Audit: [redacted]

Bauer Sandro

2023-02-24

sandr0.xyz

Prepared for ■■■■■■■■■■.

Given the time-boxed scope of this assessment and its reliance on client-provided information, the findings in this report should not be taken as a comprehensive listing of all security issues.

Therefore the authors assume no responsibility for errors, omissions, or for damages resulting from the use of the information contained herein. The authors do not guarantee the security of a system.

Contents

Revision	Date	Change	Editor
1	2023-02-17	Initial Report	Bauer Sandro
2	2023-02-24	Audit of Updates	Bauer Sandro

1. Executive Summary

Application Summary	
Application Name	■■■■■■■■■■
Application URL	■■■■■■■■■■
Application Date	January 21st
Plattform	Linux
Engagement Summary	
Engagement Type	Application Security Audit / Penetration Test
Methodology	White Box
Scope	
Commit ID	■■■■■■■■■■
Information	Some parts of the source code are not part of release builds and are explicitly out of scope.
Findings Summary	
Critical	-
High	TD-SRE: SSRF and Resource Exhaustion
Medium	TD-XSS: Cross-Site Scripting (XSS)
Low	TD-IES: Insufficient Escapes
Informational	TD-MIS: Miscellaneous, TD-OUT: Vulnerable and Outdated Components

1.1. Re-evaluation after updates

Application Summary	
Application Name	■■■■■■■■■■
Application URL	■■■■■■■■■■
Application Date	February 24th
Plattform	Linux
Scope	
Commit ID	■■■■■■■■■■
Information	Critical vulnerabilities have been fixed and the relevant parts of the application have been re-evaluated.
Findings Updates	
Critical	-
High	all addressed
Medium	all addressed
Low	TD-IES: Insufficient Escapes - ongoing work
Informational	updated where appropriate

2. Findings

2.1. SSRF and Resource Exhaustion

ID:	TD-SRE
Severity:	**High**
CWE:	CWE-918 CWE-400 CWE-20
Required Permissions:	Unauthenticated

> UPDATE: This vulnerability has been addressed and fixed in commit

> ■■■■■■■■■■

There is a severe flaw in the ■■■■■■■■■■, allowing an unauthenticated user to perform SSRF. The server will request data from other websites.

When uploading a signature, the application creates a HTML and a PDF file containing the data. Dompdf [1] is used for creating the contract PDF while the Blade templating engine [2] is used for creating the HTML.

The user supplied signature is passed directly to the templating engine without further sanitization:

```php
// **********
$signature = $request->get('signature');
// ...
$service = new PDFGenerator($registrations, $registration->**********,
    $registration->**********, $**********);
$pdf = $service->generate();
```

```php
// **********
$pdfname = $storagePath . $id . '.pdf';
$htmlname = $storagePath . $id . '.html';

$data = [
    // ...
    '**********'    => $this->**********
];

$view = \View::make('**********', $data)->render();
\File::put($htmlname, $view);

$generated = $this->generateDOMPDF($htmlname, $pdfname);
```

[1] https://dompdf.github.io/
[2] https://laravel.com/docs/9.x/blade

```
// **********
<div class="col-md-12␣$**********">
    <img src="{{␣$**********␣}}" alt="**********">
</div>
```

Even though the variable ■■■■■■■■■■ is properly escaped in the template, the attacker can supply arbitrary URLs which will be accessed by Dompdf at PDF creation.

Example Payload:

```
http://evil.com/#,
```

The server will access `http://evil.com`. This can be used in an DDoS attack against other arbitrary servers, or in accessing internal network endpoints.

2.1.1. SVG

At PDF creation, Dompdf parses all `href`- and `xlink:href`-attributes in `<image>` tags of SVG files and recursively loads them. [3] An attacker can therefore trigger Dompdf to load a malicious SVG cyclicly depending on another SVG file, resulting in a `502 - Bad Gateway` indicating that the process responsible for handling the request crashed:

Thanks to PHP-FPM, the server will continue to work normally to other users.

Proof of concept:

```
<!-- 1.svg -->
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org
    /1999/xlink" viewBox="0␣0␣800␣800">
<image width="800" height="800" href="2.svg" />
</svg>
```

```
<!-- 2.svg -->
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org
    /1999/xlink" viewBox="0␣0␣800␣800">
<image width="800" height="800" href="1.svg" />
</svg>
```

The attacker has to host both SVGs on her own webserver and trick the ■■■■■■■■■■-server to load it using the technique described in TD-SRE: SSRF and Resource Exhaustion

Using SVGs, an attacker can also cause the server to DoS another server, or exhaust its own resources.

To achieve that, an SVG with arbitrary many other `href`s can be supplied causing the server to ask all of them recursively.

[3] https://github.com/dompdf/dompdf/blob/master/src/Image/Cache.php#L55

In another possible exploitation for this vulnerability, an attacker can create an SVG loading more SVGs which can scale to use arbitrarily many resources following the principle used by the billion laughs attack. [4]

Recommendation

- Load the resource from the file system like is done in ■■■■■■■■■■ also in ■■■■■■■■■■
- Sanity check the image to really be a PNG file

2.2. Cross-Site Scripting (XSS)

ID:	TD-XSS
Severity:	**Medium**
CWE:	CWE-79
Required Permissions:	■■■■■■■■■■

> UPDATE: This vulnerability has been addressed and fixed in commit
>
> ■■■■■■■■■■

A ■■■■■■■■■■ has the option to put arbitrary HTML into "■■■■■■■■■■" in his settings. It is therefore possible to enter `<script>` tags which allows to execute arbitrary Javascript when users get shown the contract (■■■■■■■■■■). The XSS triggers twice after saving / opening common settings.

