Stored Cross-Site Scripting in Samarium - Product Category SVG Upload (v0.9.6)

Repository: https://github.com/oitcode/samarium

CVSS v4.0

Score: 4.8
Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:H/UI:P/VC:N/VI:L/VA:N/SC:N/SI:N/SA:N/E:P

Summary

A stored cross-site scripting (XSS) condition exists in Samarium v0.9.6 related to SVG images uploaded during the Product Category creation process. The application stores SVG files without sufficiently filtering SVG content. If a high-privileged user (for example, an administrator or other role with rights to create product categories) uploads a malicious SVG, the embedded script can execute when another user intentionally opens the image resource in a browser.

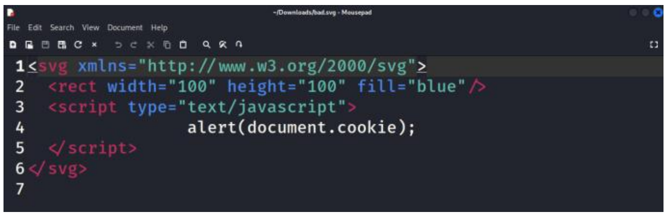
Vulnerability Description

When creating or editing product categories, the web interface accepts an image field that allows SVG uploads. The server persists the uploaded SVG without performing content sanitization that would remove script elements or inline event handlers. An uploaded SVG containing script content can therefore be served back to users and executed by their browsers if they open the SVG resource.

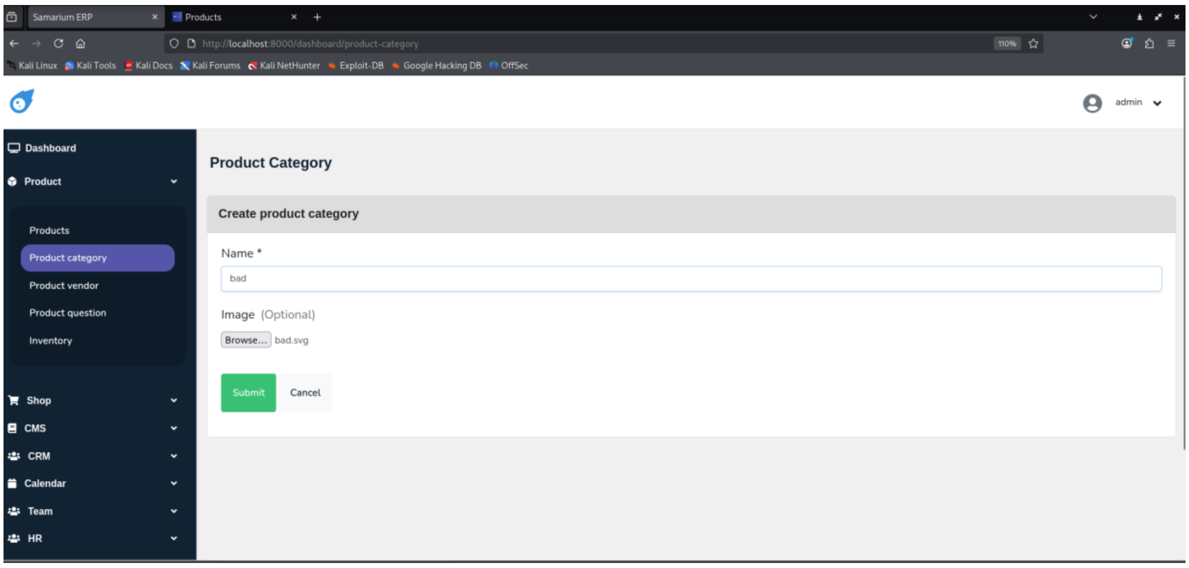
Proof of Concept

Step 1: Upload a malicious SVG while creating or editing a product category

Upload the following SVG as the category image:


~/Downloads/bad.svg - Mousepad
File Edit Search View Document Help

```
<svg xmlns="http://www.w3.org/2000/svg">
  <rect width="100" height="100" fill="blue"/>
  <script type="text/javascript">
              alert(document.cookie);
  </script>
</svg>
```



