Code Review Report

Date: 26 June 2025

Assessment performed by Satvik Hatulkar

Table of Contents

Overview

Scope Definition

- Assess publicly accessible components(code) only.
- No intrusive testing (e.g., DDoS, automated brute-force, or any activity that disrupts services).
- Do not attempt unauthorized access to internal systems.

Assessment Areas

- **Web Application Code Review:** Look for common web vulnerabilities and logic flaws such as XSS, CSRF, SQLi, Broken Authorization, etc.

Reporting Guidelines

- Document all findings in a structured report.
- Use CVSS to rate vulnerabilities.
- Include possible mitigations.
- Ensure clarity, conciseness, and professionalism in the report.

Assessment Methodology

The code review was conducted using the **OWASP Code Review Guide** and **Secure Coding Best Practices** as the primary methodology. This approach focuses on identifying security flaws, insecure coding patterns, and logic vulnerabilities to ensure robust and secure application development

The findings from this assessment are categorized based on severity levels (Critical, High, Medium, Low) and mapped to OWASP Top 10 vulnerabilities. Each issue is documented with:

- Description of the vulnerability
- Proof of Concept (PoC)
- Potential Impact on the application
- Mitigation Recommendations

Assessment Duration and Dates

Scan Mode	Target	Started	Completed
Manual Code Review	Confidential	26 June 2025	27 June 2025

Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Medium	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Impact

Impact measures the potential vulnerability's effect on operations, including confidentiality, integrity, and availability of client systems and/or data, reputational harm, and financial loss.

Overview Table

No.	Domain	Title	Severity	Risk Score	Status
1	Confidential	NoSQL Injection in multiple fields and endpoints	High	8.2	Unresolved
2	Confidential	CSRF in multiple endpoints	High	8.0	Unresolved
3	Confidential	Brute Force Leading to Account Takeover	High	8.5	Unresolved
4	Confidential	Possible XSS due to inadequate sanitization of Bootstrap	High	8.2	Unresolved
5	Confidential	Weak password policy	Medium	6.2	Unresolved

Details of found vulnerabilities

1. NoSQL Injection in multiple fields and endpoints	
Severity	High
Status	Unresolved
Risk Score	8.2/10
CWE	CWE-943: Improper Neutralization of Special Elements in Data Query Logic
CVSS	8.2 (AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:L)
Labels	NoSQL Injection, Input Validation, Mongoose, MongoDB, Security Misconfiguration, Backend

Description

The application is vulnerable to **NoSQL injection** in multiple endpoints where user input is passed directly into MongoDB query or update operators without proper sanitization or validation. By tampering with request parameters (e.g., campground[title][$gt]=), an attacker can inject NoSQL operators into database queries or documents.
Affected endpoints include:

- POST /campgrounds (createCampground)
- PUT /campgrounds/:id (updateCampground)
- POST /register (User registration)

These endpoints use req.body or req.params directly in MongoDB operations like new Model(req.body), findByIdAndUpdate, or insertions without input filtering.

Impact

- **Unauthorized Data Access:** Attackers could bypass logical conditions or manipulate data.
- **Data Corruption:** Malformed operators like $gt, $where, $ne can be stored in DB fields.
- **Privilege Escalation or Enumeration:** In login or query flows, operators can cause logic bypass.
- **Potential for Application Errors:** Certain payloads cause crashes (e.g., CastError, type errors).

Proof Of Concept

```
module.exports.createNewCampground = async (req, res, next) => {
    const geoData = await geocoder.forwardGeocode({
        query: req.body.campground.location,
        limit:1
    }).send()

    const campground = new Campground(req.body.campground)
    campground.geometry = geoData.body.features[0].geometry
    campground.images = req.files.map(f => ({ url: f.path, filename: f.filename }))
    campground.author = req.user._id
    await campground.save()
    // console.log(campground)
    req.flash('success', 'Successfully made a new campground!')
    res.redirect(`/campgrounds/${campground._id}`)
}
```

```
module.exports.updateCampground = async (req, res) => {
    const { id } = req.params
    const campground = await Campground.findByIdAndUpdate(id, { ...req.body.campground })
    const imgs = req.files.map(f => ({ url: f.path, filename: f.filename }))
    campground.images.push(...imgs)
    await campground.save()
    if (req.body.deleteImages) {
        for (let filename of req.body.deleteImages) {
            await cloudinary.uploader.destroy(filename);
        }
        await campground.updateOne({ $pull: { images: { filename: { $in: req.body.deleteImages } } } })
    }
    req.flash('success', 'Successfully updated campground!')
    res.redirect(`/campgrounds/${campground._id}`)
}
```

```
module.exports.register = async (req, res) => {
    try {
        const { email, username, password } = req.body
        const user = new User({ email, username })
        const registeredUser = await User.register(user, password)
        req.login(registeredUser, err => {
            if (err) return next(err)
            req.flash('success', 'Welcome to YelpCamp!')
            res.redirect('/campgrounds')
        })
    } catch (e) {
        req.flash('error', e.message)
        res.redirect('/register')
    }
}
```

