

Sri Lanka Institute of Information Technology



Bug Bounty Report - 07

Module: IE2062

Web Security

Year 2, Semester 2

Aazaf Ritha. J – IT23151710

B.Sc. (Hons) in Information Technology

Specialized in Cyber Security

Table of Contents

Introduction	3
Vulnerability Title - 01	4
Description.....	5
Affected Component.....	6
Impact Assessment	7
Steps to Reproduce.....	8
Proof of Concept (Screenshots)	9
Proposed Mitigation or Fix.....	11
Vulnerability Title – 02.....	12
Description.....	13
Affected Component.....	14
Impact Assessment	15
Steps to Reproduce.....	16
Proof of Concept (Screenshots)	17
Proposed Mitigation or Fix.....	19
Conclusion.....	20

Introduction

This report outlines a set of security misconfigurations and vulnerabilities discovered on Greenfly's public website (<https://www.greenfly.com>) as part of active reconnaissance and vulnerability testing conducted through the HackerOne bug bounty program. The issues were identified using a combination of manual testing techniques and automated tools such as OWASP ZAP. This document details each vulnerability with technical context, clear steps to reproduce, proof-of-concept evidence, associated CVEs (where applicable), and practical recommendations for remediation. The goal of this report is to support Greenfly's security team in improving the overall security posture of their web infrastructure.

Vulnerability Title - 01

Title: Use of Vulnerable JavaScript Library (pdf.js)

Risk Level: High

Domain: <https://www.greenfly.com>

Description

Greenfly's website includes a vulnerable version of pdf.js (v2.11.338) loaded from the WordPress plugin directory. This version is associated with [CVE-2024-4367](#) and could allow attackers to exploit client-side vulnerabilities like XSS or memory corruption. Including outdated libraries introduces unnecessary risk and violates best practices outlined in **OWASP A06:2021 – Vulnerable and Outdated Components**.

Affected Component

pdf.js Library (v2.11.338):

This version is known to contain vulnerabilities (e.g., CVE-2024-4367) and is included in the site's frontend resources. Its presence poses a risk if it handles user-generated content or is used in dynamic rendering.

WordPress Plugin – pdf-viewer-for-wordpress:

The vulnerable library is bundled with this plugin, indicating either outdated dependencies or lack of plugin updates.

URL Path:

<https://www.greenfly.com/wp-content/plugins/pdf-viewer-for-wordpress/build/pdf.js?ver=11.9.0>

Impact Assessment

Client-Side Exploitation: Vulnerabilities in the outdated pdf.js library could allow attackers to run malicious scripts in users' browsers.

Cross-Site Scripting (XSS): If user input interacts with the PDF rendering, an attacker might inject scripts that compromise user sessions or steal sensitive data.

Memory and DOM Manipulation Risks: Known issues in the version used could lead to browser crashes, denial-of-service conditions, or malicious UI changes.

Increased Attack Surface: Including outdated libraries with known CVEs makes the application a more attractive and easier target for attackers.

User Trust and Reputation Risk: Security-conscious users and tools may flag or block the site, harming trust and credibility.

Steps to Reproduce

Launch **OWASP ZAP** and configure your browser to proxy traffic through ZAP (typically 127.0.0.1:8080).

In browser (proxied through ZAP), navigate to:

<https://www.greenfly.com>

Let ZAP passively scan the site as you browse.

In ZAP, go to the **Alerts** tab and look for a high-severity issue titled:

"Vulnerable JS Library - pdf.js 2.11.338"

Double-click the alert to view the details. ZAP will report the exact file URL:

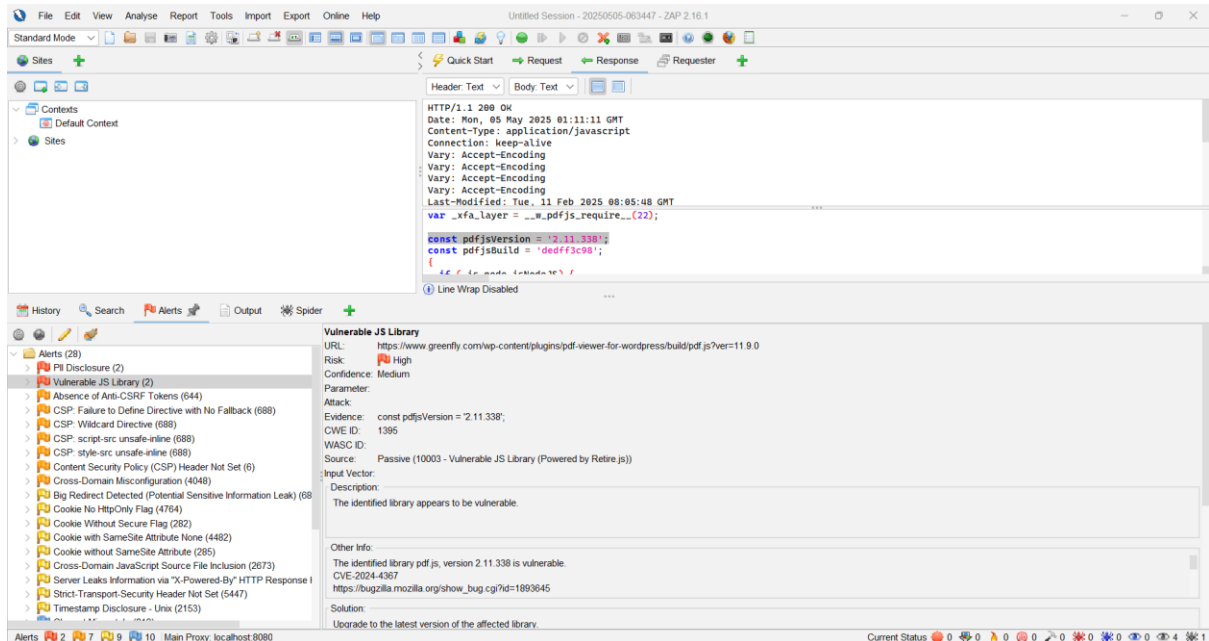
<https://www.greenfly.com/wp-content/plugins/pdf-viewer-for-wordpress/build/pdf.js?ver=11.9.0>

Verify the version and check the CVE reference in the alert, which typically points to:

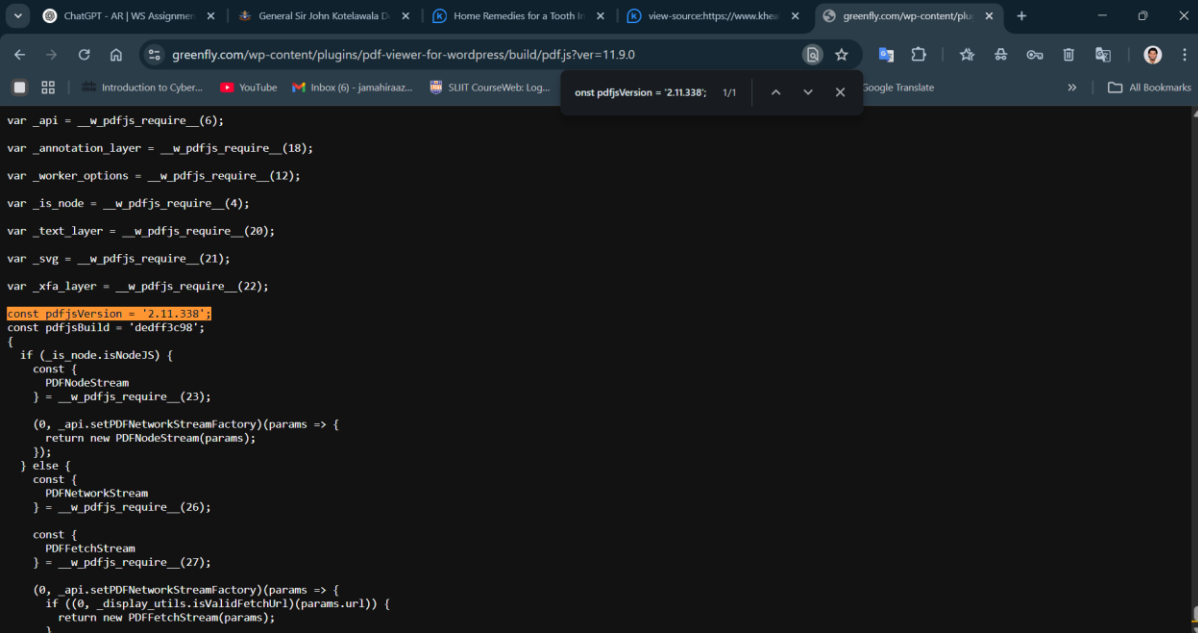
[CVE-2024-4367](#)

