AZIZ Souhayl AL2

Rapport Saé 304 – Découvrir le pentesting

Dans ce rapport, nous verrons 10 des différents CTF (Capture The Flag) que j'ai pu effectuer lors de cette SAE. En dernière page, il y a une annexe définissant tous les termes utilisé dans le document, les challenges sont classés par catégorie de la manière suivante :

1. [Web-client](#Webclient)
   1. [Javascript – Native code](#nativecode)
2. [Réseau](#Reseau)
   1. [ETHERNET – Transmission altérée](#transmialterer)
3. [Forensic](#Forensic)
   1. [Docker layers](#docker)
   2. [Active Directory – GPO](#adgpo)
4. [Cryptanalyse](#Cryptanalise)
   1. [Fichier – PKZIP](#pkzip)
   2. [Substitution monoalphabétique - César](#cesar)
5. [Web-serveur](#serveur)

* 1. [PHP – Injection de commande](#PHPinjec)
  2. [CRLF](#CRLF)
  3. [http – Directory indexing](#httpdi)
  4. [Insecure code management](#insec)

1. [ANNEXE](#annexe)

------------------------------------------------- WEB-client -------------------------------------------------

Une image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquementPour ce CTF, il n’y a pas d’énoncé ni d’indice présents, si ce n’est le titre « Javascript – Native code ». On a donc deux éléments à retenir :

**Javascript**: On va donc s’intéresser au code source  
**Native Code :**  On va devoir déchiffrer du native code

On démarre le challenge et on voit ceci :

Une image contenant texte, capture d’écran, logiciel, multimédia

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquementJe sélectionne « OK » et j’affiche le code source afin d’y trouver une balise <script> qui correspondrai à la balise javascript à laquelle on s’intéresse :

Native code

On voit que le code javascript est obfusqué, ici en Native code. Pour le désobfusquer, j’utilise un site (<https://www.dcode.fr/javascript-unobfuscator>), j’entre le code obfusqué et j’obtiens ceci :

function anonymous( ) {   
 a=prompt('Entrez le mot de passe');  
 if (a=='toto123lol') {  
 alert('bravo');  
 } else {  
 alert('fail...');  
 }  
}

function anonymous( ) { 🡸 Fonction Javascript  
 a=prompt('Entrez le mot de passe'); 🡸 Variable ‘a’ qui récupère le mot de passe entré  
 if (a=='toto123lol') { 🡸 Condition qui vérifie si la variable est égal à ‘toto123lol’  
 alert('bravo'); 🡸 Si c’est le cas alors la page web affiche ‘bravo’  
 } else {  
 alert('fail...'); 🡸 Sinon la page web affiche ‘fail…’  
 }  
}

Donc le mot de passe est toto123lol

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement------------------------------------------------------- Réseau -------------------------------------------------------

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquementUne image contenant texte, capture d’écran, Police, nombre

Description générée automatiquementPour ce CTF, nous avons un échange de trames Ethernet ;





On peut voir que l’inscription ‘>>> INGRESS >>>’ est présente 3 fois et ‘<<< EGRESS <<<’ qui font référence à 3 entrée et une sortie donc une question puis une réponse. On va regarder la composition d’une trame Ethernet :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement  
(Source 1 : https://repository.root-me.org/R%C3%A9seau/FR%20-%20Format%20des%20trames%20Ethernet.pdf)  
(Source 2 : <https://repository.root-me.org/R%C3%A9seau/FR%20-%20Les%20r%C3%A9seaux%20Ethernet%20-%20le%20format%20des%20trames.pdf> )

On peut voir à l’aide des documents fourni par rootme comment est constitué une trame Ethernet ainsi que son encapsulation IP. Détaillons donc les trames :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement  
C’est comme ça que nous allons couper chaque trame (elle sont numéroté au-dessus) : (capture de mon fichier de résolution de ce CTF)