Suggested Fix

Input Validation & Sanitization

- Use libraries like Joi, Zod, or express-validator to strictly define and validate user input schema.

Whitelist Fields

- Use field whitelisting rather than blindly trusting req.body

Use express-mongo-sanitize Middleware

- Prevent payloads containing `$` or `.` in keys

Enforce Mongoose Schema Types

- Ensure all fields are explicitly typed (String, Number, etc.) to trigger schema validation errors.

2. CSRF in multiple endpoints	
Severity	High
Status	Unresolved
Risk Score	8.0 /10
CWE	CWE-352: Cross-Site Request Forgery (CSRF)
CVSS	8.0 (AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:H/A:N)
Labels	CSRF, WEB, State-Change, Session-Based, Auth-Bypass

Description

The application fails to enforce CSRF protection on several state-changing endpoints that rely on cookie-based authentication. As a result, an attacker can craft malicious web pages or forms that trigger unintended actions on behalf of an authenticated user.
The affected routes include:

- POST /campgrounds/:id/reviews – Create review
- DELETE /campgrounds/:id/reviews/:reviewId – Delete review
- POST /campgrounds – Create campground
- PUT /campgrounds/:id – Edit campground
- DELETE /campgrounds/:id – Delete campground

Impact

- **Privilege Abuse:** Attackers can force a logged-in victim to perform destructive actions (e.g., deleting reviews or posts).
- **Data Integrity Violation:** Malicious reviews, spam content, or unintended updates can be submitted on the victim's behalf.
- **Account Compromise:** If any admin functionality is affected, this could lead to privilege escalation or widespread impact.
- **Reputation Damage:** Malicious content injected via CSRF could lead to defacement or trust loss.

Proof Of Concept

```
module.exports.createReview = async (req, res) => {
    const { id } = req.params;
    if (!Types.ObjectId.isValid(id)) {
        req.flash('error', 'Invalid campground ID!');
        return res.redirect('/campgrounds');
    }

    const campground = await Campground.findById(req.params.id)
    const review = new Review(req.body.review)
    review.author = req.user._id
    campground.reviews.push(review)
    await review.save()
    await campground.save()
    req.flash('success', 'Created new review!')
    res.redirect(`/campgrounds/${campground._id}`)
}
```

```
module.exports.deleteReview = async (req, res) => {
    const { id, reviewId } = req.params;

    if (!Types.ObjectId.isValid(id) || !Types.ObjectId.isValid(reviewId)) {
        req.flash('error', 'Invalid ID!');
        return res.redirect('/campgrounds');
    }

    const campground = await Campground.findByIdAndUpdate(id, { $pull: { reviews: reviewId } })
    const review = await Review.findByIdAndDelete(reviewId);
    req.flash('success', 'Successfully deleted review!')
    res.redirect('/campgrounds/' + id)
}
```

```
module.exports.createNewCampground = async (req, res, next) => {
    const geoData = await geocoder.forwardGeocode({
        query: req.body.campground.location,
        limit:1
    }).send()

    const campground = new Campground(req.body.campground)
    campground.geometry = geoData.body.features[0].geometry
    campground.images = req.files.map(f => ({ url: f.path, filename: f.filename }))
    campground.author = req.user._id
    await campground.save()
    // console.log(campground)
    req.flash('success', 'Successfully made a new campground!')
    res.redirect(`/campgrounds/${campground._id}`)
}
```

```
module.exports.updateCampground = async (req, res) => {
    const { id } = req.params
    const campground = await Campground.findByIdAndUpdate(id, { ...req.body.campground })
    const imgs = req.files.map(f => ({ url: f.path, filename: f.filename }))
    campground.images.push(...imgs)
    await campground.save()
    if (req.body.deleteImages) {
        for (let filename of req.body.deleteImages) {
            await cloudinary.uploader.destroy(filename);
        }
        await campground.updateOne({ $pull: { images: { filename: { $in: req.body.deleteImages } } } })
    }
    req.flash('success', 'Successfully updated campground!')
    res.redirect(`/campgrounds/${campground._id}`)
}
```

Suggested Fix

Implement CSRF Protection Middleware

- This ensures that all state-changing requests must include a valid CSRF token.

