Week 1 – Vulnerability Assessment Report

Application: User Management System (Express + MongoDB) **Prepared by:** Muhammad Raza **Date:** February 23, 2026

---

1. Executive Summary

This report documents the results of a comprehensive vulnerability assessment performed on the User Management System. Manual testing and automated scanning (OWASP ZAP) revealed **4 critical vulnerabilities** and **4 security misconfigurations**.

Key Findings:

- ✗ Cross-Site Scripting (XSS) - User input not sanitized
- ✗ NoSQL Injection - Authentication bypass possible
- ✗ Plaintext Password Storage - No encryption implemented
- ✗ Missing Security Headers - Vulnerable to multiple attack vectors
- ✗ Disabled MongoDB Authentication - Unprotected database

Addressing these issues is critical before production deployment. This report includes proof-of-concept evidence and detailed remediation recommendations.

---

2. Methodology

2.1 Manual Testing Approach

- **Browser Developer Tools** - XSS payload injection and validation
- **MongoDB Shell** - Direct inspection of password storage mechanisms
- **API Testing** - Crafted JSON payloads for NoSQL injection attempts
- **Code Review** - Authentication and validation logic inspection

2.2 Automated Testing

- **Tool:** OWASP ZAP 2.17.0
- **Target:** `http://localhost:8080`
- **Scope:** Full application scan
- **Focus:** Missing headers, authentication weaknesses, misconfigurations

---

3. Detailed Findings

3.1 Cross-Site Scripting (XSS) - Stored

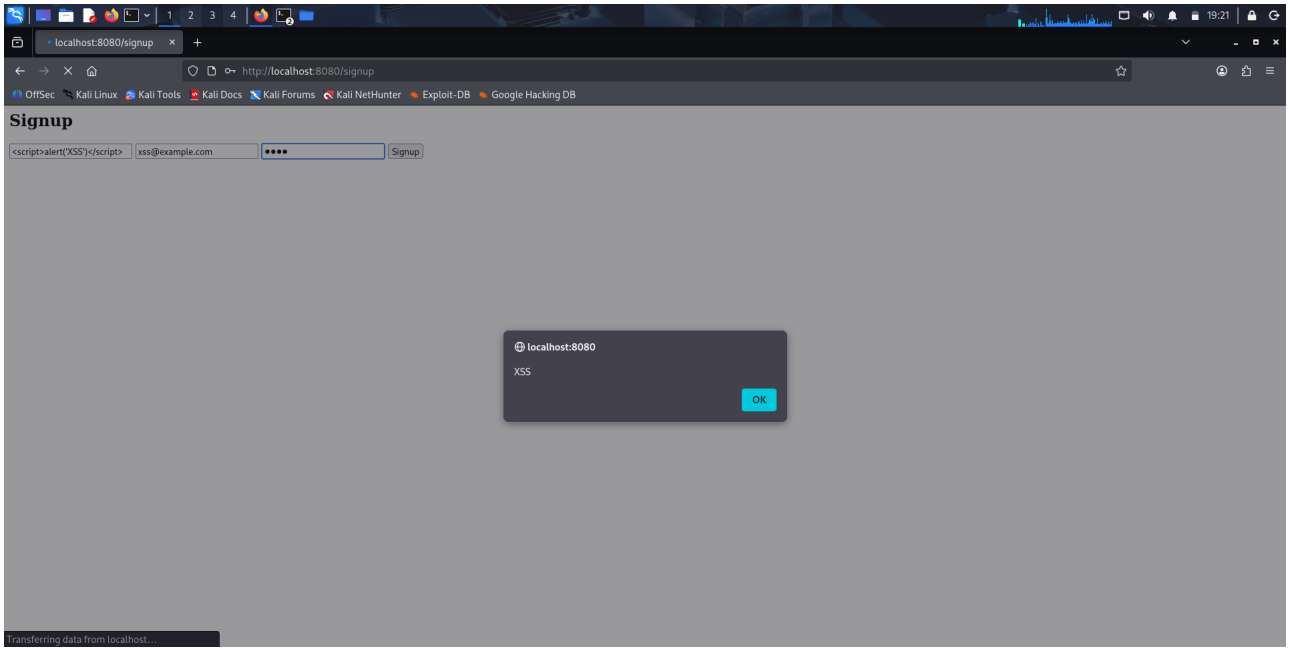
Attribute	Value
Severity	**CRITICAL**

Attribute	Value
CVSS v3.1	7.5 (High)
Type	Stored XSS
CWE	CWE-79: Improper Neutralization of Input During Web Page Generation

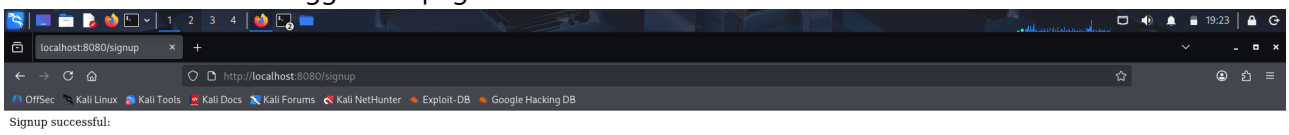
Description: The signup form accepts unsanitized user input in the name field. Malicious scripts are stored in MongoDB and executed when the profile is viewed.