On constate donc qu’il s’agit d’un échange ICMP en IPv6. Dans la dernière trame, les ‘?’ indiquent les éléments altéré. On trouve donc facilement l’EtherType car il reste le même, (8100) ainsi que dans la réponse le type de réponse ICMP (81), ainsi que le protocole ICMPv6 (3a). Aussi, on remarque une différence de TCI dans les vlans donc on n’utilisera pas la première trame. Il ne reste plus qu’à résoudre les adresses mac est IP. Il nous reste :

Une image contenant texte, Appareils électroniques, capture d’écran, Police

Description générée automatiquement

A ce stade j’avais du mal à avancer. Donc j’ai bien regardé les différences entre chaque trame et je me suis rendu compte que je n’ai pas vu la différence au niveau des octet d’identification qui sont les même pour la trame 1, 2 et 4 ( ) et pour la trame 3 (). Donc il ne reste plus que la trame 2. Comme il s’agit d’une réponse ICMP il suffit d’inverser les @dest et @src :

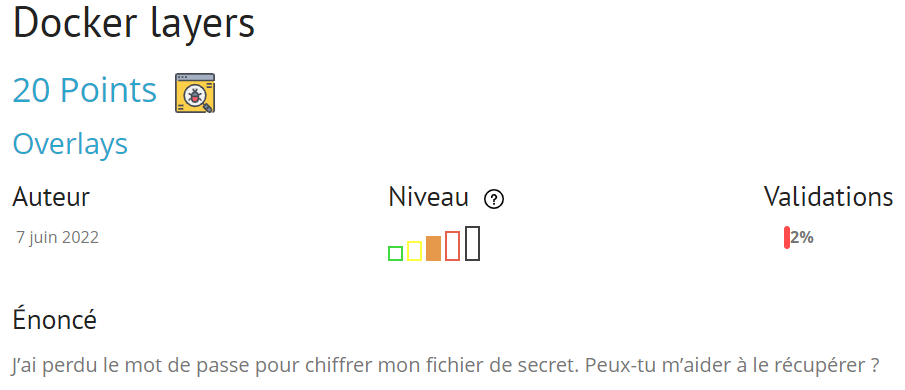
Une image contenant texte, capture d’écran, Police, affichage

Description générée automatiquement

Ici on à la trame 2 suivi de la dernière (l’ancienne trame altéré). En complétant tous les ‘ ?’ on obtient le flag suivant : **f9f781003afadab00b81**

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

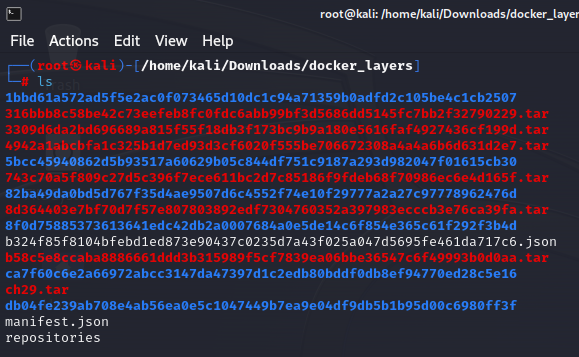
------------------------------------------------------ Forensic ------------------------------------------------------

Pour ce CTF, l’objectif est d’aider une personne à retrouver son mot de passe. En lançant le challenge, on obtient un dossier ch29.tar. Je décide donc de passer sur une machine virtuelle kali linux car je suis plus à l’aise dessus pour naviguer dans les dossier.

