**Target:**http://117.72.165.13:8888/

**Github:**https://github.com/201206030/novel

**Version:**V3.5.0

**Vulnerability Name:**DOM-based Cross-Site Scripting (XSS) Vulnerability in novel V3.5.0 (CWE-79)

**Vulnerability Type:** DOM-based Cross-Site Scripting (DOM-based XSS)

**Discovery Date:** 2025-11-02

**Vulnerability Details:**

novel (V3.5.0), a Spring Boot 3 + Vue 3-based novel reading platform, has a DOM-based XSS vulnerability in its book comment module. The flaw lies in inadequate validation/encoding of user-submitted comment content:

Back-end: The commentContent field (in BookComment entity/DTO) is stored in the database and returned via API without filtering malicious HTML/JS code;

Front-end: The platform retrieves unencoded comments, stores them in window.localStorage (key: book\_comment\_{bookId}), and renders the data directly into the DOM via Vue 3’s v-html (no sanitization).

Attackers can submit malicious comments (after logging in), induce users to access affected pages, and trigger code execution to steal session cookies, impersonate identities, or tamper with page content.

**Technical Details & Proof of Concept (PoC):**

**Preconditions：**

Target version: novel V3.5.0

Environment: Chrome/Firefox, legitimate user account (login required)

Affected URL: Novel detail page (http://117.72.165.13:8888/book/detail.html?id={bookId}) and comment list page (http://117.72.165.13:8888/book/comment-{bookId}.html)

**Vulnerability Trigger Point:**

The vulnerability is triggered in the book comment module (core business logic):

Backend: The commentContent field in the BookComment entity/DTO is stored in the book\_comment database table and returned via the comment list API (/api/book/comment/list?bookId={bookId}) without filtering malicious HTML/JS code.

Frontend: The page retrieves unencoded comment content from the API, stores it in window.localStorage (key: book\_comment\_{bookId}, e.g., book\_comment\_37 for book ID 37), and renders the data directly into the DOM via Vue 3’s v-html directive (inserted into <li class="dec"> tag) without sanitization.

**Example Attack Payloads:**

The following payloads can verify the existence of the vulnerability:

1.Store malicious code into window.localStorage via URL parameters

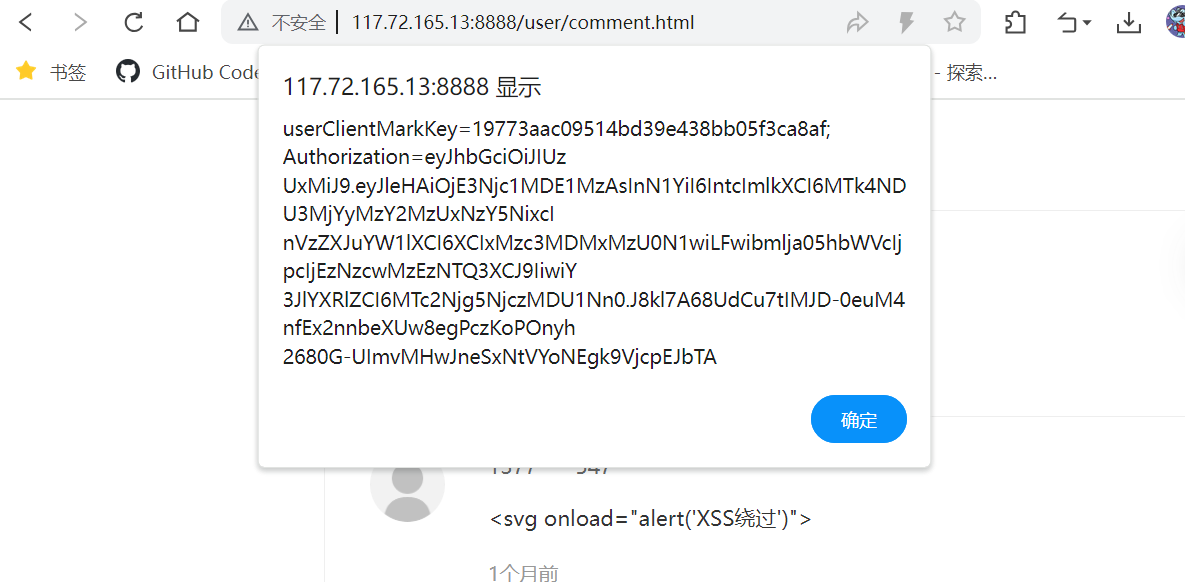
http://117.72.165.13:8888/book/1935650139770011648/1984484631582167040.html?switch=on&wvstest=<svg onload="alert('DOM-XSS-Test')">

(The page will store the value of the wvstest parameter into window.localStorage)

2.Directly inject malicious data into window.localStorage via front-end scripts

localStorage.setItem('book\_comment\_37', '<img src=x onerror=alert(document.cookie)>');





**Reproduction Steps：**

**1.**Login to the system: Access http://117.72.165.13:8888/user/login.html, log in with a legitimate account, and navigate to the novel detail page.

