-XSS vector described in Week 1.

2.2 Password Hashing

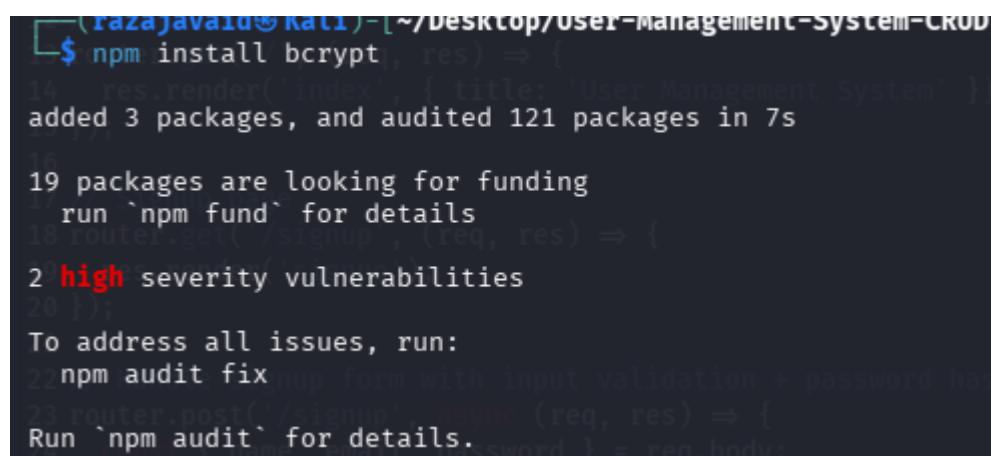
Implementation

`bcrypt` was introduced during user registration to hash passwords. The hashing occurs prior to saving the document; the raw password is never stored.

```
const bcrypt = require('bcrypt');

const hashedPassword = await bcrypt.hash(password, 10);
const user = new User({ name, email, password: hashedPassword });
await user.save();
```

Proof of Concept



```
(razajavaid@Rati)-[~/Desktop/User-Management-System-CRUD]
$ npm install bcrypt, res) => {
14  res.render('index', { title: 'User Management System' })
added 3 packages, and audited 121 packages in 7s
15
16 19 packages are looking for funding
  run `npm fund` for details
17 router.get('/signup', (req, res) => {
18   2 high severity vulnerabilities
19 });
20 });
To address all issues, run:
21 npm audit fix
22 router.post('/signup', async (req, res) => {
23   Run `npm audit` for details.
24   const hashedPassword = await bcrypt.hash(req.body.password, 10);
25   const user = new User({ name, email, password: hashedPassword });
26   await user.save();
27
28   res.send(`Signup successful: ${name}`);
29 } catch (err) {
30   res.status(400).send('Error: ' + err.message);
31 }
```

Terminal log confirming `npm install bcrypt` was executed.

```
// Password hashing
const hashedPassword = await bcrypt.hash(password, 10);

const user = new User({ name, email, password: hashedPassword });
await user.save();

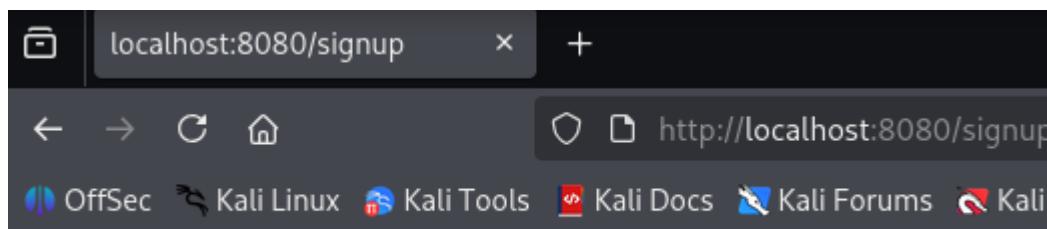
res.send(`Signup successful: ${name}`);
} catch (err) {
  res.status(400).send('Error: ' + err.message);
}
```

Server-side code that hashes the password before creating the user document.

localhost:8080/signup

test123 test123@gmail.com Signup

Browser screenshot showing a completed registration form.



Page indicating account creation; password is not displayed.

```
"_id" : ObjectId("699dbc7a14d4e947845e0621"),
"name" : "test123",
"email" : "test123@gmail.com",
"password" : "$2b$10$r18fJPAfQ.hhYAyEegwjMu09UmAcIHK9cfKT8gpSBT4j1IEtYabDy",
"__v" : 0
```

MongoDB shell output displaying the stored hashed password instead of plaintext.

Impact

Mitigates credential exposure if the database is leaked; hashes are computationally expensive to reverse.

2.3 Token-Based Authentication

Implementation

Replaced the previous sessionless login with JWT tokens using `jsonwebtoken`. Tokens are signed with a secret and expire after one hour.

```
const jwt = require('jsonwebtoken');

const token = jwt.sign({ id: user._id }, 'your-secret-key', { expiresIn: '1h' });
res.send({ message: 'Login successful', token });
```

Proof of Concept

```

25 (razajavaid@Kali)-[~/Desktop/User-Management-System-CRUD]
26 $ npm install jsonwebtoken
27   // Input validation
28   added 14 packages, and audited 135 packages in 20s
29     1 package is looking for funding
30       run `npm fund` for details
31     2 high severity vulnerabilities
32       1 package is looking for funding
33         run `npm fund` for details
34       1 package has known vulnerabilities
35         res.status(400).send('Password must be at least 8 characters long');
36
Run `npm audit` for details.