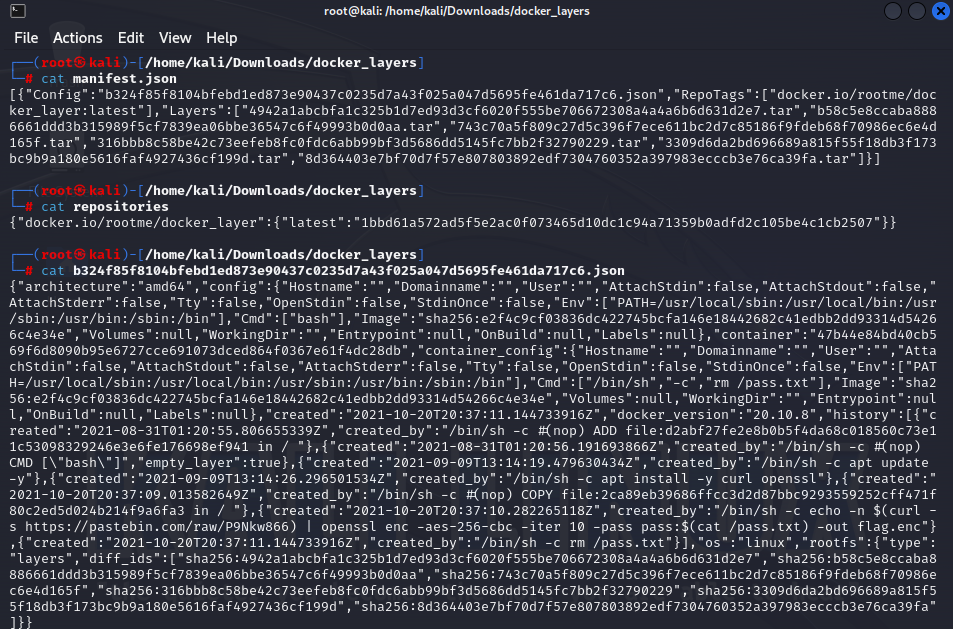
Dans un premier temp je regarde le type du dossier avec la commande **file** puis je le décompresse :



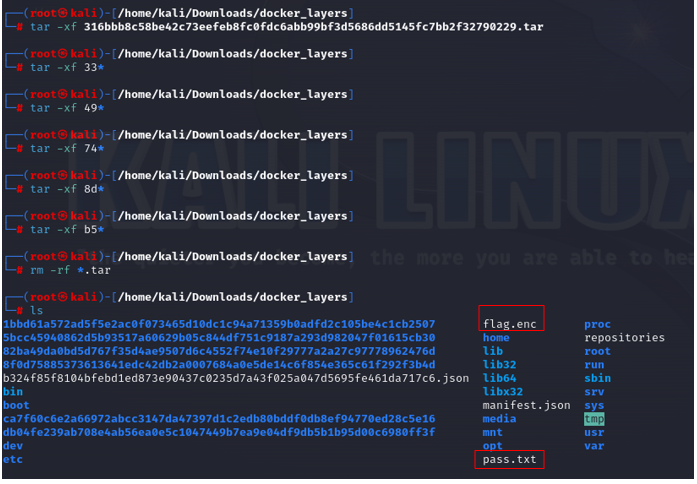
Et j’obtiens ceci :



On a beaucoup de dossier et de fichier. Je décide de commencer par regarder le contenu des fichiers et en lisant leur contenu, plusieurs éléments me semble intéressant donc je les garde en tête  :



Dans le première encadré, on a tous les dossiers non compressé et compressé de notre répertoire. Dans le deuxième encadré il y a’ latest’ suivi d’un dossier donc je me dis que je pourrais commencer mes recherches par ce dossier. Enfin dans le dernier encadré, on a la commande :  
**openssl enc -aes-256-cbc -iter 10 -pass pass :$(cat /pass.txt) -out flag.enc**  
cette commande encrypte un fichier probablement nommé  **flag**  en **flag.enc** en utilisant comme mot de passe le contenu du fichier **pass.txt**, ainsi qu’en utilisant l’algorithme de chiffrement **AES**  avec une taille de clé de **256** **bits** avec une itération de **10**. Ensuite je décide de me débarrassé des dossiers compressé en les décompressant puis en les supprimant. Il me reste :



On retrouve les fichier flag.enc et pass.txt donc pour vérifier qu’il s’agit bien des même que ceux de la commande Openssl, je regarde leur type et leur contenu :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Il s’agit bien d’un fichier encrypté avec Openssl donc je vais le décrypter en utilisant la même commande qu’on a trouvé :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