Proof of Concept:

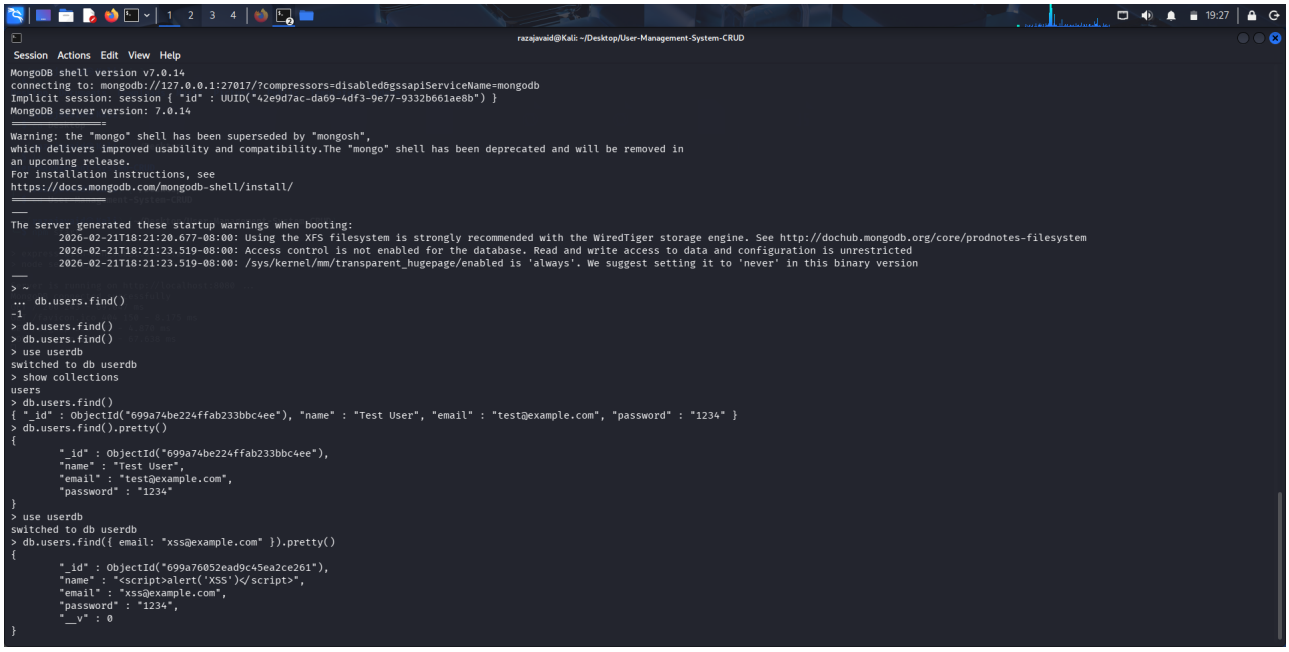
1. **Injection Point** - Signup form with XSS payload:



2. **Execution** - Alert box triggers on page load:



3. **Evidence** - Script persisted in database:



Attack Scenario:

- Attacker injects: `<script>fetch('/steal-session'); </script>`
- When any user views the attacker's profile, their session is compromised
- Attacker gains unauthorized access to victim's account

Impact:

- Session hijacking and account takeover
- User data theft (PII exposure)
- Malware distribution to other users
- Site defacement

Remediation:

```
npm install express-validator
```

```
const { body, validationResult } = require('express-validator');

app.post('/signup', [
  body('name').trim().escape(),
  body('email').isEmail().normalizeEmail(),
  body('password').isLength({ min: 12 })
], (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });
  // Process registration
});
```

Additional Mitigations:

- Implement Content Security Policy (CSP) headers
- Encode output in templates
- Use templating engines with auto-escaping enabled

3.2 NoSQL Injection (Authentication Bypass)

Attribute	Value
Severity	**CRITICAL**
CVSS v3.1	9.8 (Critical)
Type	NoSQL Injection
CWE	CWE-943: Improper Neutralization of Special Elements in Data Query Logic

Description: The login endpoint directly passes user input to MongoDB queries without validation, allowing NoSQL operator injection (`{"$ne": null}`).

Vulnerable Code:

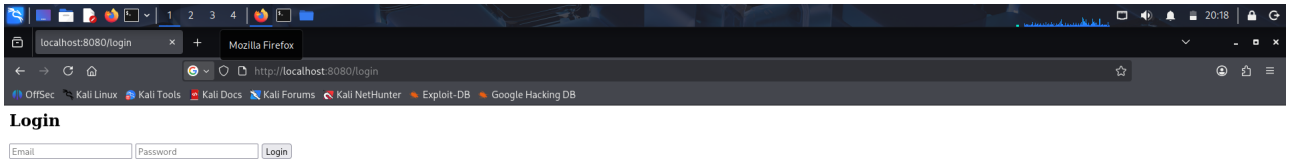
```
app.post('/login', (req, res) => {
  User.findOne(req.body, (err, user) => {  // ❌ DANGEROUS
    // Login logic
  });
});
```