http://localhost:8000/dashboard/product-category

Product Category

Create product category

Name *
bad

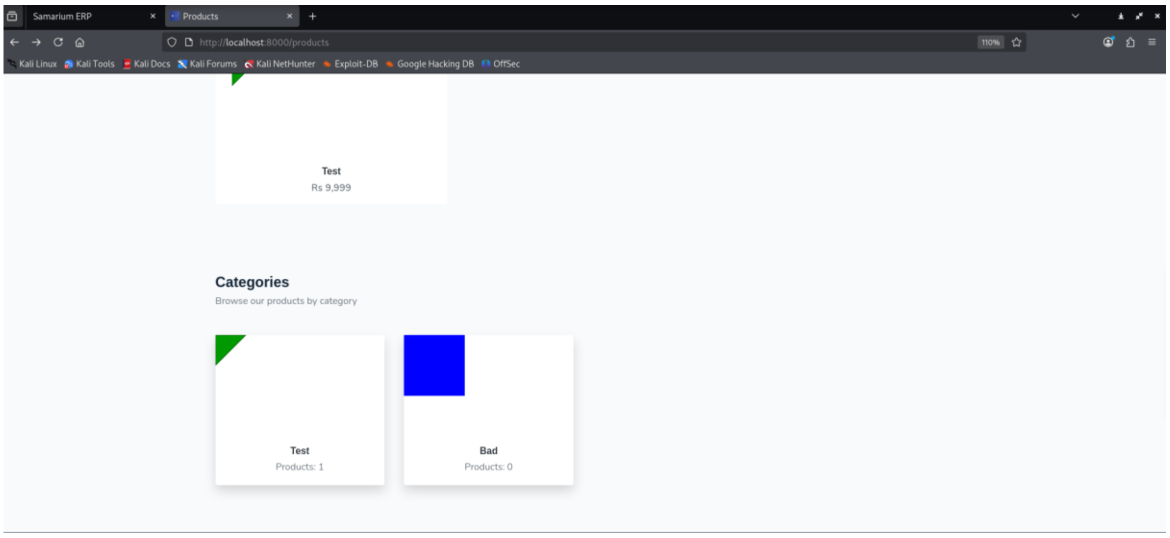
Image (Optional)
Browse... bad.svg

Submit Cancel

The file is accepted and stored on the server without modification.

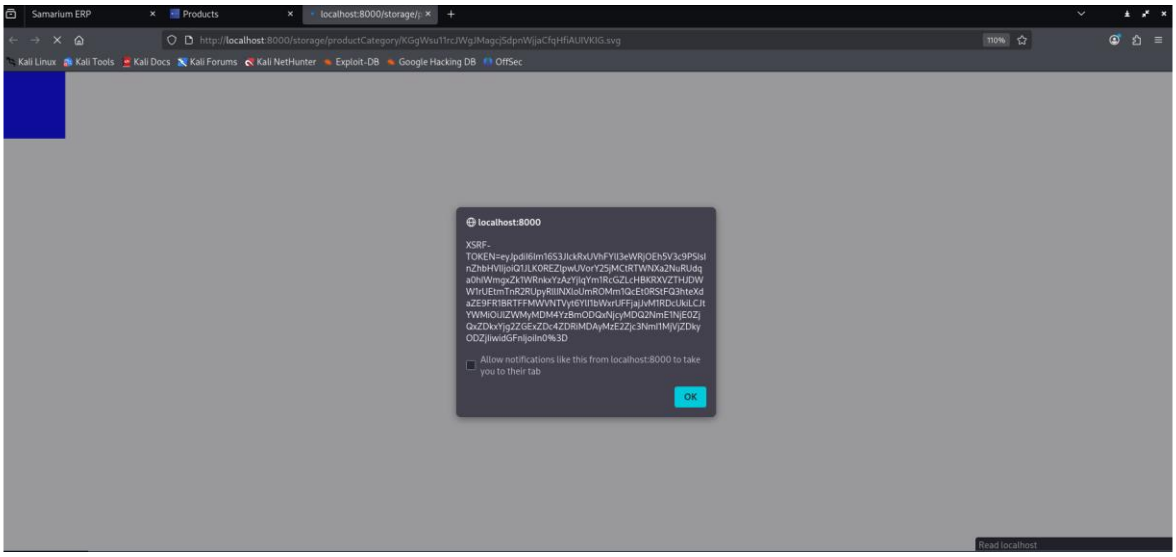
Step 2: Navigate to the product display page

Go to the product display area. Find the newly created category entry and locate its image.



Step 3: Access the image resource

Open the category image in a browser tab.
The embedded JavaScript executes, allowing theft of session cookies or any other action within the session context.



localhost:8000

```
XSRF-
TOKEN=eyJpdiI6Im16S3JlckRxUVhFYlI3eWRjOEh5V3c9PSIsI
nZhbHVlIjoiQ1JLK0REZlpwUVorY25jMCtRTWNXa2NuRUdq
a0hlWmgxZk1WRnkxYzAzYjlqYm1RcGZLcHBKRXVZTHJDW
W1rUEtmTnR2RUpyRllINXloUmROMm1QcEt0RStFQ3hteXd
aZE9FR1BRTFFMWVNTVyt6YlI1bWxrUFFjajJvM1RDcUkiLCJt
YWMiOiJlZWMyMDM4YzBmODQxNjcyMDQ2NmE1NjE0Zj
QxZDkxYjg2ZGExZDc4ZDRiMDAyMzE2Zjc3NmI1MjVjZDky
ODZjIiwidGFnIjoiIn0%3D
```

Allow notifications like this from localhost:8000 to take you to their tab

OK

Impact

- Stored XSS triggered whenever the malicious SVG is rendered
- Session cookies can be exfiltrated to an attacker
- Exploitation requires an account with elevated privileges to upload or modify product category images (e.g., a user with create category permission)
- Administrators and other users who view or open the stored image can be affected
- May lead to account takeover or other malicious activity

- This vulnerability is considered medium severity

Remediation

- Reject SVG uploads entirely, or
- Sanitize SVG content server-side, removing <script>, <foreignObject>, and event handlers
- Enforce strict MIME type and file-content validation
- Serve user-uploaded content from a separate domain or force download (e.g., Content-Type: application/octet-stream)
- Deploy a Content Security Policy to prevent execution of untrusted scripts