```

Installation output for the `jsonwebtoken` package.

```

// Generate JWT token
const token = jwt.sign({ id: user._id }, 'your-secret-key', { expiresIn: '1h' });

res.send({ message: 'Login successful', token });
} catch (err) {
  res.status(500).send('Error: ' + err.message);
}

```

Server code that signs and returns a JWT after successful authentication.

The screenshot shows a browser window with the address bar set to `localhost:8080/login`. The main content area displays a simple login form with two text input fields and a 'Login' button. The first field contains the email `test123@gmail.com`, and the second field contains a masked password. The browser interface includes standard navigation buttons (back, forward, refresh) and a toolbar with links to various Kali Linux resources.

Browser view of the login page with valid credentials filled in.

The screenshot shows the Network tab of a browser developer tools panel. An incoming request from `localhost:8080/login` is visible. The response is a 200 OK status. The response body is a JSON object containing a `message` field with the value `"Login successful"` and a `token` field with a long encoded string.

HTTP response showing the issued token and success message.

Impact

Enables stateless authentication, avoids storing credentials on the client and reduces session hijacking risk.

2.4 HTTP Security Headers

Implementation

The `helmet` middleware was applied at the router level to inject several security headers by default, including `X-Frame-Options`, `X-Content-Type-Options` and `Strict-Transport-Security`.

```
const helmet = require('helmet');
router.use(helmet());
```

Proof of Concept

```
[razajavaid@Kali:~/Desktop/User-Management-System-CRUD]
$ npm install helmet
32   added 1 package, and audited 136 packages in 4s
33     19 packages are looking for funding
34       run `npm fund` for details
35
36 2 high severity vulnerabilities
37    To address all issues, run:
38      npm audit fix
39    Run `npm audit` for details.
```

Terminal confirming `npm install helmet`.

```
// Apply Helmet globally for secure HTTP headers
router.use(helmet());
```

Code snippet where the `helmet()` middleware is attached to the router.

Header	Value
Status	200 OK
Version	HTTP/1.1
Transferred	1.09 kB (225 B size)
Referrer Policy	no-referrer
Request Priority	Highest
DNS Resolution	System
Response Headers (869 B)	
Connection	keep-alive
Content-Length	225
Content-Security-Policy	default-src 'self'; base-uri 'self'; font-src 'self' https: data: form-action 'self'; frame-ancestors 'self'; img-src 'self' data: object-src 'none'; script-src 'self'; script-src-attr 'none'; style-src 'self' https: 'unsafe-inline'; upgrade-insecure-requests
Content-Type	text/html; charset=utf-8
Cross-Origin-Opener-Policy	same-origin
Cross-Origin-Resource-Policy	same-origin
Date	Tue, 24 Feb 2026 15:09:04 GMT
ETag	W/"e1-t4Q2ZocXpzXBUBwEjzWsh/h0"
Keep-Alive	timeout=5
Origin-Agent-Cluster	?1
Referrer-Policy	no-referrer
Strict-Transport-Security	max-age=31536000; includeSubDomains
X-Content-Type-Options	nosniff
X-DNS-Prefetch-Control	off
X-Download-Options	noopen
X-Frame-Options	SAMEORIGIN
X-Permitted-Cross-Domain-Policies	none
X-XSS-Protection	0

Browser developer tools network panel listing the new headers (`X-Frame-Options`, `X-Content-Type-Options`, etc.).

Impact

Provides defense-in-depth against clickjacking, MIME confusion and other header-based attacks.

3. Remediation Summary

Area	Library / Tool	Key Change	Screenshot(s)
Input validation	validator	Added format checks and sanitisation	validator_install, validator_code, validator_signup, validator_error
Password storage	bcrypt	Hash before save	bcrypt_install, bcrypt_code, bcrypt_browser1/2, bcrypt_mongo
Authentication	jsonwebtoken	JWT tokens with 1-h expiry	jwt_install, jwt_code, jwt_browser, jwt_token
HTTP headers	helmet	Applied secure headers globally	helmet_install, helmet_code, helmet_headers

4. References

- [validator npm package](#)
- [bcrypt documentation](#)
- [JSON Web Tokens \(jwt.io\)](#)
- [Helmet.js middleware](#)
- OWASP Top 10 – 2021
- OWASP XSS Prevention Cheat Sheet
- Node.js Security Checklist

5. Conclusion

All high-severity vulnerabilities identified in Week 1 have now been remediated. The application enforces proper input validation, stores passwords securely, uses stateless token authentication, and sets recommended HTTP headers. Week 3 will focus on automated validation tests, implementing CSP, enforcing HTTPS, and verifying the fixes. Continual monitoring and security testing remain essential as development progresses.