From Markdown to a Shell: Two RCEs in Obsidian Desktop

CVE-2026-104077, CVE-2026-104078

CVSS 8.5 & 8.4



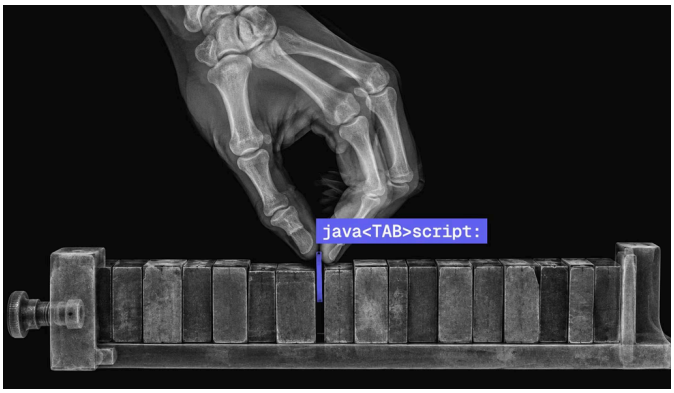
Hemanshu Baviskar

Oct 12, 2026

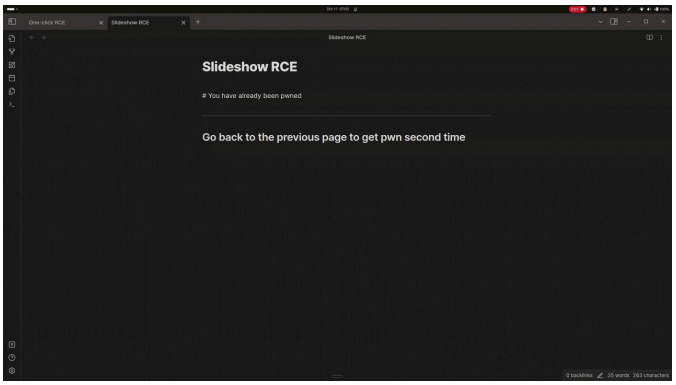
CONTENTS

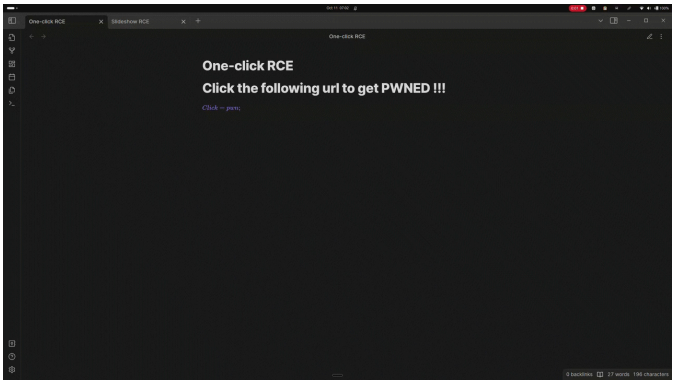
00 Introduction
01 At a glance
02 Electron puts a browser next to Node
03 Obsidian's note renderer had Node authority
04 RCE via Slides Plugin Markdown
05 RCE via MathJax Safe Filter Bypass
06 Two parser decisions, one privileged sink
07 Disclosure timeline
08 Primary references

0% READ



Two rendering paths turned attacker-controlled Markdown into JavaScript inside Obsidian's Node-enabled Electron renderer. One hid code in a Slides background iframe. The other hid a URL scheme behind a single TAB byte.





At a glance

What	Slides and MathJax each allowed a crafted Markdown value to reach a JavaScript URL sink.
Impact	The resulting JavaScript could call Node's `require()`, write files, and start a child process with the Obsidian user's permissions.
Trigger	The victim starts a presentation for Slides or clicks the rendered equation for MathJax.
Versions	Slides reproduced in 1.3.x and 1.13.x, then failed safely in 1.14.x. MathJax reproduced in 1.3.x, 1.13.x, and 1.14.x.

Status	Obsidian 1.14.x documents a Slides iframe-injection fix. Its bundled MathJax 4.1.3 still contains the TAB-sensitive parser.
CVSS	7.8 High under v3.1; 8.4 High under v4.0. Both vectors include local document processing and active victim interaction.

Electron puts a browser next to Node

Obsidian Desktop is an Electron application, so a note is not drawn by a plain web view, and that is where both bugs live.

Electron splits into two processes. The **main process** is Node.js, holding the windows and the filesystem; each window is painted by a **renderer**, a Chromium page, the same engine as a browser tab. In a browser that tab is sandboxed: script reaches the page and the network, never the machine. Electron keeps the renderer that weak by default, and a window's `webPreferences` is what can loosen it.