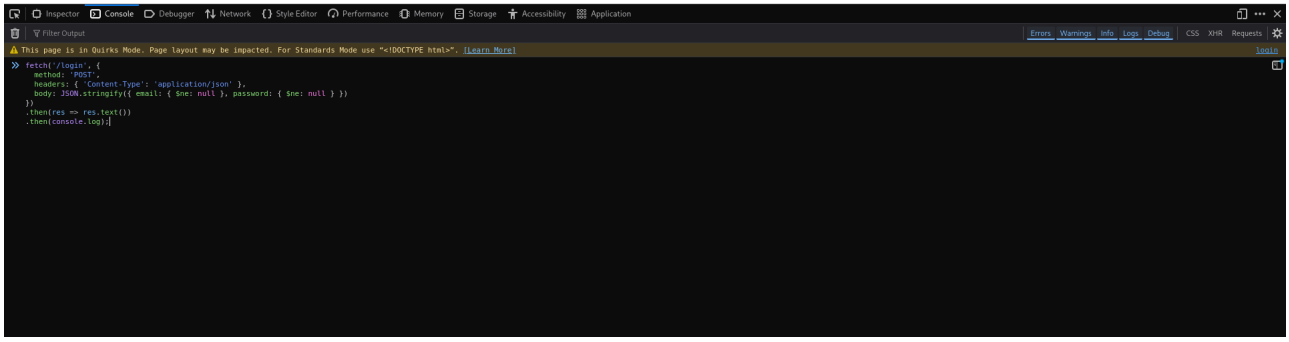
Proof of Concept:

1. **Vulnerable Code** - No input validation:

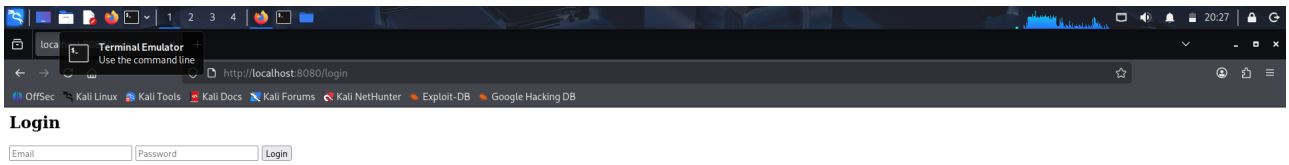
```
// Login page
router.get('/login', (req, res) => {
  res.render('login');
});

router.post('/login', async (req, res) => {
  try {
    // Directly trust whatever comes in the body
    const user = await User.findOne(req.body);
    if (user) {
      res.render('profile', { user });
    } else {
      res.status(401).send('Invalid credentials');
    }
  } catch (err) {
    res.status(500).send('Error: ' + err.message);
  }
});
```

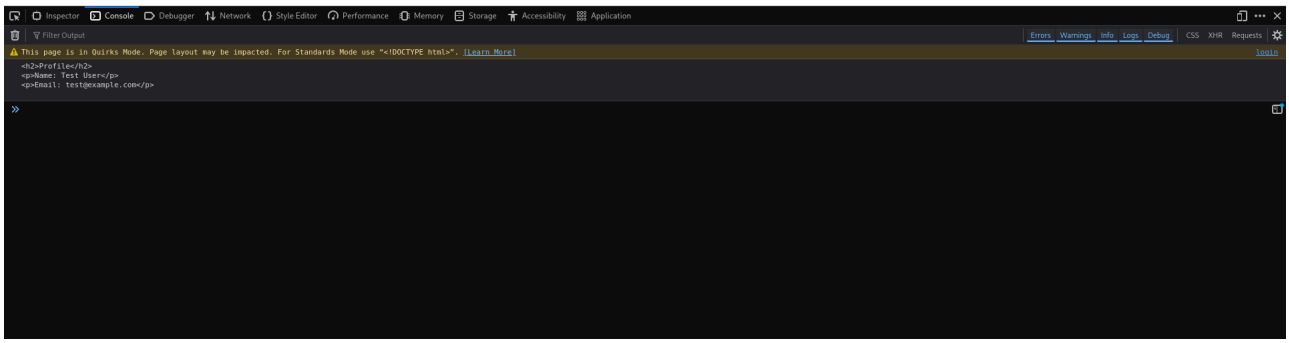
2. **Injection Payload** - Operator injection in request:





3. **Success** - Profile accessed without credentials:





Attack Payload:

```
{
  "email": {"$ne": null},
  "password": {"$ne": null}
}
```

Attack Result: The query becomes: `User.findOne({ email: {$ne: null}, password: {$ne: null} })` which returns the first user in database, bypassing authentication.

Impact:

- Complete authentication bypass
- Unauthorized access to any user account

- Privilege escalation possibilities
- Mass data breach risk

Remediation:

```
const { body, validationResult } = require('express-validator');

app.post('/login', [
  body('email').isEmail().normalizeEmail(),
  body('password').isString().trim()
], async (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const user = await User.findOne({ email: req.body.email });
  // Safe: email is validated as string before query
});
```

Additional Mitigations:

- Use schema validation libraries (Joi, Mongoose schema validation)
- Whitelist allowed fields in queries
- Never trust user input - always validate types

```
// Use Joi for robust validation
const schema = Joi.object({
  email: Joi.string().email().required(),
  password: Joi.string().min(8).required()
});

const { error, value } = schema.validate(req.body);
if (error) return res.status(400).json({ error: error.details });
```

