Security Report — Elearning Platform (University of Bouira)

Reporter: Zerf-003 (bug bounty / security researcher)

Scope / Target: `https://elearning.univ-bouira.dz`

Disclosure policy: Responsible disclosure — I request coordinated remediation before any public disclosure.

1. Introduction

As a bug bounty hunter and security researcher (white/grey hat), I performed security testing on the Elearning platform of Bouira University. My actions were strictly professional and limited to non-destructive tests. The purpose of this report is to describe findings, provide reproducible proof of concept (PoC), and give remediation advice to reduce risk.

2. Executive summary

- **Vulnerability type (primary):** HTML Injection (reflected / stored)
- **Affected component:** Comments Sections and Chats between users
- **Impact:** UI manipulation, potential stored XSS when payloads are rendered/executed in other users' browsers, potential CSRF / internal request triggers via injected markup.
- **Severity (preliminary):** Medium → High (depends on whether injected HTML is executed in victim browsers and on user roles affected).
- **Status:** Proof of concept produced. Recommendation: fix input sanitization and output encoding; follow remediation steps below.

3. Finding

#1 - HTML Injection

3.1 Description

The application accepts user input that is later rendered into HTML without proper context-aware escaping. This allows an attacker to inject HTML fragments into the page. Depending on where the injected HTML is rendered and how the application inserts it into the DOM, this may lead to reflected or stored XSS (execution of attacker JavaScript in other users' browsers), UI redressing, or forced requests (images/forms) that trigger actions on internal services. All website inputs must be checked for escaping user inputs.

3.2 Affected endpoints / locations

- `POST /lib/ajax/service.php?sesskey=******&info=core_message_send_messages_to_conversation` — the text, or other form parameters supplied during sending a message t are reflected in the response and are included in rendered pages.

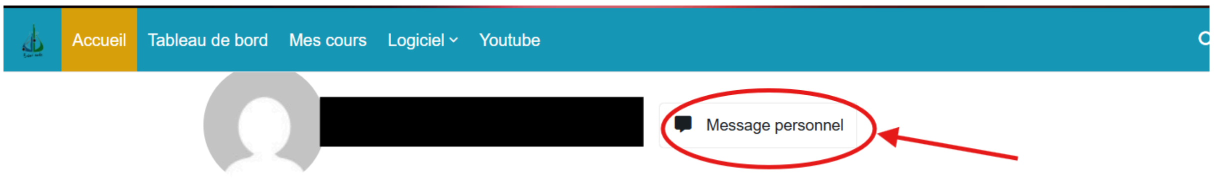
3.3 Reproduction (PoC) — minimal, non-destructive

> Do not run these on third-party systems. These steps are for your controlled testing environment or for the vendor to reproduce.

