Stored XSS in Attribute View gallery/kanban cover rendering allows arbitrary command execution in the desktop client

Critical **88250** published **GHSA-rx4h-526q-4458** last week

Package

github.com/siyuan-note/siyuan/kernel (Go)

Affected versions	Patched versions
3.6.1	v3.6.2

Description

Summary

An attacker who can place a malicious URL in an Attribute View `mAsse` field can trigger stored XSS when a victim opens the Gallery or Kanban view with “Cover From -> Asset Field” enabled. The vulnerable code accepts arbitrary `http(s)` URLs without extensions as images, stores the attacker-controlled string in `coverURL`, and injects it directly into an `<img src="...">` attribute without escaping. In the Electron desktop client, the injected JavaScript executes with `nodeIntegration` enabled and `contextIsolation` disabled, so the XSS reaches arbitrary OS command execution under the victim’s account.

Details

The vulnerable flow is:

1. `IsPossiblyImage(assetPath)` accepts arbitrary `http(s)` URLs without validating that they are safe image URLs.
2. When an Attribute View card uses `Cover From -> Asset Field`, the application copies `asset.Content` directly into `galleryCard.CoverURL / kanbanCard.CoverURL`.
3. The front-end renderer inserts `coverURL` directly into `<img src="${getCompressURL(item.coverURL)}">` without escaping quotes or other attribute-breaking

characters.

4. A payload such as `https://example.com/" onerror="require('child_process').exec('calc')` breaks out of the `src` attribute and adds an attacker-controlled `onerror` handler.
   When the image fails to load, the injected JavaScript runs in the Electron renderer. Because the desktop app enables `nodeIntegration: true` and disables `contextIsolation` and `webSecurity`, that JavaScript can access Node.js APIs and execute system commands.

PoC

1. Install Electron Desktop app.
2. Create a database / Attribute View with an mAsset column and add at least one row.
3. Add any legitimate image to that mAsset field so the entry is stored as type image.
4. Switch the view to Gallery or Kanban.
   5.Set Cover From to Asset Field and choose the mAsset column.
5. Edit the existing image asset entry and replace its link with the following payload:

```
https://example.com/" onerror="require('child_process').exec('calc')
```

7. Save the change and reopen or refresh the Gallery / Kanban view.
8. Observe that the rendered HTML contains an injected onerror handler and the Calculator application starts on Windows.

Example rendered output:

```
<img loading="lazy" class="av__gallery-img" src="https://example.com/" onerror="re (
```

Impact

An attacker can store malicious content in a database asset field and execute arbitrary JavaScript when another user opens the affected Gallery or Kanban view. In the desktop client, that JavaScript has access to Node.js APIs, so the impact is not limited to browser-context XSS. The payload executes OS commands with the victim's local user privileges, which turns this into remote code execution on the desktop application once the malicious content is delivered and rendered.

Severity

Critical 9.1 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low

Privileges required	Low
User interaction	Required
Scope	Changed
Confidentiality	High
Integrity	High
Availability	High

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:H

CVE ID

CVE-2026-34448

Weaknesses

- CWE-79
- CWE-94

Credits

ngocnn97 Reporter