Two of those settings matter here. `nodeIntegration` is the wall between page script and the machine: off, the page cannot read a file or start a program; on, its JavaScript can call `require()` and pull in Node built-ins like `fs` and `child_process` directly. `contextIsolation` is the wall between the page and the trusted **preload** script that normally hands it a short, vetted set of privileged calls; off, the two share one world and the page can help itself to whatever the preload holds. Turn the first on and the second off together, and the renderer stops being a sandboxed tab, so page JavaScript now runs with the authority of a local application.

That one choice sets the price of a rendering bug. A script injected into a normal browser page is bounded by the browser and its origin rules. A script

injected into a Node-enabled renderer is bounded only by the user account it runs as. Electron's own security guidance says this plainly: disabling Node integration is what stops an XSS from becoming remote code execution, and its sandbox documentation records that enabling Node integration disables the renderer sandbox.

Obsidian's note renderer had Node authority

The Obsidian desktop build opened its vault window with exactly that configuration: Node integration on, context isolation off. This is the bridge that turns any JavaScript running inside a note into Node access, rather than browser-only script injection.

```
// beautified Obsidian desktop bundl
webPreferences: {
  contextIsolation: false,
  nodeIntegration: true,
  nodeIntegrationInWorker: true,
  spellcheck: true,
  webviewTag: true
}
```

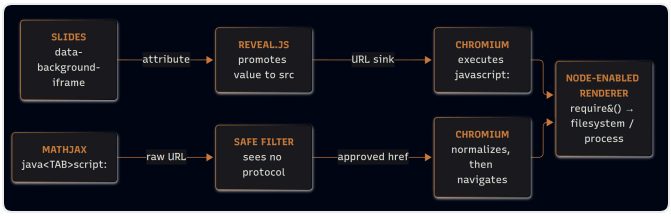
Configuration extracted from the tested desktop application, not an illustrative Electron sample.

So the question for both findings is narrow: can attacker-controlled Markdown get a line of javascript to run inside that window? Obsidian's rendering pipeline is built to stop exactly that. A note is parsed from Markdown to HTML, and that HTML is passed through a **sanitizer** (a filter that strips out anything dangerous, such as `<script>` tags, inline event handlers, and risky URL schemes) before the result reaches the page. If the pipeline ended there, a note would be inert.

It does not end there. After the sanitized HTML is in the DOM, Obsidian runs **post-processors** over it: later stages that read the already-approved HTML and build new elements from it. The Slides core plugin hands the deck to a bundled copy of Reveal.js; math is handed to a bundled copy of MathJax. These

components read the already-approved DOM and build new nodes from it. Both bugs live here, not in the sanitizer, but in a later stage that re-interprets a value the sanitizer already judged safe.

That gives both paths the same shape. The sanitizer approves a value because, at the moment it looks, the value is not a URL. A later stage then turns that same value into a URL and navigates to it, without re-checking. Slides promotes an inert data attribute into an iframe's `src`. MathJax approves a scheme that a single TAB byte has hidden from its filter, and Chromium strips the TAB back out when it follows the link. Two different mistakes, one result: a `javascript:` URL executing in a renderer that can reach Node.



RCE via Slides Plugin Markdown

The Slides path began with a raw HTML `<section>` in a note. Its `data-background-iframe` attribute survived sanitization as custom data. The sanitizer allowed it for a simple reason: an attribute named `data-*` is just storage for arbitrary values (it holds text, it doesn't *do* anything), so there was nothing dangerous to strip. That judgment held only while the attribute stayed inert data.

Reveal.js later read the attribute, created an iframe, placed the value in `data-src`, and copied it into `src`. The moment the value lands in `src`, it stops being storage and becomes a URL the browser will load. This is the sink. No URL-scheme check ran at that final transition, so a `javascript:` value executed inside the generated background iframe and reached the parent renderer through `parent.require()`.

```
// bundled Reveal.js 4.3.1 · format
const value = slide.getAttribute("da
const iframe = document.createEleme
iframe.setAttribute("data-src", valu

iframe.setAttribute("src", iframe.ge
```

The excerpt preserves the operations in the shipped minified bundle; variable names are expanded for readability.

```
<!-- benign proof shape (Markdo
<section data-background-iframe=
  "javascript:parent.require('f
    .writeFileSync('/tmp/obsidi

# Quarterly project review

</section>
```

The supplied proof writes one fixed marker. Starting the presentation is the required trigger.

RCE via MathJax Safe Filter Bypass

The second path used MathJax's `\href` macro, which lets a formula carry a clickable link. Obsidian configured MathJax to reject dangerous link schemes like `javascript:` and `data:`. The way that filter decided a scheme was dangerous is where it went wrong.