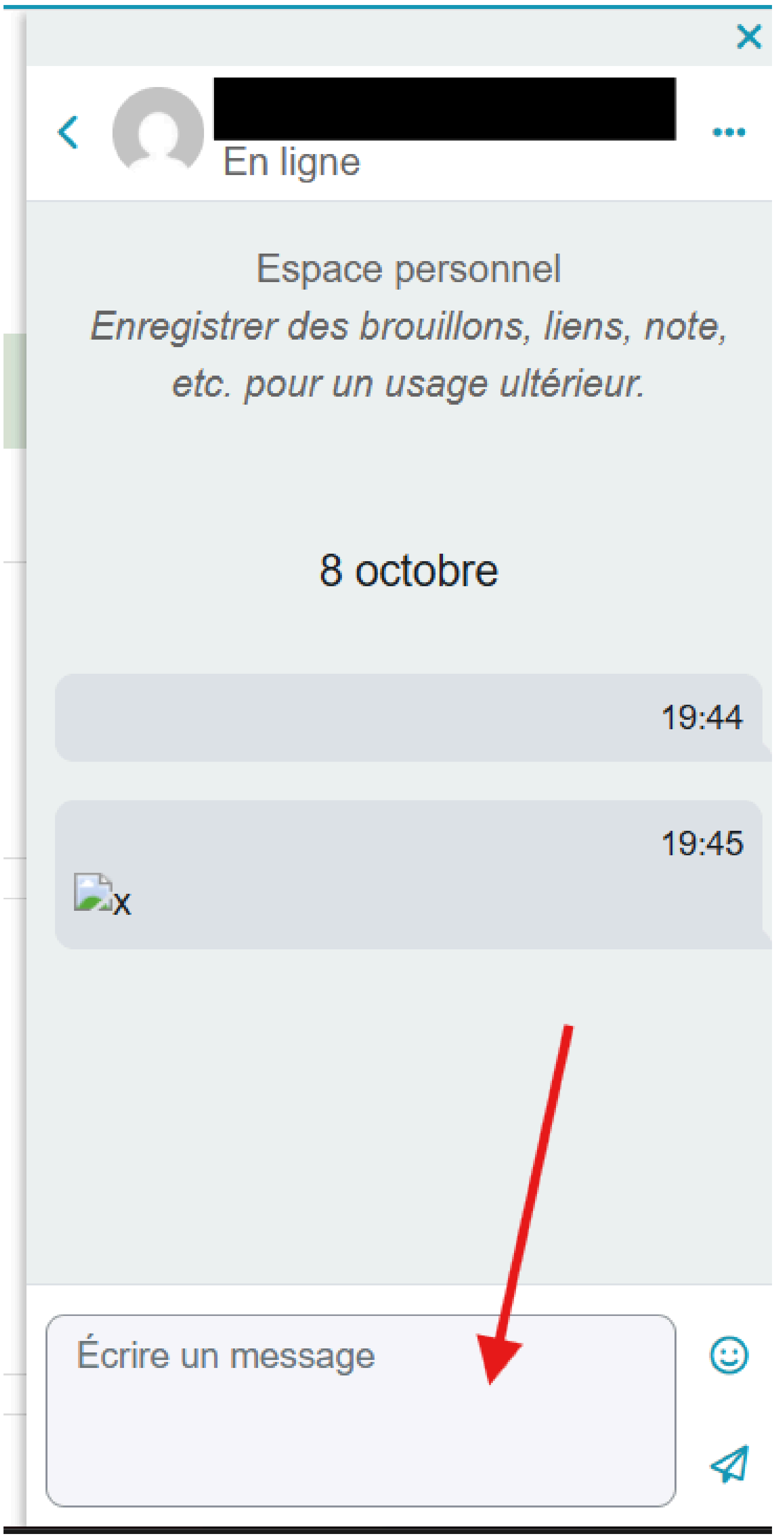
1. Prepare a test event with a malicious `text` (or other field) containing a benign payload:

```
<h1>HTML injection test</h1>
```

As follow in the images:



[redacted]