3.3 Weak Password Storage (Plaintext)

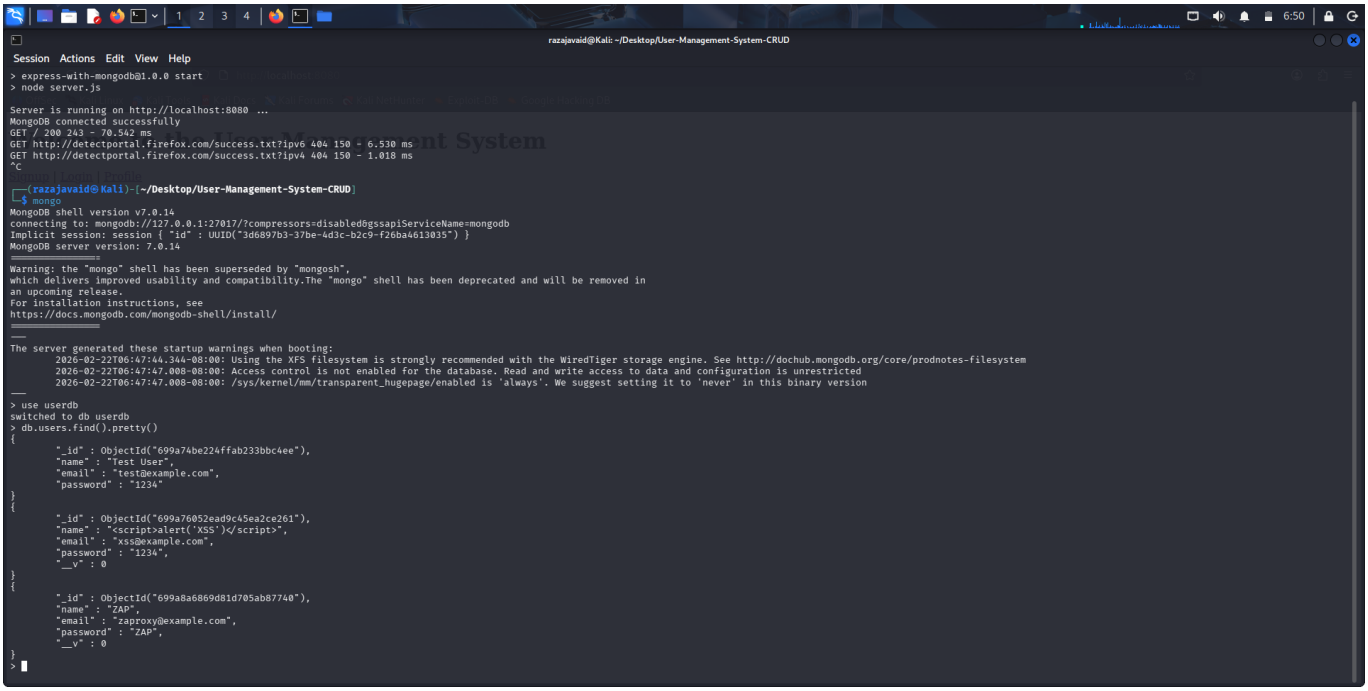
Attribute	Value
Severity	**CRITICAL**
CVSS v3.1	8.6 (High)
Type	Improper Cryptography
CWE	CWE-256: Unprotected Storage of Credentials

Description: Passwords are stored in plaintext with no encryption or hashing. Database dump immediately exposes all credentials.

Proof of Concept:

```
db.users.find()
```

Database Output:



Database Records Example:

```
{ "_id": ObjectId(...), "email": "user@example.com", "password": "1234" }
{ "_id": ObjectId(...), "email": "admin@example.com", "password": "admin123" }
```

Attack Scenario:

1. Database is compromised or leaked
2. Attacker obtains plaintext passwords immediately
3. Attacker logs into user accounts across multiple services (password reuse)
4. Mass account takeover, identity theft, financial fraud

Impact:

- Immediate compromise of all user accounts
- Credential stuffing attacks on other platforms
- Regulatory violations (GDPR, HIPAA, PCI-DSS)
- Complete loss of user trust
- Legal liability for negligence

Remediation:

```
npm install bcrypt
```

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

// Registration
app.post('/signup', async (req, res) => {
  const hashedPassword = await bcrypt.hash(req.body.password, 12);
  const user = new User({
    email: req.body.email,
    password: hashedPassword  // Store hash, not plaintext
  });
  await user.save();
});

