Stored XSS via Malicious SVG Upload Lab

1. Introduction

CVE-2026-5026 is a recently disclosed vulnerability (2026) that highlights a critical security issue in modern web applications: the unsafe handling of **SVG (Scalable Vector Graphics) files** during upload and rendering.
SVG files are **XML-based documents** that can contain active content, such as JavaScript code and event handlers, in contrast to conventional picture formats like JPEG or PNG. These files may serve as a vector for **Cross-Site Scripting (XSS)** attacks if they are not properly verified.
In affected systems, applications allow users to upload SVG files and subsequently render them in the browser without sanitizing their contents. As a result, attackers are able to inject malicious scripts inside the SVG file. When another user or an administrator views the uploaded image, the embedded script is run within the web application.
This vulnerability is classified as **Stored XSS**, since the malicious payload is permanently stored on the server and automatically executed when accessed. Such attacks may have serious consequences, including:

- Session hijacking through cookie theft;
- Sensitive data exposure;
- Full compromise of user accounts (especially administrative accounts);

CVE-2026-5026 reflects a broader class of vulnerabilities where developers mistakenly treat all image formats as passive content. It emphasizes the importance of **strict input validation, content sanitization, and secure file handling practices** in web applications.
The goal of this lab is to demonstrate how this vulnerability can be exploited in practice. By completing it, students will:

- Understand how XSS attacks work in non-traditional contexts;
- Learn how file upload features can introduce security risks;
- Perform a real attack to extract sensitive data (flag);
- Propose and implement mitigation strategies.

2. Overview

This lab provides a hands-on exploration of Stored XSS vulnerabilities through the lens of a realistic file upload scenario. It is structured in three main phases: understanding the problem, exploiting the vulnerability, and studying mitigations.

Understanding the problem. We begin by examining how XSS attacks work, focusing on the stored variant where the payload is persisted on the server and triggered automatically upon access. We then study why SVG files represent a

non-obvious but highly effective attack vector for XSS exploitation. Unlike JPEG or PNG files, SVGs are XML documents that browsers parse and execute, meaning any JavaScript embedded within them runs in the context of the hosting page. Students will analyze the structure of a malicious SVG payload and understand the trust boundary that is violated when such a file is uploaded and served without sanitization.

Exploiting the attack. Students will interact with a deliberately vulnerable web application that allows SVG file uploads. The lab tasks escalate in complexity:

- First, students will upload a crafted SVG that executes a simple JavaScript alert, confirming that script execution is possible.
- Next, they will refine the payload to extract the session cookie of any user who views the uploaded file, exfiltrating it to an attacker-controlled listener, demonstrating a realistic session hijacking scenario.
- Finally, students will craft a payload that performs actions on behalf of the victim (e.g., sending an HTTP request that modifies application state), simulating a full account compromise without requiring the victim's credentials.

Understanding the solution. The lab concludes by presenting the main mitigation strategies used to prevent SVG-based XSS vulnerabilities and discussing how they are applied in real-world systems. Applications should avoid accepting SVG files when not necessary or sanitize them to remove scripts and event handlers. Uploaded files should be served safely, preventing inline execution in the browser. Additionally, enforcing a strict Content Security Policy (CSP) and implementing proper input validation on both client and server sides helps reduce the risk of exploitation.

Lab Environment. The lab can be conducted using a web application running locally (e.g., Flask server). Students may use browser developer tools and a simple HTTP listener (e.g., netcat or Python HTTP server).

3. Lab Tasks

3.1 Task 1: Executing a Basic XSS Attack

Students must upload a malicious SVG file that triggers a JavaScript alert when viewed.

3.2 Task 2: Stealing Session Cookies

Students modify the SVG payload to extract the victim’s session cookie to an attacker-controlled server.

3.3 Task 3: Performing Actions on Behalf of the Victim

Students craft a payload that sends an authenticated request, demonstrating account compromise.