[redacted]
En ligne

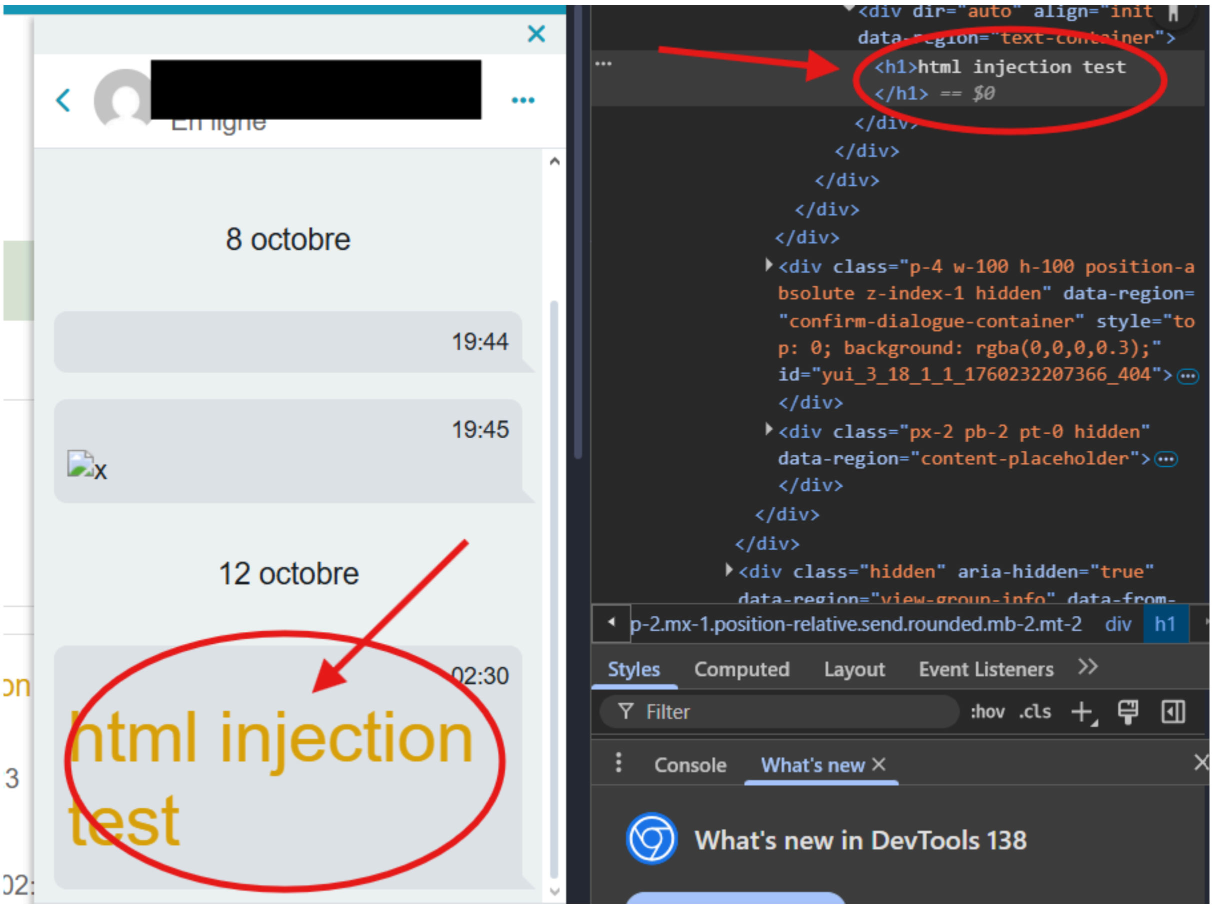
Espace personnel
Enregistrer des brouillons, liens, note, etc. pour un usage ultérieur.

