Written By :

- Brian Kheng 13521049
- Fawwaz Abrial Saffa 18221069
- Jonathan Arthurito Aldi Sinaga 18221079

1. Possible Non-existing CSRF Token

First, we could search all types of <form> under /src directory. There are several possible vulnerable forms. I assume that all GET forms are not vulnerable since there will be no operation invoked on the server-side. Here are the list of files containing POST <form>

1. /src/views/pages/company/profile/index.php
2. /src/views/pages/company/jobs/create/index.php
3. /src/views/pages/company/jobs/[jobId]/edit/index.php
4. /src/views/pages/company/jobs/[jobId]/applications/[applicationId]/index.php
5. /src/views/pages/jobs/[jobId]/apply/index.php
6. /src/views/pages/auth/sign-in/index.php
7. /src/views/pages/auth/sign-up/job-seeker/index.php
8. /src/views/pages/auth/sign-up/company/index.php

In each that <form> tag, I saw that there's no csrf_token embedded in it, which means all of those eight POST <form> could be a vector for CSRF attack. For the sake of simplicity, let's just login as a job-seeker and try to do the CSRF attack. The attack will be done on 3 endpoints. It could be simulated using just CURL or using HTML which will run the script for sending the POST request.

The attack will be done by first cloning the main branch of the repo, and then copying the all files under the /csrf directory in fix/csrf-token branch.

1. /src/views/pages/company/profile/index.php
   The endpoint of this is /company/profile and it's possible to do the attack. Try to open csrf/1.html from the source code.

```
curl -X POST \
  -b "PHPSESSID=<session>" \
  -d "name=gooner&location=gooner&about=23423423gooning234" \
  http://172.19.0.3/company/profile
```

2. /src/views/pages/company/jobs/create/index.php
   The endpoint of this is /company/jobs/create and it’s possible to do the attack. Try to open csrf/2.html from the source code.

```
curl -X POST \
  -b "PHPSESSID=<session>" \
                -H       "Content-Type:       multipart/form-data;
boundary=----geckoformboundary3e243ccce1427a7f8d8d41eb1ef5df1f" \
                                                        --data-binary
$'------geckoformboundary3e243ccce1427a7f8d8d41eb1ef5df1f\r\nCont
ent-Disposition:                                        form-data;
name="position"\r\n\r\ntodottt\r\n------geckoformboundary3e243ccc
e1427a7f8d8d41eb1ef5df1f\r\nContent-Disposition:       form-data;
name="description"\r\n\r\n<p>testing
goon1</p>\r\n------geckoformboundary3e243ccce1427a7f8d8d41eb1ef5d
f1f\r\nContent-Disposition:                            form-data;
name="job-type"\r\n\r\nfull-time\r\n------geckoformboundary3e243c
cce1427a7f8d8d41eb1ef5df1f\r\nContent-Disposition:     form-data;
name="location-type"\r\n\r\nhybrid\r\n------geckoformboundary3e24
3ccce1427a7f8d8d41eb1ef5df1f\r\nContent-Disposition:   form-data;
name="attachments[]";                 filename=""\r\nContent-Type:
application/octet-stream\r\n\r\n\r\n------geckoformboundary3e243c
cce1427a7f8d8d41eb1ef5df1f--' \
  http://172.19.0.3/company/jobs/create
```

3. /src/views/pages/company/jobs/[jobId]/edit/index.php
   The endpoint of this is /company/jobs/create and it’s possible to do the attack. Try to open csrf/3.html from the source code.

```
curl -X POST \
  -b "PHPSESSID=<session>" \
```

```
                -H       "Content-Type:       multipart/form-data;
boundary=----geckoformboundaryf4212a470717711aab1f9ffc40fa1416" \
                                                        --data-binary
$'------geckoformboundaryf4212a470717711aab1f9ffc40fa1416\r\nCont
ent-Disposition:     form-data;     name="position"\r\n\r\ngooner
123\r\n------geckoformboundaryf4212a470717711aab1f9ffc40fa1416\r\
nContent-Disposition:                                   form-data;
name="description"\r\n\r\n<p>edited          for          testing
</p>\r\n------geckoformboundaryf4212a470717711aab1f9ffc40fa1416\r
\nContent-Disposition:                                   form-data;
name="is-open"\r\n\r\n0\r\n------geckoformboundaryf4212a470717711
aab1f9ffc40fa1416\r\nContent-Disposition:                form-data;
name="is-open"\r\n\r\n1\r\n------geckoformboundaryf4212a470717711
aab1f9ffc40fa1416\r\nContent-Disposition:                form-data;
name="job-type"\r\n\r\nfull-time\r\n------geckoformboundaryf4212a
470717711aab1f9ffc40fa1416\r\nContent-Disposition:     form-data;
name="location-type"\r\n\r\nhybrid\r\n------geckoformboundaryf421
2a470717711aab1f9ffc40fa1416\r\nContent-Disposition:   form-data;
name="attachments[]";                 filename=""\r\nContent-Type:
application/octet-stream\r\n\r\n\r\n------geckoformboundaryf4212a
470717711aab1f9ffc40fa1416--' \
  http://172.19.0.3/company/jobs/<jobnumber>/edit
```

We could solve it by adding the csrf-token in it. There needs to be rework on the authentication logic of the application. Here are the steps for the CSRF resolution.

1. Create utils to give the logic of handling all csrf-related functions.
2. For incoming requests, if the content type includes 'application/json' and the HTTP method is not GET, the CSRF token must be validated.
3. Embed the <meta csrf-token> in the root-layout so the csrf-token could be accessed from anywhere.
4. Embed the csrf-token inside all POST <form> in the application so it'll be included in the form when the form is sent.

5. Special case handling the sign-out. Because it is implemented in javascript and not in php like any other POST request, we have to manually send the csrf_token in its request.

2. Insecure Configuration PHPSESSID in Cookies

- Insecure session cookie, http_only not being set to true, no expiration time, etc.
- Since this web only supports HTTP, session hijack might happen. Hence, short-lived cookies might benefit more.
- The fix would be:

```
// Called before session_start()
session_set_cookie_params([
      'lifetime' => 0,
      'path' => '/',
      'domain' => '',
      'secure' => true,
      'httponly' => true
]);
```

3. Weak Error Handling

- Exceptions not properly sanitized before being returned to the user. For example:

```
throw HttpExceptionFactory::createInternalServerError('Cannot connect to the database');
```

- The fix would be:

```
error_log('Cannot connect to the database');
throw  HttpExceptionFactory::createInternalServerError('Something went wrong, please try again later');
```

4.Cross Side Scripting

To find this, we check every file where HTML is used and find the one that doesn't uses special encoding (such as htmlspecialchars()) or sanitization.

Inside /src/views/pages/jobs/[jobId]/index.php, the page to view details of a job, there is an attribute that isn't encoded.

```
<div class="content__rich-text">
    <?= $application->getStatusReason() ?>
</div>
```

```
<div class="content__rich-text">
    <?= $job->getDescription() ?>
</div>
```

This XSS Vulnerability can be exploited by having the stored job description as a malicious script such as:

Paired with the insecure session configuration, this vulnerability can get user logins.

To fix this, we can encode the attribute using htmlspecialchars() as shown in branch fix/xss.