J’utilise la même commande sauf que je spécifie que le mot de passe est le fichier pass.txt   
(-pass file :pass.txt), j’indique le fichier à déchiffrer (-in flag.enc), la destination du résultat   
(-out flag.txt) et surtout j’utilise l’option ‘-d’ pour déchiffrer. Et j’obtiens le flag :

**Well\_D0ne\_D0ckER\_L@y3rs\_Inspect0R**

**Une image contenant texte, Police, capture d’écran, ligne

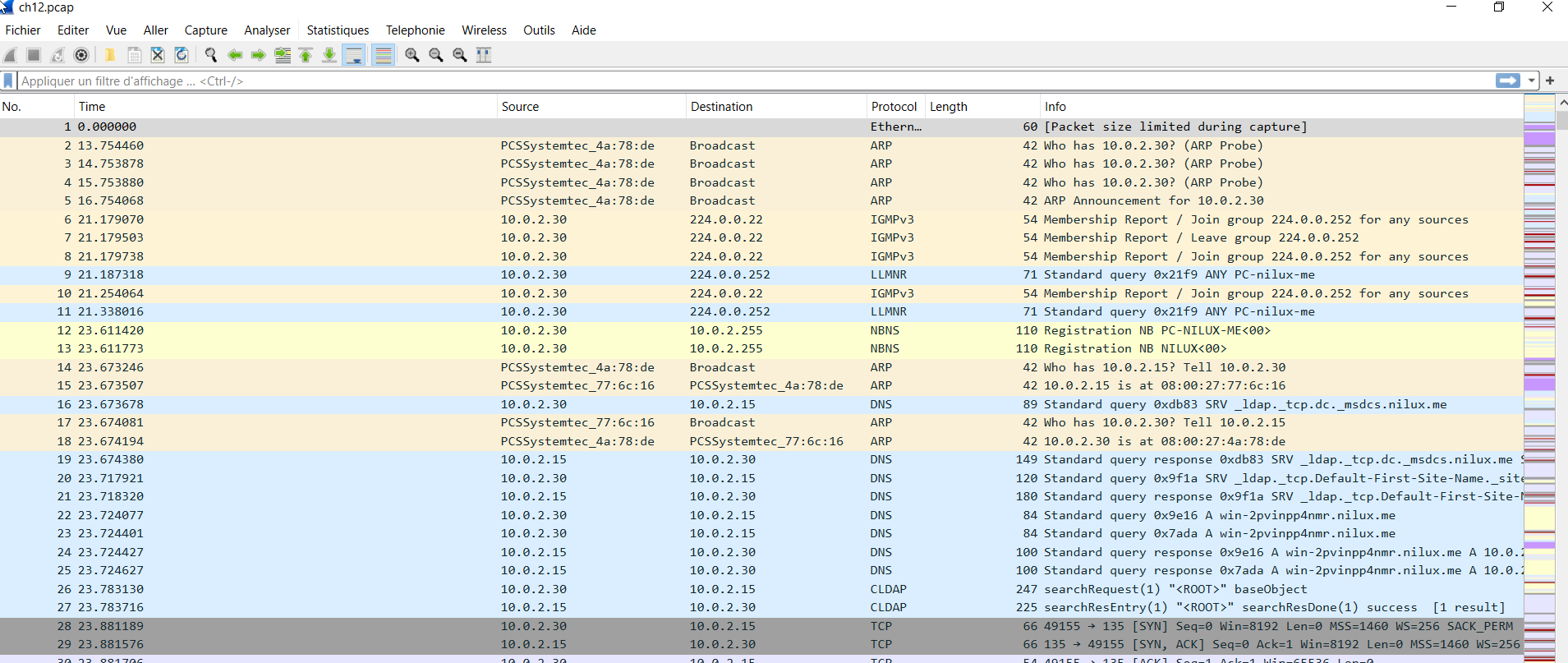
Description générée automatiquement**

Un autre challenge de Forensic que j’ai pu faire est :