8 octobre

19:44

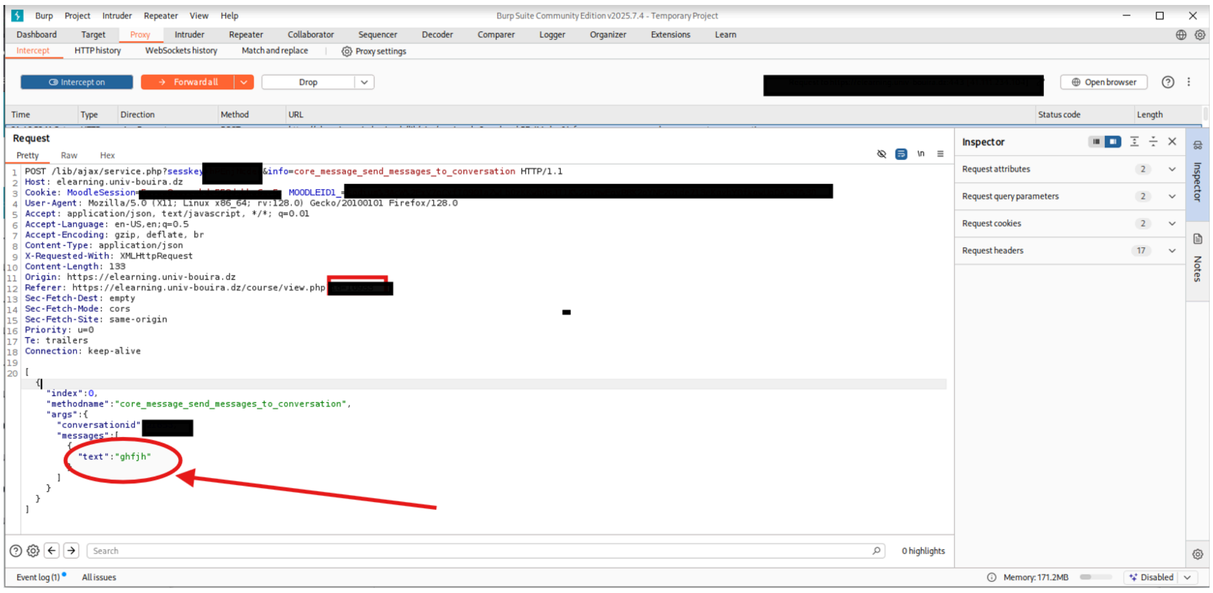
19:45
x

Écrire un message



```
<div dir="auto" align="init
data-region="text-container">
  <h1>html injection test
  </h1> == $0
  </div>
 </div>
 </div>
 </div>
</div>
<div class="p-4 w-100 h-100 position-absolute z-index-1 hidden" data-region="confirm-dialogue-container" style="top: 0; background: rgba(0,0,0,0.3);" id="yui_3_18_1_1_1760232207366_404"> ⋯
</div>
<div class="px-2 pb-2 pt-0 hidden" data-region="content-placeholder"> ⋯
</div>
</div>
</div>
<div class="hidden" aria-hidden="true" data-region="view-group-info" data-from-
```

p-2.mx-1.position-relative.send.rounded.mb-2.mt-2 div h1
Styles
Computed
Layout
Event Listeners
Filter
:hov .cls
Console
What's new
What's new in DevTools 138


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.7.4 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward all
Drop
Open browser
Time Type Direction Method URL Status code Length
Request
Pretty Raw Hex
Inspector
Request attributes 2
Request query parameters 2
Request cookies 2
Request headers 17
Search
0 highlights
Event log (1)
All issues
Memory: 171.2MB
Disabled

```
POST /lib/ajax/service.php?sesskey[redacted]&info=core_message_send_messages_to_conversation HTTP/1.1
Host: elearning.univ-bouira.dz
Cookie: MoodleSession=[redacted] MOODLEID1_=[redacted]
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 133
Origin: https://elearning.univ-bouira.dz
Referer: https://elearning.univ-bouira.dz/course/view.php[redacted]
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers
Connection: keep-alive

[
  {
    "index":0,
    "methodname":"core_message_send_messages_to_conversation",
    "args":{
      "conversationid"[redacted]
      "messages":[
        {
          "text":"ghfjh"
        }
      ]
    }
  }
]
```

Note: POSSIBLE SSRF.

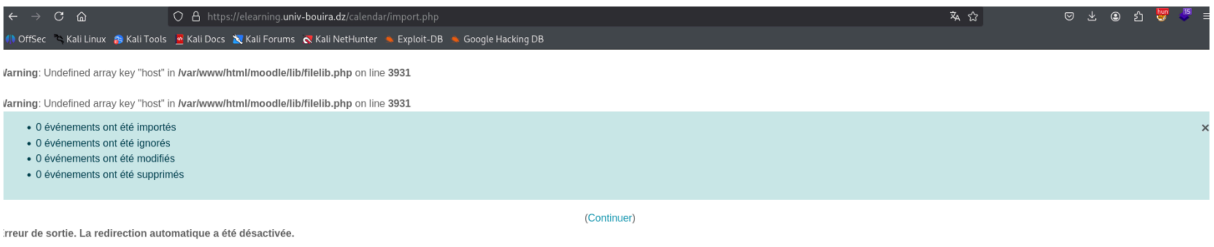
During testing I observed a PHP warning (Undefined array key "host" in /var/www/html/moodle/lib/filelib.php on line 3931) when passing malformed input; the application rejected certain malformed URLs where i tried a Server `Side Request Forgery.` However, properly formed payloads (HTML snippets encoded or in allowed URL fields) can be reflected in the UI.you can try it on the end point: `/calendar/import.php` , by passing the `name` and the `url` values payloads as:

`http://ATTACKER_HOST/test.ics`

`http://ATTACKER_HOST:8080/test.ics`

`http://localhost/test.ics`

Evidence:



https://elearning.univ-bouira.dz/calendar/import.php

```
Varning: Undefined array key "host" in /var/www/html/moodle/lib/filelib.php on line 3931

Varning: Undefined array key "host" in /var/www/html/moodle/lib/filelib.php on line 3931
```

- 0 événements ont été importés
- 0 événements ont été ignorés
- 0 événements ont été modifiés
- 0 événements ont été supprimés

(Continuer)

rreur de sortie. La redirection automatique a été désactivée.

Mitigation / Recommended fixes

Apply **defense in depth**:

1. **Output encoding** — escape user data based on the insertion context:
   - HTML body context: HTML-escape (`&lt;`, `&gt;`, `&amp;`, `&quot;`).
   - Attribute context: attribute-encode values or use DOM APIs (`element.setAttribute`) not `innerHTML`.
   - JavaScript context: JS string escape.
2. **Avoid `innerHTML` for user content**. Use `textContent` or safe templating engines that do auto-escaping.
3. **Sanitize allowed HTML** only with a strict whitelist (if you must allow some HTML). Use vetted libraries (e.g. DOMPurify on the client + server-side

sanitizer for any stored HTML).

4. **Validate inputs server-side**: strip or reject embedded scripts or suspicious characters in fields that are expected to be plain text (like event names).
5. **Set secure cookie flags**: `HttpOnly` , `Secure` , `SameSite=strict` to limit impact of client-side script stealing cookies.
6.
    - I attempted to use the calendar import `url` field to force the server to fetch an attacker-hosted `test.ics` . The application returned a PHP warning (`Undefined array key "host"`) and the import failed — indicating the server validated or rejected the malformed input. No inbound request was logged on my test server for the malformed attempts.
    - Next steps if you want to confirm SSRF capability: expose a valid test URL accessible to the host (ngrok or a public server), and submit a valid `http://your-host/test.ics` via the import form. If the server attempts to fetch it, you should see a GET in your test server logs.

NOTES:

HTTP Security

Content Security Policy❌ No

Strict Transport Policy❌ No

X-Content-Type-Options❌ No

X-Frame-Options✅ Yes

X-XSS-Protection

NO DNS SECURITY:

Without DNSSEC, it's possible for MITM attackers to spoof records and lead users to phishing sites. This is because the DNS system includes no built-in methods to verify that the response to the request was not forged, or that any other part of the process wasn’t interrupted by an attacker. The DNS Security Extensions (DNSSEC) secures DNS lookups by signing your DNS records using public keys, so browsers can detect if the response has been tampered with. Another solution to this issue is DoH (DNS over HTTPS) and DoT (DNS over TLD).

MORE!:

The checks I have performed were proof-of-concept, non-destructive validations to confirm observed behavior. If the university requests a full security assessment, I ask for a formal authorization detailing scope, test accounts (if any), acceptable times, and rules of engagement. Under those terms I will perform an in-depth assessment (automated scanning, manual verification, targeted exploitation where allowed, SSRF/XSS/CSRF logic checks), using industry-grade tools, and produce a technical report with evidence, impact analysis, and a prioritized remediation plan. I also request a confidentiality window (e.g., X days) to allow coordinated remediation before any public disclosure.