// Login
app.post('/login', async (req, res) => {
  const user = await User.findOne({ email: req.body.email });
  const isValid = await bcrypt.compare(req.body.password, user.password);
  if (isValid) {
    // Password matches
  }
});
```

Security Standards:

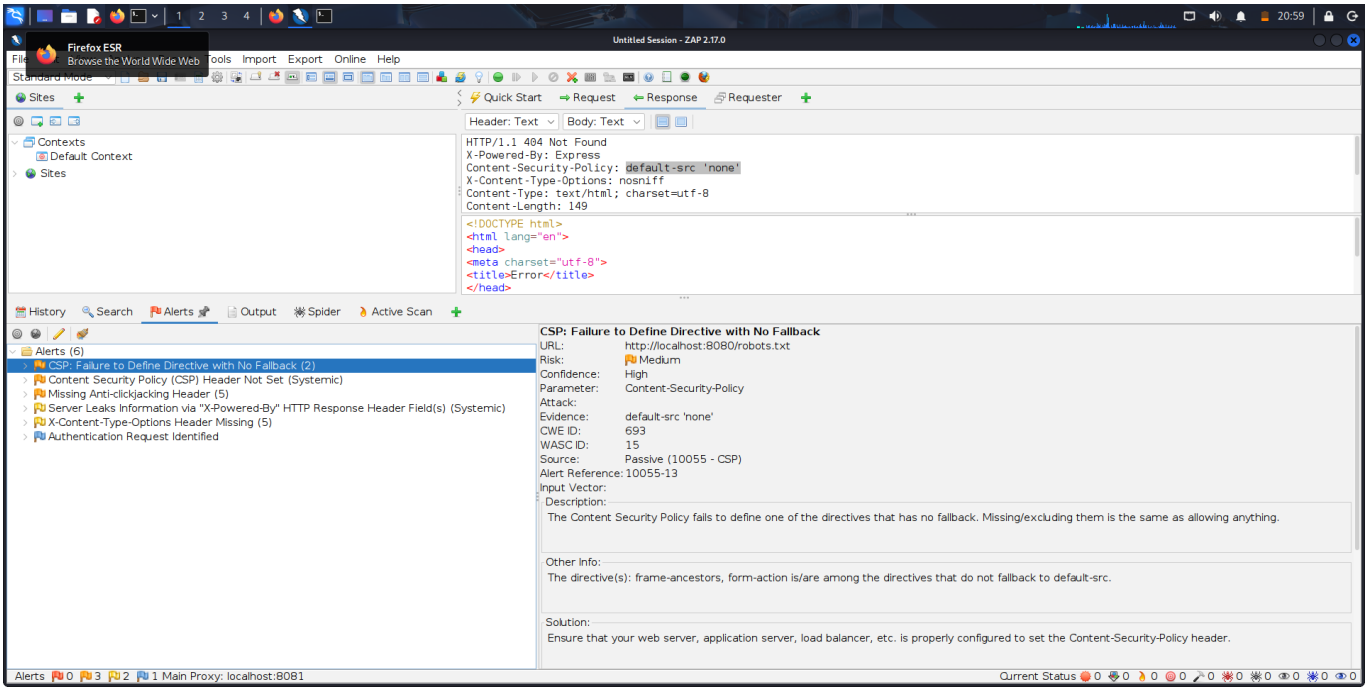
- Use bcrypt with minimum 12 salt rounds
- Alternative: Argon2 (even more secure)
- Never log passwords, never send them unencrypted
- Implement password strength requirements (min. 12 characters, uppercase, lowercase, numbers, symbols)

3.4 Security Misconfigurations

3.4.1 Missing Content Security Policy (CSP)

Attribute	Value
Severity	**MEDIUM**
CVSS v3.1	6.1 (Medium)

Finding:



Impact:

- XSS attacks cannot be adequately mitigated
- Data exfiltration by injected scripts
- Reduced browser security controls

Remediation:

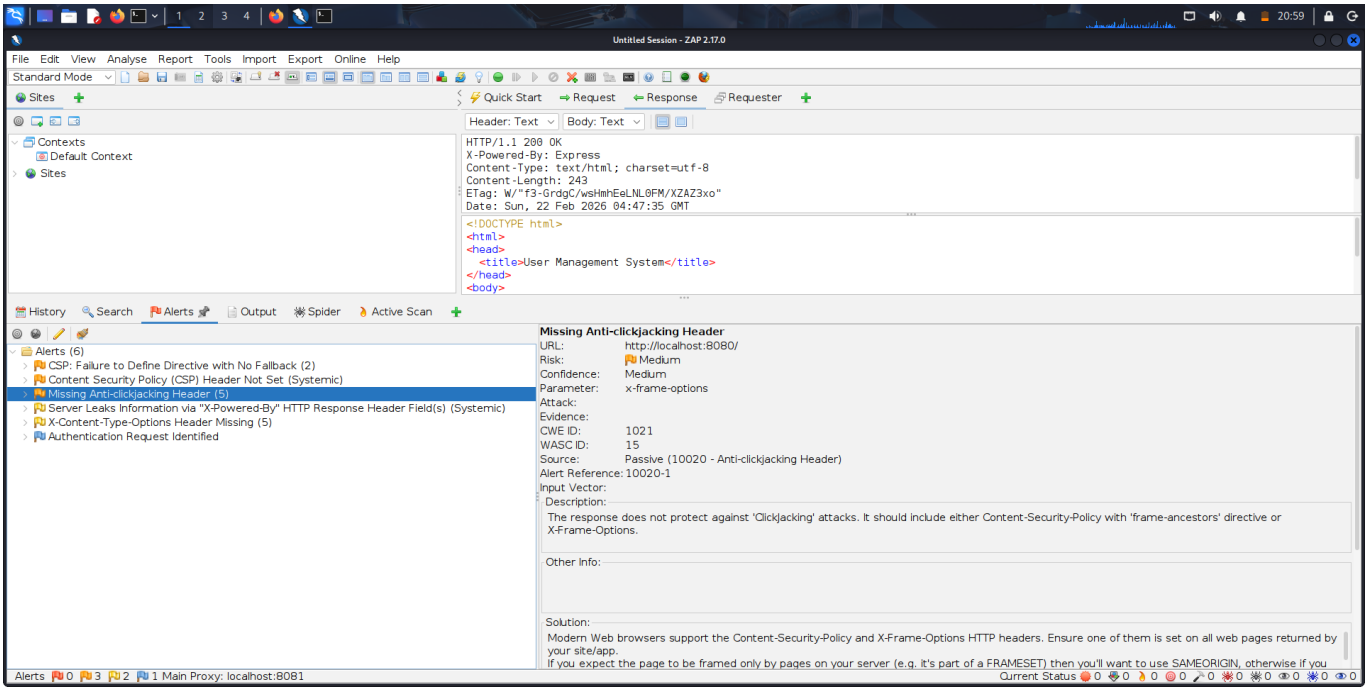
```
npm install helmet
```