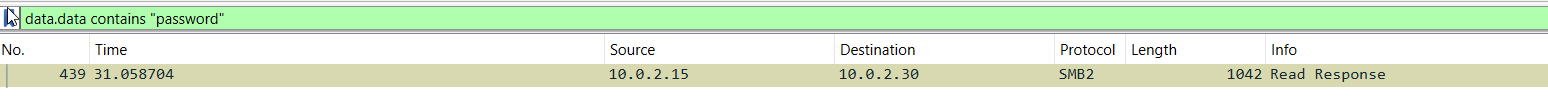
Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

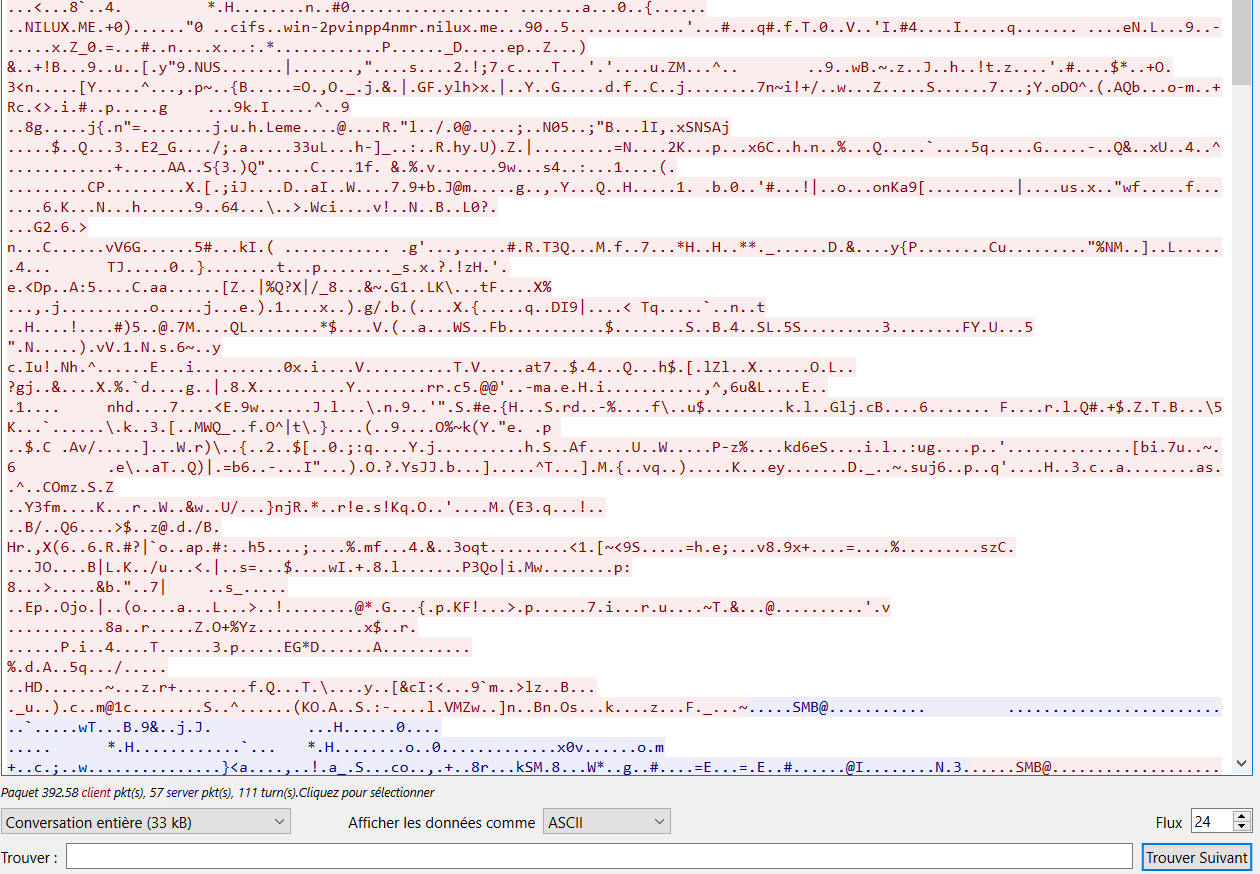
Pour celui-ci, nous avions un fichier de capture réseau Wireshark dans lequel nous devions retrouver le mot de passe de l’administrateur.



