MINISTERSTVO
INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA
A INFORMATIZÁCIE
SLOVENSKEJ REPUBLIKY

CSIRT.SK

When Rendering Hurts: Turning SVG into Browser DoS in OTRS

CVE-2026-48208

Created by: CSIRT.SK
Ministerstvo investícií, regionálneho rozvoja a informatizácie SR
Pribinova 25
811 09 Bratislava

Date of creation: March 2026

TLP: Clear

Introduction

Modern web applications often assume that disabling JavaScript is sufficient to safely render untrusted content.

But what happens when the browser itself becomes the execution engine?

In this article, we explore a denial-of-service vulnerability in OTRS arising from how ticket content is rendered. By embedding specially crafted SVG content inside a ticket, an attacker can cause the agent's browser tab to crash upon viewing.

This attack requires no JavaScript, bypasses Content Security Policy (CSP), and relies entirely on the browser's rendering behavior.

The Core Issue

At the heart of this vulnerability is a mismatch between what the application considers "safe content" and what the browser is capable of processing.

OTRS renders HTML content directly within the ticket interface. This includes inline SVG elements.

While CSP prevents script execution, it does not prevent the browser from parsing and rendering complex SVG structures.

Certain SVG constructs can trigger excessive resource consumption or instability during rendering.

Key Insight

- The application blocks code execution. The browser still executes rendering logic.

Root Cause

The root cause is not the presence of active scripting, but the lack of restrictions on **renderable content complexity**.

The application assumes that:

- Blocking JavaScript is sufficient
- Rendering content is inherently safe

Both assumptions break when faced with complex SVG inputs.

Why This Is Subtle

This is not a typical cross-site scripting issue.

Instead, it is:

- A **rendering-based denial of service**
- Triggered by **valid, standards-compliant content**
- Caused by **abuse of browser internals rather than application logic**

This makes it particularly dangerous because:

- No obvious "malicious" payload is required
- CSP provides a false sense of security
- The issue manifests only when content is rendered

Attack Scenario

The attack leverages normal OTRS functionality.

1. An attacker sends an email containing crafted SVG content
2. OTRS automatically creates a ticket from the email
3. An agent opens the ticket in the web interface
4. The browser attempts to render the SVG
5. The tab crashes or becomes unresponsive

Key Observation

- No authentication required
- No user interaction beyond opening the ticket
- No scripting involved

Impact

In affected environments, an attacker can:

- Cause browser tab crashes for agents
- Disrupt ticket handling workflows
- Repeatedly trigger denial of service via automated ticket creation

In environments with automated email ingestion, this can be amplified:

- Multiple malicious tickets can be generated
- Multiple agents can be affected simultaneously

In short:

➢ A remote attacker can degrade or disrupt support operations without executing a single line of JavaScript.

Why CSP Does Not Help

Content Security Policy is designed to control **resource loading and script execution**.

However:

- SVG rendering is allowed as part of standard HTML processing
- No external resources are required for the attack
- No scripts are executed

As a result:

➢ CSP successfully blocks the wrong threat model.

Proof of Concept (High-Level)

Goal

Demonstrate that rendering SVG content alone can lead to browser instability.

Core Idea

A crafted SVG payload is embedded into a ticket via email.

When the ticket is opened:

- The browser parses the SVG
- Rendering triggers excessive processing
- The tab crashes or becomes unresponsive

Delivery

The payload can be delivered through standard email-to-ticket pipelines.

Note

The vulnerability does not depend on:

- JavaScript execution
- External resources
- User interaction beyond viewing the ticket

Full PoC

A full proof-of-concept script is available here:

https://github.com/Habuon/CVE-2026-48208

Detection Considerations

Detecting this issue can be challenging:

- No logs indicate exploitation
- No script execution occurs
- Failures appear as client-side crashes

Indicators may include:

- Agents reporting browser instability when opening tickets
- Reproducible crashes tied to specific tickets

Defensive Lessons

This vulnerability highlights several important lessons:

1. Rendering Is an Attack Surface

Displaying untrusted content is not passive. It invokes complex browser logic.

2. CSP Is Not a Complete Defense

Blocking JavaScript does not eliminate all forms of client-side abuse.

3. Treat SVG as Active Content

SVG is not just an image format — it is a programmable rendering system.

4. Sanitize Beyond Scripts

Security controls must account for structural and rendering complexity, not just executable code.

Mitigation Strategies

Recommended mitigations include:

- Strip or disable inline SVG content in ticket bodies
- Sanitize SVG elements and attributes aggressively
- Render untrusted content inside sandboxed iframes
- Convert SVG to safe raster formats before display

Disclosure

This issue was responsibly disclosed to the OTRS Product Security Team through a coordinated disclosure process.

Timeline

- **2026-03-29 —** Vulnerability discovered during research into OTRS ticket rendering and browser-side attack surfaces.
- **2026-03-30 —** The issue was privately reported to the OTRS Product Security Team together with technical details and proof-of-concept material.
- **2026-04-09 —** The vendor confirmed the vulnerability and verified the denial-of-service impact caused by crafted SVG content.
- **2026-05-21 —** The issue was assigned **CVE-2026-48208**.
- **2026-06-01 —** Coordinated public disclosure and advisory release.

Vendor Assessment

CVSS v4.0

High — **7.1**

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:N/VI:N/VA:H/SC:N/SI:N/SA:N/AU:Y/R:U/RE:L/U:Amber

CVSS v3.1

Medium — **6.5**

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

Conclusion

This vulnerability is a reminder that modern web security extends beyond script execution.

Applications often focus on preventing code injection, while overlooking the risks of rendering complex, untrusted content.

This wasn't a failure of filtering.

It was a failure to recognize that rendering itself can be exploited.