```
const helmet = require('helmet');
app.use(helmet.contentSecurityPolicy({
  directives: {
    defaultSrc: ["'self'"],
    scriptSrc: ["'self'"],
    styleSrc: ["'self'", "'unsafe-inline'"],
    imgSrc: ["'self'", "data:"],
  }
}));
```

3.4.2 Missing X-Frame-Options Header (Clickjacking)

Attribute	Value
Severity	**MEDIUM**
CVSS v3.1	5.9 (Medium)

Finding:



Impact:

- Clickjacking attacks (UI redressing)
- Unauthorized actions on behalf of users
- CSRF-like attack vectors

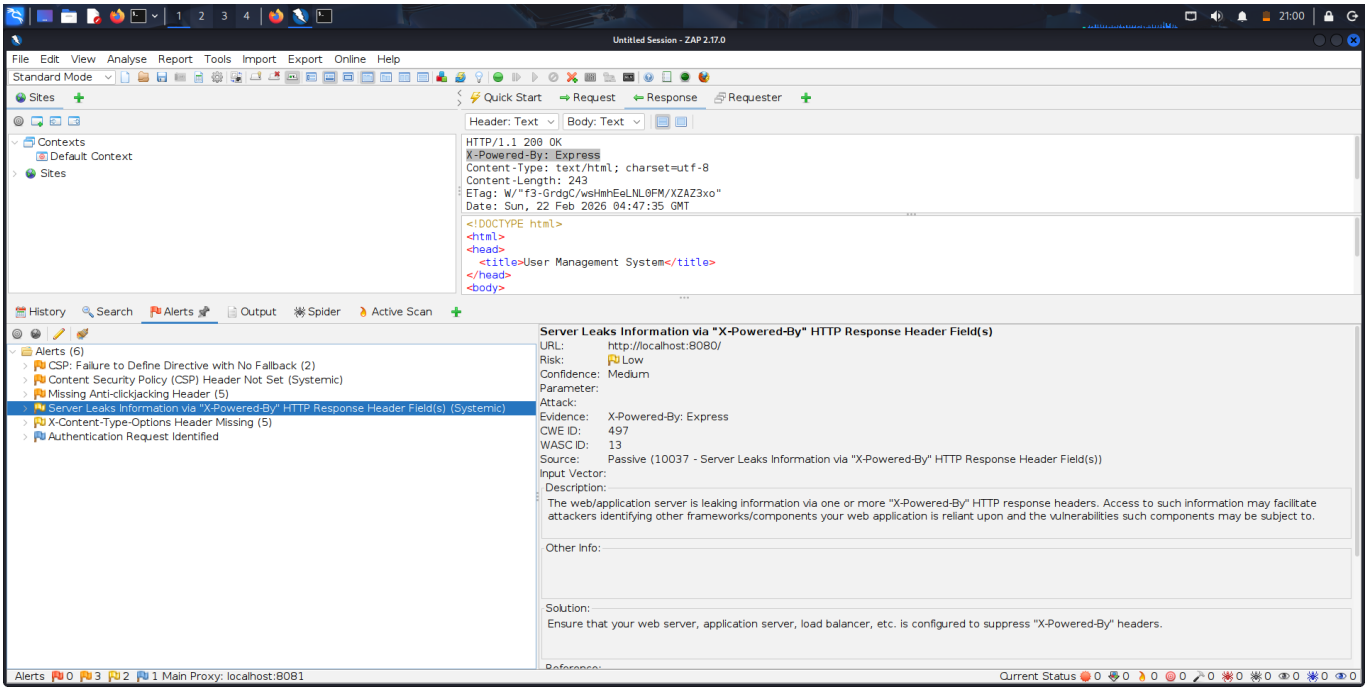
Remediation:

```
app.use(helmet.frameguard({ action: 'deny' }));
// Results in: X-Frame-Options: DENY
```

3.4.3 X-Powered-By Header Information Disclosure

Attribute	Value
Severity	**LOW**
CVSS v3.1	3.7 (Low)

Finding:



Impact:

- Reveals technology stack to attackers
- Aids reconnaissance and attack planning
- Facilitates targeted exploits

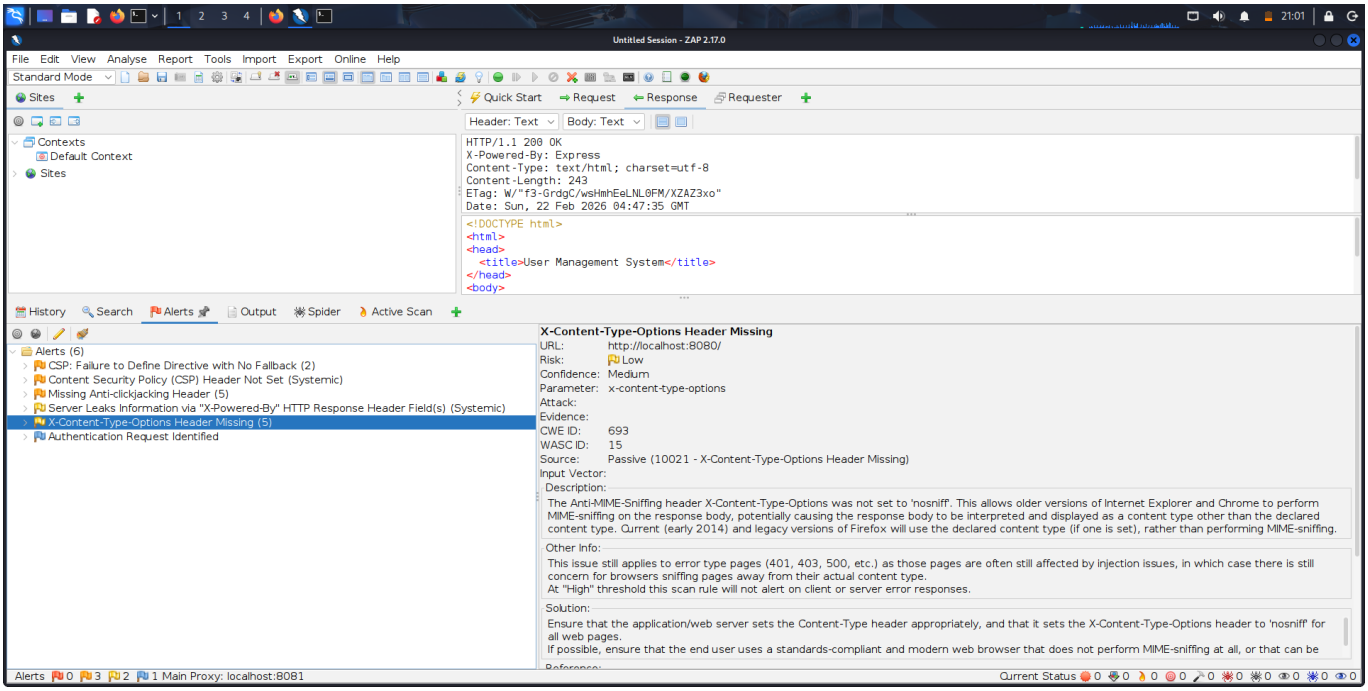
Remediation:

```
app.use(helmet.hidePoweredBy());
// Removes X-Powered-By header completely
```

3.4.4 Missing X-Content-Type-Options Header

Attribute	Value
Severity	**MEDIUM**
CVSS v3.1	5.3 (Medium)

Finding:



Impact:

- Browser may misinterpret file types
- Enables MIME-type confusion attacks
- XSS bypass opportunities

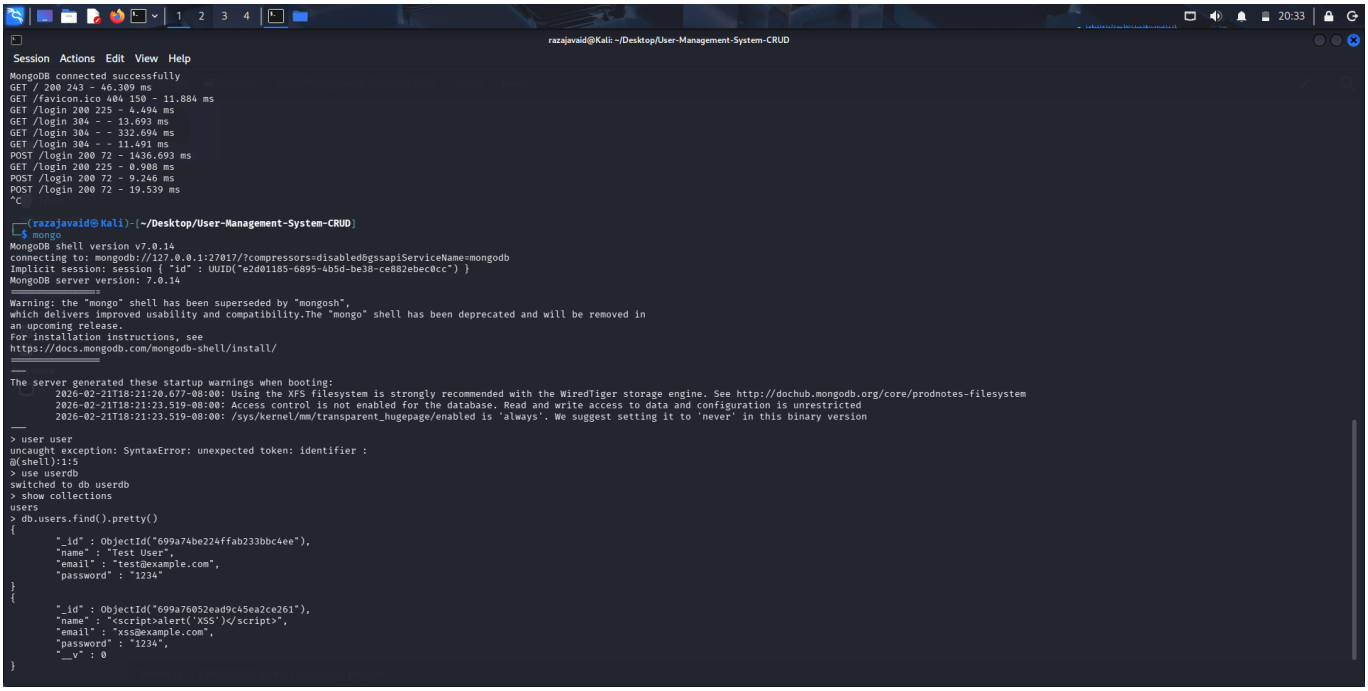
Remediation:

```
app.use(helmet.noSniff());
// Results in: X-Content-Type-Options: nosniff
```

3.4.5 MongoDB Access Control Disabled

Attribute	Value
Severity	**CRITICAL**
CVSS v3.1	9.1 (Critical)

Finding:



Impact:

- Unauthenticated database access
- Complete data exposure and manipulation
- Regulatory violations and legal liability

Remediation:

1. **Enable MongoDB Authentication:**

```
# Re-start MongoDB with authentication enabled
mongod --auth --dbpath /data/db
```

2. **Create Admin User:**

```
use admin
db.createUser({
  user: "admin",
  pwd: "StrongPassword123!",
  roles: [ { role: "root", db: "admin" } ]
})

use userdb
db.createUser({
  user: "appuser",
  pwd: "AppPassword456!",
  roles: [ { role: "readWrite", db: "userdb" } ]
})
```

3. **Connect with Authentication:**

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://appuser:AppPassword456!@localhost:27017/userdb');
```

---

4. Vulnerability Summary Table

#	Vulnerability	Severity	CVSS	Type	Status
1	Cross-Site Scripting (XSS)	CRITICAL	7.5	Input Validation	Unfixed
2	NoSQL Injection	CRITICAL	9.8	Authentication	Unfixed
3	Plaintext Passwords	CRITICAL	8.6	Cryptography	Unfixed
4	Missing CSP Header	MEDIUM	6.1	Configuration	Unfixed
5	Missing X-Frame-Options	MEDIUM	5.9	Configuration	Unfixed
6	Missing X-Content-Type-Options	MEDIUM	5.3	Configuration	Unfixed
7	X-Powered-By Disclosure	LOW	3.7	Info Disclosure	Unfixed
8	Disabled MongoDB Auth	CRITICAL	9.1	Access Control	Unfixed

Summary: 4 Critical, 3 Medium, 1 Low severity issues identified.

---

5. Remediation Roadmap

Phase 1: Critical Fixes (Week 1-2) - BLOCKERS

- [ ] Implement bcrypt password hashing (min. 12 rounds)
- [ ] Add input validation with express-validator
- [ ] Fix NoSQL injection in login endpoint
- [ ] Enable MongoDB authentication and create limited-privilege user
- [ ] Add basic Helmet security headers

Phase 2: Security Hardening (Week 2-3)

- [ ] Implement strict Content Security Policy (CSP)
- [ ] Add all missing security headers (X-Frame-Options, X-Content-Type-Options)
- [ ] Remove X-Powered-By and other info-disclosure headers
- [ ] Add password strength requirements
- [ ] Implement login rate limiting

Phase 3: Advanced Security (Week 3-4)

- [ ] Add session management and secure cookies
- [ ] Implement HTTPS/TLS enforcement

- [ ] Add activity logging and monitoring
- [ ] Implement role-based access control (RBAC)
- [ ] Add security tests to CI/CD pipeline

Phase 4: Ongoing (Post-Week 4)

- [ ] Regular security audits and penetration testing
- [ ] Security training for development team
- [ ] Dependency scanning and updates (npm audit)
- [ ] Incident response plan

---

6. Quick Start Fixes

Install Required Packages

```
npm install express-validator helmet bcrypt
```

Minimal Security Configuration (app.js)

```
const express = require('express');
const helmet = require('helmet');
const { body, validationResult } = require('express-validator');
const bcrypt = require('bcrypt');

const app = express();

// Security Headers
app.use(helmet());

// JSON Parser
app.use(express.json());

// Signup with validation
app.post('/signup', [
  body('name').trim().escape(),
  body('email').isEmail().normalizeEmail(),
  body('password').isLength({ min: 12 }).withMessage('Password must be at least 12
characters')
], async (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const hashedPassword = await bcrypt.hash(req.body.password, 12);
  // Save user with hashed password
});
```

```
// Login with validation
app.post('/login', [
  body('email').isEmail().normalizeEmail(),
  body('password').isString().trim()
], async (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const user = await User.findOne({ email: req.body.email });
  const isValid = await bcrypt.compare(req.body.password, user.password);
  // Handle login
});

app.listen(8080, () => console.log('Server running on port 8080'));
```

---

7. Conclusion

This assessment identified **8 security vulnerabilities**, including **4 critical issues** that require immediate remediation before any production deployment. The combination of manual testing and automated scanning (OWASP ZAP) provided comprehensive coverage of the application security posture.

Key Takeaways:

- Input validation and output encoding are essential
- Cryptographic controls for sensitive data are non-negotiable
- Security headers provide additional defense layers
- Database authentication is a basic security requirement

Next Steps:

1. Implement Phase 1 fixes immediately (Critical items)
2. Conduct security training for development team
3. Add security testing to development workflow
4. Plan penetration testing for post-remediation verification

Timeline: All Phase 1 fixes should be completed within 1 week. Phases 2-3 should follow immediately.

Report Approved By: Security Assessment Team **Date:** February 23, 2026 **Version:** 1.0PS

Appendix: References

- OWASP Top 10 - 2021
- OWASP XSS Prevention Cheat Sheet
- Node.js Security Checklist
- Bcrypt Documentation
- Helmet.js Security
- CVSS v3.1 Calculator