Add CSRF Tokens in Forms and AJAX Requests

- For JavaScript requests, send token in headers.

Enforce Secure Cookie Flags

- Ensure session attribute are there eg. Http-Only, SameSite.

Reject Unexpected Content Types

- For endpoints not meant for form submissions, reject application/x-www-form-urlencoded and multipart/form-data unless needed.

3. Brute Force Leading to Account Takeover	
Severity	High
Status	Unresolved
Risk Score	8.5/10
CWE	CWE-307: Improper Restriction of Excessive Authentication Attempts
CVSS	8.5 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N)
Labels	Brute Force Vulnerability, Weak Authentication Controls, Account Takeover Risk, Improper Access Control

Description

The login functionality is vulnerable to brute-force attacks due to the absence of mechanisms like rate limiting, IP blocking, user account lockouts, or CAPTCHA. An attacker can automate repeated login attempts using common tools (e.g., Burp Suite Intruder, Hydra) to guess valid usernames and passwords.
The backend accepts unlimited login attempts without any delay or restriction, which could result in unauthorized access and full account takeover, especially if users have weak or reused passwords.

Impact

- **Account Takeover:** Attackers can compromise user accounts, especially admin or privileged ones.
- **User Enumeration:** Clear login error messages help identify valid usernames.
- **Credential Stuffing Risk:** Common reused credentials can be rapidly tested.
- **Compliance Violation:** Absence of brute-force protection may violate security policies (e.g., OWASP ASVS, NIST).
- **Business Risk:** Loss of user trust, reputation damage, potential data breaches.

Proof Of Concept

```
module.exports.login = (req, res) => {
    req.flash('success', 'Welcome back!');
    const redirectUrl = res.locals.returnTo || '/campgrounds'; // up
    res.redirect(redirectUrl);
}
```

```
router.route('/login')
    .get(users.renderLogin)
    .post(storeReturnTo, passport.authenticate('local', {failureFlash: true, failureRedirect: '/login'}), users.login)
```

Suggested Fix

Implement Rate Limiting

- Use middleware like express-rate-limit to restrict login attempts

Add Account Lockouts / Delays

- Lock user accounts temporarily after X failed login attempts.
- Introduce incremental delay (setTimeout) between failures.

Use CAPTCHA or 2FA

- Trigger CAPTCHA after a threshold of failed attempts.
- Enforce 2FA for critical accounts.

Generic Error Messages

- Avoid revealing if username or password is incorrect

Log and Alert on Abnormal Login Behavior

- Log all failed attempts.
- Flag accounts with too many failures for review.

4. Possible XSS due to inadequate sanitization of Bootstrap	
Severity	High
Status	Unresolved
Risk Score	8.2 /10
CWE	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CVSS	8.2 (AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:H/A:N)
Labels	XSS, Bootstrap, Library-Vulnerability, CVE-2024-6531

Description

The application includes Bootstrap v4.5.0, which is affected by a known Cross-Site Scripting (XSS) vulnerability identified as CVE-2024-6531. This vulnerability arises from inadequate sanitization in Bootstrap's internal tooltip and popover components when used with user-controlled data.
If untrusted content is injected into attributes like data-title, data-content, or similar tooltip/popover content fields without proper HTML sanitization, attackers can inject arbitrary JavaScript. This leads to stored or reflected XSS depending on context.