Proof of Concept (Screenshots)

OWASP ZAP Scan



The attached OWASP ZAP scan screenshot highlights a high-risk vulnerability on Greenfly's website related to the use of an outdated JavaScript library—pdf.js version 2.11.338. The library is served via the WordPress plugin pdf-viewer-for-wordpress from the URL <https://www.greenfly.com/wp-content/plugins/pdf-viewer-for-wordpress/build/pdf.js?ver=11.9.0>. ZAP, using its Retire.js integration, has flagged this version as vulnerable based on the known issue CVE-2024-4367. Additionally, the raw JavaScript code in the HTTP response confirms the version via the line `const pdfjsVersion = '2.11.338';`. The presence of this outdated library in a production environment poses a security risk and may allow for client-side exploitation such as XSS or memory-related attacks.



```

var _api = __w_pdfjs_require__(6);
var _annotation_layer = __w_pdfjs_require__(18);
var _worker_options = __w_pdfjs_require__(12);
var _is_node = __w_pdfjs_require__(4);
var _text_layer = __w_pdfjs_require__(20);
var _svg = __w_pdfjs_require__(21);
var _xfa_layer = __w_pdfjs_require__(22);
const pdfjsVersion = '2.11.338';
const pdfjsBuild = 'dedff3c98';
{
  if (_is_node.isNodeJS) {
    const {
      PDFNodeStream
    } = __w_pdfjs_require__(23);

    (0, _api.setPDFNetworkStreamFactory)(params => {
      return new PDFNodeStream(params);
    });
  } else {
    const {
      PDFNetworkStream
    } = __w_pdfjs_require__(26);

    const {
      PDFFetchStream
    } = __w_pdfjs_require__(27);

    (0, _api.setPDFNetworkStreamFactory)(params => {
      if ((0, _display_utils.isValidFetchUrl)(params.url)) {
        return new PDFFetchStream(params);
      }
    });
  }
}

```

The screenshot captures a portion of the pdf.js source file served by Greenfly's website at <https://www.greenfly.com/wp-content/plugins/pdf-viewer-for-wordpress/build/pdf.js?ver=11.9.0>. It clearly shows the declaration `const pdfjsVersion = '2.11.338';`, confirming that the site is using an outdated and vulnerable version of the pdf.js library. This version has known security flaws, including those referenced in CVE-2024-4367. The direct visibility of this version within a publicly accessible script verifies the issue without the need for scanning tools, supporting the claim that the application is at risk due to reliance on insecure third-party components.

Proposed Mitigation or Fix

Update to Latest Secure Version: Upgrade pdf.js to the most recent stable and secure version that is free of known CVEs.

Monitor Third-Party Dependencies: Regularly audit JavaScript libraries using tools like Snyk, npm audit, or Dependabot to catch outdated/vulnerable versions.

Test Plugin Compatibility: After upgrading the plugin or library, test thoroughly to ensure functionality is not broken, especially for any PDF rendering.

Limit User-Controlled Input to PDF Renderer: If user input is passed to pdf.js, apply strict validation and sanitization to prevent injection or XSS vectors.

Vulnerability Title – 02

Title: PII Exposure in HTTP Response (Credit Card Detected)

Risk Level: High

Domain: <https://www.greenfly.com/blog/mlb-fan-engagement/>

Description

A valid-looking Mastercard number (2681 5349 8541 4515) was found embedded in the HTTP response of a public blog post. This indicates possible leakage of sensitive backend data, either through improper sanitization or leftover debug content. It falls under **OWASP A01:2021 – Broken Access Control** and **A04:2021 – Insecure Design**, due to improper handling of sensitive information.