To find the scheme, the filter read the characters from the start of the URL up to the first colon, accepting only letters. The idea is that `javascript:` begins with a run of letters followed by `:`, so reading that run tells you the scheme. But the filter demanded the letters be *contiguous*, an unbroken run right up to the colon. Slip one TAB character between `java` and `script`, and the run

of letters is broken before the colon is reached. The filter finds no clean scheme at all, concludes the link has no protocol worth blocking, and lets it through.

```
// MathJax 3.2.2 · ts/ui/safe/SafeMe
const protocol =
  (url.match(/^\s*([a-z]+):/i) || [

return allow === 'all' ||
  (allow === 'safe' &&
    (safe.options.safeProtocols[pro
    ? url : null;
```

`[a-z]+` matches only an unbroken run of letters. A TAB breaks the run, the match fails, and the empty-protocol branch treats the link as safe.

MathJax 4.1.3 changed the expression to tolerate line-feed and carriage-return characters; it now matches them inside the scheme and strips them out before comparing. But TAB was still not on the list, so the same gap remained:

```
// Obsidian 1.14.x · lib/mathjax/obs
const protocol =
  (url.match(/^\s*([a-z\n\r]+):/i)
    .replace(/[\n\r]/g, "")
    .toLowerCase();

// TAB is neither matched nor remove
```

Formatted from the exact minified MathJax 4.1.3 bundle shipped in Obsidian 1.14.x.

Here is the mismatch that makes this exploitable. MathJax keeps the tabbed value as written and puts it into an anchor's `href`. When the user clicks, Chromium has to turn that `href` into a real URL to follow it, and Chromium's own URL parser quietly discards stray TAB bytes as part of normalizing the address. So the exact string MathJax waved through as “no scheme” becomes a clean `javascript:` URL the instant the browser acts on it. Two parsers looked at one value and disagreed about what it was;

the code ran only when the victim clicked the rendered equation.

```
canonicalization mismatch

MathJax receives     java<TAB>script
MathJax classifies   protocol = ""
Chromium exposes     javascript:…
```

The visible `<TAB>` label represents one literal byte, `0x09`.

> **Credit.** The MathJax filter defect was public upstream before this draft. MathJax maintainer Davide Cervone opened and merged PR #1557, which adds TAB to the filtered protocol characters, for the 4.2 milestone. The work described here establishes the Obsidian-specific path to Node, affected application versions, and the result on the current public 1.14.x build.

Two parser decisions, one privileged sink

Property	Slides	Ma
Attacker-controlled value	`data-background-iframe`	`ja`
Mismatch	Data attribute later becomes a URL	Filt car diff
Final sink	`iframe.src`	An
Victim action	Start presentation	Cli equ
Node primitive	`parent.require()`	`re`

> Neither path executed when the note merely existed in the vault. Slides required its core plugin to be enabled and the presentation to start. MathJax required the formula to render and the user to click its anchor. Those preconditions belong in any reproduction or risk discussion.

Disclosure timeline

The 1.14.x Slides run ended without an attacker-controlled iframe or marker. The 1.14.x MathJax run still turned `java<TAB>script:` into `javascript:`, accepted a trusted click, and created the child-process marker. Those are the two measured outcomes at the public release boundary.

2026-09-16	Both issues reported to Obsidian.
2026-10-06	Obsidian 1.14.x is now available as a public release.

Primary references

- Electron process model and security guidance.
- Electron renderer sandboxing documentation.
- Obsidian 1.14.x public changelog.
- MathJax PR #1557: Filter TAB from URL protocols.
- Obsidian desktop release metadata.

Frequently Asked Questions

Why are these two bugs in one post?

Does opening a note trigger either one?

I'm on 1.14.x — am I safe?

Do I need a community plugin to be exposed?

Is Obsidian Mobile affected?