**2.**Submit malicious comment:

Locate the comment textarea (id="txtComment") on the page, input the malicious payload (<script>alert(document.cookie)</script>).

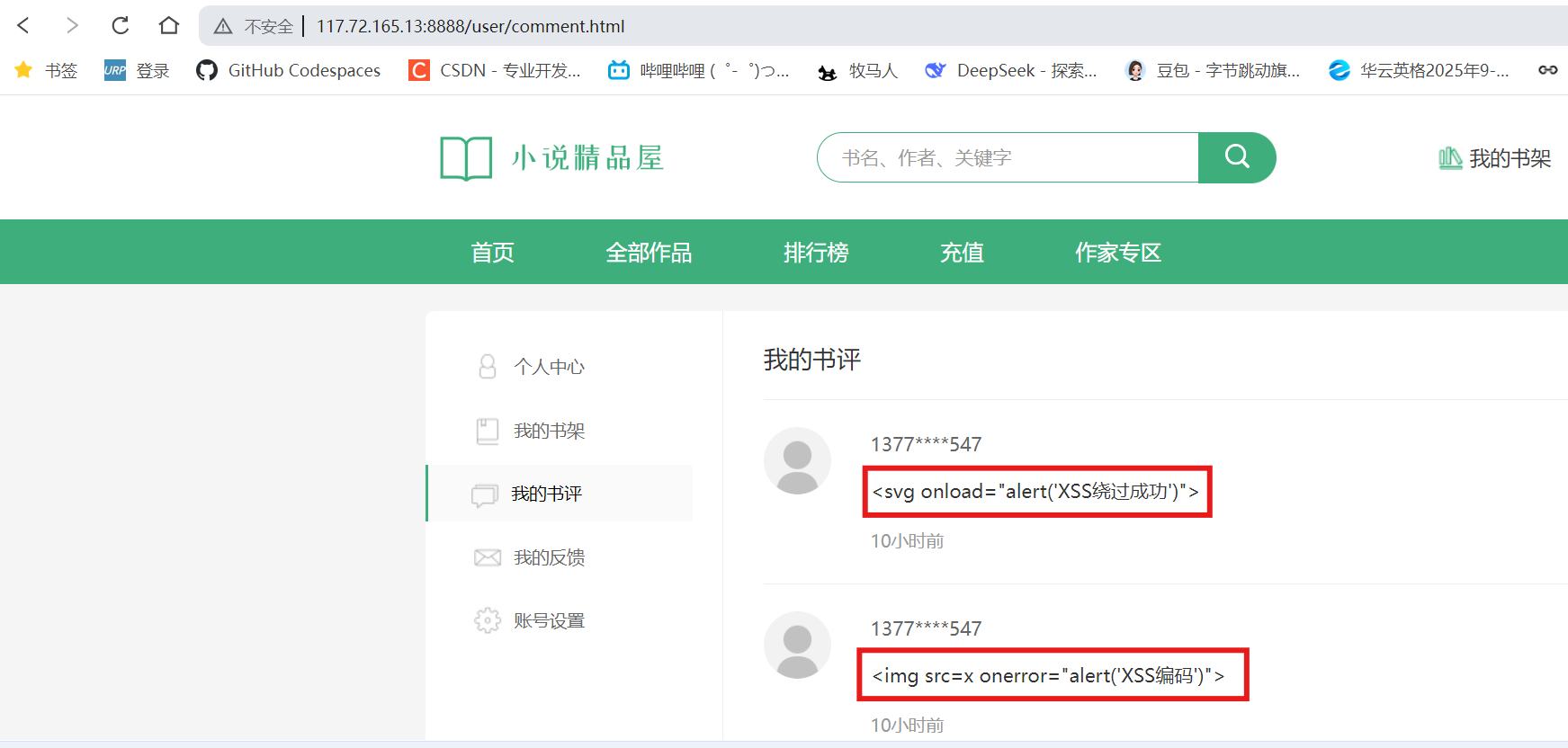
Click the "Submit" button to trigger the BookDetail.SaveComment(37, 0, payload) method (37 is the bookId).

**3.**Trigger localStorage storage:

Refresh the novel detail page; the front-end calls /api/book/comment/list?bookId=37 to retrieve the malicious comment, and stores it in window.localStorage (key: book\_comment\_37).

**4.**Verify XSS execution:

Observe the page pops up a window displaying the user’s session cookie (confirm malicious code execution).

Press F12 to open Developer Tools Switch to 「Elements」 panel, search for <script>alert(document.cookie)</script> in the DOM structure (confirm the payload is inserted into<liclass="dec">tag).



**Reporter Information**

Name/Organization:shuang li

Contact Information: 3074597669q@gmail.com