Impact

- Arbitrary JavaScript execution in users' browsers
- Session hijacking, account takeover, or phishing
- Exfiltration of sensitive data, such as CSRF tokens or cookies (if not HttpOnly)
- Reputation damage or compliance violations (e.g., OWASP Top 10 A3:2021 – Injection)

Proof Of Concept

```
            <!-- Include the validateForm.js script -->
            <script src="/javascripts/validateForm.js"></script>

            <!-- jQuery library -->
            <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

            <!-- Bootstrap JavaScript -->
            <script src="/vendor/bootstrap/js/bootstrap.min.js"></script>

            <!-- Include the Bootstrap JS library from a CDN -->
            <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"></script
>


            <!-- Include the Bootstrap JS library from a CDN -->
            <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
                integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4H0zUF0QKbrJ0EcQF"
                crossorigin="anonymous"></script>
```

```
<div id="myCarousel" class="carousel"></div>
<a href="javascript:alert('XSS href')" data-slide="prev">
  Previous Slide
</a>
```

Suggested Fix

Upgrade Bootstrap

- Update to the latest patched version: Bootstrap ≥ v4.6.3 or ≥ v5.3.3, which fixes CVE-2024-6531.

Sanitize All User-Controlled Inputs in Attributes

- Avoid directly inserting user input into title, data, or innerHTML.
- Use libraries like DOMPurify to sanitize content

Disable HTML

- True in Bootstrap Tooltip/Popover if not required

Perform Output Encoding on the Server Side

- Escape user input before rendering it in templates using appropriate functions (e.g., ejs, pug, handlebars auto-escape mechanisms).

5. Weak password policy	
Severity	Medium
Status	Unresolved
Risk Score	6.2/10
CWE	CWE-521: Weak Password Requirements
CVSS	6.2 (AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:N)
Labels	Authentication, Bruteforce, Weak Policy

Description

The application currently allows users to set weak passwords without enforcing minimum complexity requirements. There is no validation for password length, character variety (uppercase, lowercase, numeric, symbols), or protection against commonly used passwords (e.g., 123456, password, admin).
This significantly lowers the entropy of stored credentials and increases the risk of account compromise through password guessing, brute-force, or credential stuffing attacks.

Impact

- **Account Takeover**: Weak passwords can be easily guessed or cracked using automated tools.
- **Brute-force Attack Surface**: Combined with missing rate limits or MFA, attackers can target multiple users at scale.
- **Compliance Issues**: Fails to meet security standards like OWASP ASVS, NIST 800-63B, ISO 27001, and PCI DSS.
- **Privilege Escalation Risk**: If an admin account is protected by a weak password, the entire application could be compromised.

Proof Of Concept

```
module.exports.register = async (req, res) => {
    try {
        const { email, username, password } = req.body
        const user = new User({ email, username })
        const registeredUser = await User.register(user, password)
        req.login(registeredUser, err => {
            if (err) return next(err)
            req.flash('success', 'Welcome to YelpCamp!')
            res.redirect('/campgrounds')
        })
    } catch (e) {
        req.flash('error', e.message)
        res.redirect('/register')
    }
}
```

```
router.route('/register')
    .get(users.renderRegister)
    .post(catchAsync(users.register))
```

Suggested Fix

Enforce Strong Password Policy

- Minimum length: ≥ 8 or 12 characters
- At least one lowercase, uppercase, digit, and symbol
- Reject commonly used passwords (use a blacklist or dictionary)

Integrate Password Blacklist Checks:

- Use datasets like HaveIBeenPwned to detect weak patterns.

Educate Users on Password Strength:

- Provide real-time strength indicators during registration/reset.
- Recommend use of passphrases (e.g., ThreeHorseBatteries!).

Encourage MFA (Multi-Factor Authentication):

- Even strong passwords can be compromised. MFA adds a critical second layer of security.

Summery

The assessment revealed 4 High and 1 Medium severity vulnerabilities that could lead to potential security breaches. Immediate remediation of high-risk vulnerabilities is strongly recommended to mitigate exploitation risks.
It is strongly advised to fix all High vulnerabilities within the next 7-14 days and conduct a follow-up security assessment to verify remediation effectiveness.

References & Security Best Practices

- OWASP Secure Coding Practices
- OWASP ASVS (Application Security Verification Standard)
- Node.js Security Best Practices (OWASP + Open Source)
- Content Security Policy (CSP) Guide
- OWASP Code Review Guide v2
- Google's Secure Coding Guidelines

About the Security Penetration Tester

This assessment was conducted by Satvik Hatulkar, specializing in penetration testing, vulnerability assessment. For further assistance or follow-ups, please contact me at satwikhatulkar@gmail.com.