Affected Component

HTTP Response Body:

The credit card number (2681534985414515) is present in the raw response sent to any visitor, suggesting a failure in output sanitization or data filtering before rendering.

Blog Page Endpoint:

<https://www.greenfly.com/blog/mlb-fan-engagement/>

The data is exposed in a static or dynamically rendered section of this blog post, possibly due to embedded scripts or unfiltered content blocks.

Backend/Template Layer:

The presence of sensitive data points to an issue in backend logic or content templates that failed to remove or obfuscate PII before sending data to the frontend.

Impact Assessment

Sensitive Data Disclosure: A credit card number found in a public response could be misused for fraudulent transactions.

Regulatory and Legal Violations: Exposing PII, especially financial data, violates standards like PCI-DSS and laws such as GDPR or CCPA.

Risk of Identity Theft or Fraud: Attackers could harvest the information for social engineering, phishing, or unauthorized purchases.

Lack of Data Sanitization: Indicates potential backend flaws or insecure coding practices where debugging or sensitive data is not properly filtered out.

Reputational and Financial Damage: Such leaks can lead to user distrust, negative press, and potential fines or lawsuits.

Steps to Reproduce

Open OWASP ZAP and set up the browser proxy as before.

In the browser, visit:

<https://www.greenfly.com/blog/mlb-fan-engagement/>

Let ZAP capture and analyze the HTTP response.

In ZAP, navigate to the Sites tab and locate the page under:

<https://www.greenfly.com> > </blog/mlb-fan-engagement/>

Right-click the entry and select "Show Response in Browser" or view the response in the Response tab.

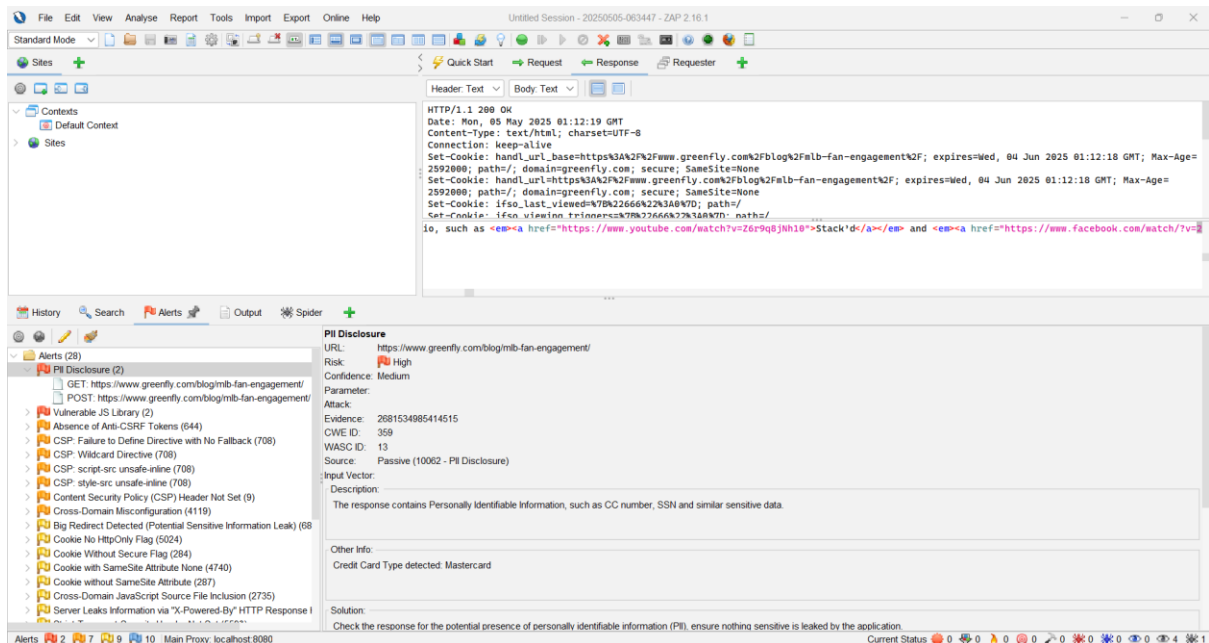
Search for the value 2681534985414515 or use the Alerts tab to locate a high-severity alert:

"Private IP Disclosure / PII Disclosure"

Confirm that a 16-digit number matching a Mastercard format is visible in the raw HTTP body, accessible without authentication.