Cross site scripting can be used to extract cookies from victims visiting the web page. It is possible for an attacker with ■■■■■■■■■■-permissions to gain admin access tricking an admin to visit the webpage and stealing their cookie. [5]

Recommendation

- Restrict tags which can be used in all ■■■■■■■■■■ forms to sane tags (for example `<i>`, `<b>`, `<strong>`) and escape/strip other tags by default.

2.3. Insufficient Escapes

ID:	TD-IES
Severity:	**Low**
CWE:	CWE-116
Required Permissions:	■■■■■■■■■■

[4] https://en.wikipedia.org/wiki/Billion_laughs_attack
[5] https://owasp.org/www-community/attacks/xss/

> UPDATE: Ongoing work, individual update for each ■■■■■■■■■■ necessary

The Blade templating engine does not escape single backslashes.

When creating a new Event, the ■■■■■■■■■■ can set the ■■■■■■■■■■ to `**********\` which leads to broken JavaScript code on the event page.

The template

```
// File **********
$('.**********').html('<h4>{{␣$**********}}</h4>{{␣$**********␣}}<br>{{
    ␣$**********␣}}␣{{␣$**********␣}}');
```

gets evaluated as

```
$('.**********').html('<h4>**********</h4>**********<br>**********\');
```

which does not end the string.

In theory, if the attacker is able to control another input after the insufficient escape, this could lead to XSS. Since only a ■■■■■■■■■■ (or admin) can edit this and further we did not find any exploitable case, we estimate the severity at **Low**.

Recommendation

- Strip, escape or blacklist single backslashes

2.4. Vulnerable and Outdated Components

ID:	TD-OUT
Severity:	**Informational**
CWE:	CWE-1352

> UPDATE: Components are updated regularly and this has been addressed

The software is hosted in a docker with image `nginx:1.21`. The docker contains 10 critical, 18 high, 14 medium and 95 low vulnerabilities and should be updated. [6]

PHP 8.0 is no longer actively supported since 2022-11-26 and only gets security fixes until 2023-11-26. [7] The GitLab CI uses PHP 8.0, while the it gets deployed with 8.2. From 8.0 to 8.2, there exist a few incompatibilities which can work in testing but may fail in release. [8,9]

[6] https://snyk.io/test/docker/nginx%3A1.21
[7] https://www.php.net/supported-versions.php
[8] https://www.php.net/manual/en/migration81.incompatible.php
[9] https://www.php.net/manual/en/migration82.incompatible.php

Positive: There are no known vulnerabilities for the used PHP components, checked with `php vendor/bin/security-checker security:check`.

Both, backend and frontend both have outdated packages which should be updated.

Recommendation

- Update the components to the latest version

2.5. Miscellaneous

ID:	TD-MIS
Severity:	**Informational**

2.5.1. Missing validation

> UPDATE: this has been addressed and fixed

It is possible to query ■■■■■■■■■■ with POST data ■■■■■■■■■■. Then the webpage throws an exception since `data` is set to `null` but ■■■■■■■■■■ tries to access `$data['**********']` for logging. `"**********"` validation is missing in ■■■■■■■■■■.

Recommendation

- Add missing validation to ■■■■■■■■■■

2.5.2. Missing default folder

> UPDATE: this has been addressed and fixed

Folder ■■■■■■■■■■ does not exist by default, while other folders in ■■■■■■■■■■ have either `.gitignore` or `.gitkeep`.

When deploying, the missing folder will cause the application to fail uploading a driver's license. The missing folder may not be found until a customer reports a problem.

Recommendation

- Add missing folder to git using `.gitkeep` or `gitignore`

2.5.3. Buggy template

> UPDATE: this has been addressed and fixed

In multiple templates, `readfile` will return the read bytes which get printed by the templating engine.

```
// ********** (among others)
<style>
    {!! readfile(public_path('css/print.css')) !!}
</style>
```

will be evaluated to

```
<style>
...
.********** {
  **********: **********;
  **********: **********%;
}
1763
</style>
```

Since this does not lead to errors in the display of the application this finding is purely informational.

Recommendation

- Do not print the result, but use `@php` to evaluate the `readfile`

2.5.3.1. Dead Code

> UPDATE: this has been addressed and fixed

The following function is never used, but is still present in the code. Further (as the comment) states, it is potentially dangerous and should therefore be removed.

```
// **********
protected function **********($htmlname, $pdfname)
{
    **********
}
```

Recommendation

- Remove unused function. If later needed, it can be recovered via git

A. Severity

Critical	Vulnerability is a high severity issue, but needs immediate action to mitigate it.
High	Vulnerability introduces a significant risk and does not need other vulnerabilities to be exploitable.
Medium	Vulnerability does not lead directly to the compromise of the system but already exposes a risk.
Low	Vulnerability indicates a limited risk. May require the presence of additional vulnerabilities.
Informational	No direct security impact, but should be fixed when following best practices.