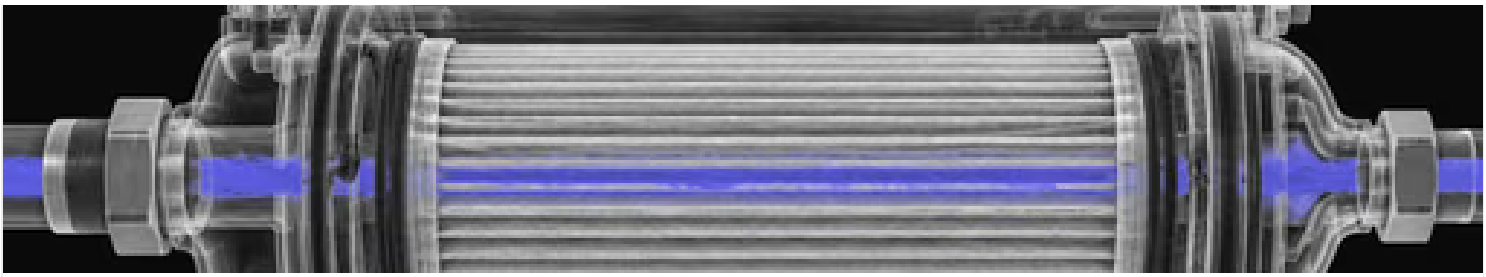
More from CodeAnt

View All



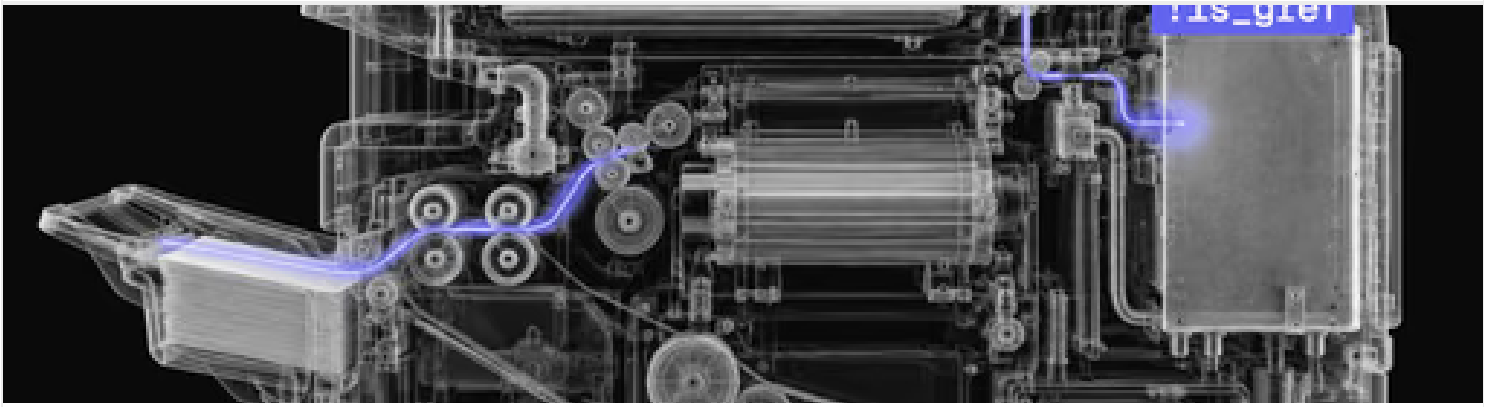
CVE-2026-29000: pac4j-jwt Auth Bypass PoC With a Public Key

CVE-2026-29000 CVSS 10



CVE-2026-28292: simple-git RCE via Case-Sensitivity Bypass

CVE-2026-28292 CVSS 9.8

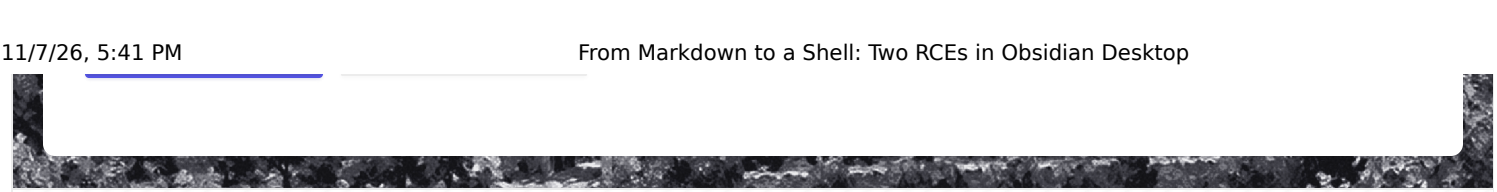


Xenbleed: The Eleven-Year-Old Shortcut

CVE-2026-102703 CVSS 9.4

Find out what's already
exploitable in your codebase.

Book a demo Start free trial



CodeAnt AI

355 Bryant St, San Francisco, CA 94107

All systems normal

Product
- AI Pentesting
- AI Code Review
- Code Security
- Code Quality
- Developer 360

Company
- About
- Pricing
- Customer Story
- Blogs
- Glossary
- Enterprise
- Contact Sales

Developers
- Documentation
- Integrations
- Security Research
- Vulnerability Data
- System Status

© 2026 CodeAnt AI. All rights reserved.

DPA Terms and Conditions Privacy Policy

Trust Center EU AI Act Policy Report a Vulnerability