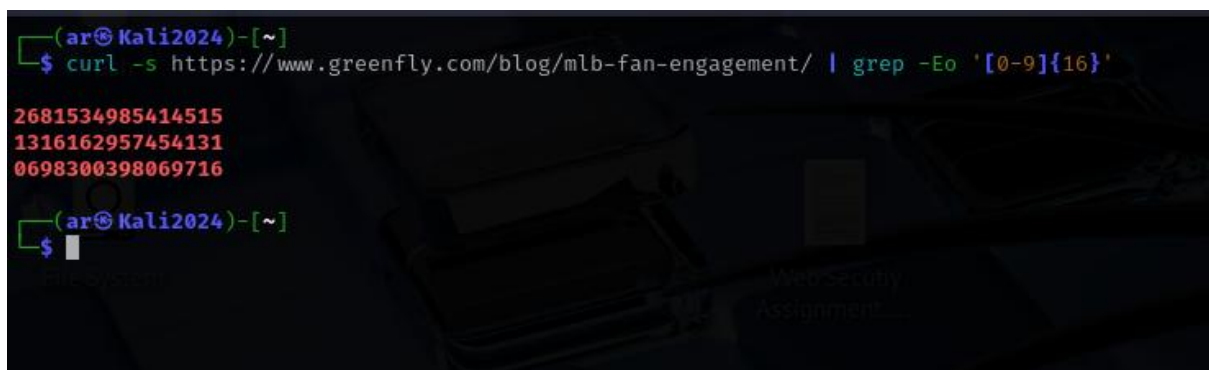
Proof of Concept (Screenshots)

OWASP ZAP Scan



The OWASP ZAP scan has identified a high-risk **Personally Identifiable Information (PII) Disclosure** vulnerability on the Greenfly blog page at <https://www.greenfly.com/blog/mlb-fan-engagement/>. The ZAP alert highlights that a **Mastercard-format credit card number (2681534985414515)** was found within the HTTP response body. This was detected using passive scanning under the alert category 10062 - PII Disclosure. The presence of such sensitive data in a publicly accessible response indicates a failure in data sanitization or improper exposure of backend data. If this is real customer data, it constitutes a severe violation of **PCI-DSS** and data privacy regulations like **GDPR**, posing risks of financial fraud and legal liability.

In Terminal (With curl)



The screenshot presents a command-line proof of concept confirming the exposure of sensitive numerical data on Greenfly's blog page (<https://www.greenfly.com/blog/mlb-fan-engagement/>). By using curl utility to fetch the raw HTML response and applying a regular expression with grep, multiple 16-digit numbers were extracted. One of these numbers, 2681534985414515, conforms to the structure of a valid Mastercard credit card number.

Proposed Mitigation or Fix

Immediately Remove Exposed Data: Eliminate the credit card number from the page or backend response.

Review Backend and Templates: Audit the backend logic and template rendering pipeline for leftover debug data, hardcoded test values, or database leaks.

Implement Output Sanitization: Sanitize all data before it is rendered or exposed to clients, especially in public-facing content.

Enable Data Loss Prevention (DLP): Deploy DLP tools or regex-based filters to catch sensitive data (e.g., credit card numbers) before it reaches production.

Perform Full Security Review: Conduct a deeper review to ensure no other sensitive data is being leaked across other pages or endpoints.

Conclusion

The vulnerabilities identified on Greenfly’s public website—specifically the use of an outdated and vulnerable JavaScript library (pdf.js) and the exposure of sensitive PII in an HTTP response—pose significant security and compliance risks. Left unaddressed, these issues could lead to client-side attacks, data breaches, or regulatory violations.

Proactive remediation steps such as updating third-party components, auditing content delivery pipelines, and implementing strict data sanitization can substantially reduce the attack surface. These findings highlight the importance of continuous security testing and dependency monitoring in maintaining a secure web environment.

Addressing these issues promptly will help strengthen Greenfly’s security posture, protect user trust, and ensure compliance with industry standards.