SiYuan Mermaid `javascript:` Link Injection Leads to Stored XSS and Electron RCE

Critical **88250** published **GHSA-x63q-3rcj-hhp5** last week

Package

siyuan (Go)

Affected versions

<=3.6.3

Patched versions

v3.6.4

Description

SiYuan renders Mermaid diagrams with `securityLevel: "loose"` and then injects the returned SVG into the DOM using `innerHTML`.

Because Mermaid in this configuration preserves attacker-controlled `href="javascript:..."` links inside the generated SVG, a malicious Mermaid code block can execute JavaScript in the renderer when the victim clicks the diagram node/link.

On desktop builds, this becomes code execution because Electron windows are created with:

- `nodeIntegration: true`
- `contextIsolation: false`
- `webSecurity: false`

As a result, attacker-controlled JavaScript from a Mermaid block can directly call Node APIs such as `require("child_process")`.

This is a real injection issue, not an intended formatting feature. Supporting Mermaid hyperlinks is one thing; allowing `javascript:` URLs to survive into an Electron renderer with Node integration is a broken security boundary.

Impact

Desktop / Electron

Stored XSS becomes user-assisted RCE.

A malicious note, imported document, synced workspace item, shared workspace artifact, or plugin-generated note containing a crafted Mermaid block can execute arbitrary OS commands once the victim opens the note and clicks the malicious Mermaid node.

Practical impact includes:

- Arbitrary command execution as the current desktop user
- Theft of local notes, credentials, tokens, SSH keys, browser data, or workspace secrets
- Persistence by dropping files into the user profile or workspace

Browser / non-Electron frontend

The same bug is still a stored XSS issue in web contexts because attacker-controlled JavaScript URLs are injected into DOM-rendered SVG.

Source to Sink

Source

User-controlled Mermaid code block content is stored in `data-content` and then passed to Mermaid:

- `app/src/protyle/toolbar/index.ts:1170`
- `app/src/protyle/render/mermaidRender.ts:100`

Relevant flow:

1. User edits or imports a Mermaid code block.
2. The code block payload is stored in the element `data-content`.
3. `mermaidRender()` reads `item.getAttribute("data-content")`.
4. The payload is rendered by Mermaid with unsafe settings.

Unsafe Mermaid configuration

`app/src/protyle/render/mermaidRender.ts:26-34`

```
const config: any = {
    securityLevel: "loose",
    ...
    flowchart: {
        htmlLabels: true,
```

```
        useMaxWidth: !0
    },
```

Sink

`app/src/protyle/render/mermaidRender.ts:100-101`

```
const mermaidData = await window.mermaid.render(id, Lute.UnEscapeHTMLStr(item.getA u
renderElement.lastElementChild.innerHTML = mermaidData.svg;
```

This is the critical sink. The Mermaid-generated SVG is trusted and inserted directly via `innerHTML`.

Electron privilege boundary collapse

Electron windows are created with Node exposed to renderer JavaScript:

- `app/electron/main.js:314`
- `app/electron/main.js:1157-1160`

Example:

```
webPreferences: {
    contextIsolation: false,
    nodeIntegration: true,
    webviewTag: true,
    webSecurity: false,
}
```

Once `javascript:` executes in the renderer, Node APIs are reachable.

Proof of Concept

Malicious Mermaid block

Trigger

1. Open a note containing the Mermaid block.
2. Let SiYuan render the Mermaid diagram.
3. Click node `A`.

Result

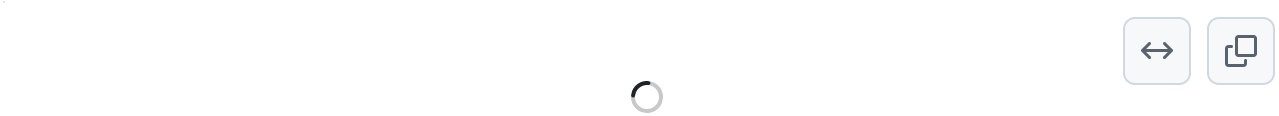
The JavaScript URI executes in the renderer context. On Electron, `require()` is available, so the payload can execute arbitrary OS commands.

Validation Performed

I validated the dangerous part directly against the bundled Mermaid version shipped in this repo:

- Bundle used: `app/stage/protyle/js/mermaid/mermaid.min.js`
- Version loaded by renderer: `11.13.0`

Using a local jsdom harness with the project's Mermaid bundle and the same frontend config (`securityLevel: "loose"`), rendering the following input:



produced SVG output containing:

```
href="javascript:alert(1)"
```

That proves attacker-controlled `javascript:` survives Mermaid rendering and reaches the DOM injection sink.

Suggested Fix

Any one fix alone is weaker than a defense-in-depth combination. The safe approach is to do all of the following:

1. Stop using Mermaid `securityLevel: "loose"` unless there is a narrowly justified need.
2. Reject or strip `javascript:`, `data:`, and other active URL schemes from Mermaid-generated links before DOM insertion.
3. Do not inject Mermaid SVG via raw `innerHTML` unless it is sanitized first.
4. Harden Electron windows:
   - disable `nodeIntegration`
   - enable `contextIsolation`
   - keep `webSecurity` enabled
5. Add regression tests that assert Mermaid output cannot contain active-script URLs.

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-2026-40322

Weaknesses

▶ CWE-79

Credits

ch1nhpd Reporter