Il y avait beaucoup de trame dans la capture. Mais pendant ma période en entreprise précédente, nous cherchions un filtre Wireshark permettant de trouver un mot en claire directement dans la data des trames (data.data contains « chaine de caractère »). Donc je l’ai réutilisé :



J’ouvre la trame afin d’avoir tout le contenu :

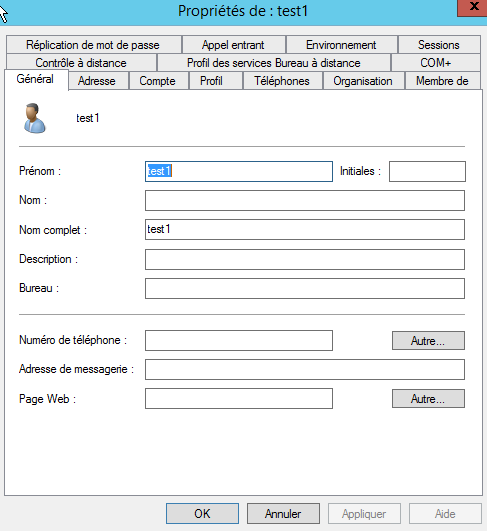
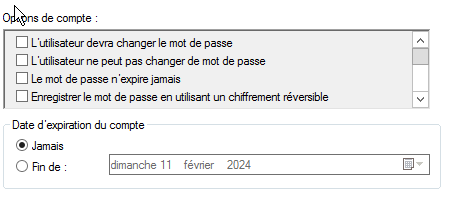


Je cherche ensuite le mot « password » dans la barre de recherche en bas :

Une image contenant texte, capture d’écran, nombre, logiciel

Description générée automatiquement

En lisant le contenu, je me rend compte qu’il s’agit du même contenu que ce qu » l’on trouve dans l’active directory lors de la création d’un utilisateur :

(Source : Contrôleur de domaine de mon entreprise)

<Groups clsid="{3125E937-EB16-4b4c-9934-544FC6D24D26}"><User clsid="{DF5F1855-51E5-4d24-8B1A-D9BDE98BA1D1}" name="Helpdesk" image="2" changed="2015-05-06 05:50:08" uid="{43F9FF29-C120-48B6-8333-9402C927BE09}"><Properties action="U" newName="" fullName="" description="" cpassword="PsmtscOuXqUMW6KQzJR8RWxCuVNmBvRaDElCKH+FU+w" changeLogon="1" noChange="0" neverExpires="0" acctDisabled="0" userName="Helpdesk"/></User><User clsid="{DF5F1855-51E5-4d24-8B1A-D9BDE98BA1D1}" name="Administrateur" image="2" changed="2015-05-05 14:19:53" uid="{5E34317F-8726-4F7C-BF8B-91B2E52FB3F7}" userContext="0" removePolicy="0"><Properties action="U" newName="" fullName="Admin Local" description="" cpassword="LjFWQMzS3GWDeav7+0Q0oSoOM43VwD30YZDVaItj8e0" changeLogon="0" noChange="0" neverExpires="1" acctDisabled="0" subAuthority="" userName="Administrateur"/></User>

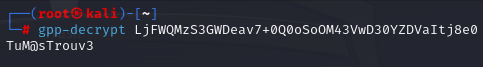
Donc je m’intéresse notamment à la partie qui suit « **Administrateur**«  et on a :

name="Administrateur"   
fullName="Admin Local"   
cpassword="LjFWQMzS3GWDeav7+0Q0oSoOM43VwD30YZDVaItj8e0"  
userName="Administrateur"

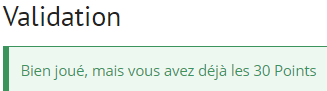
Donc le mot de passe est un mot de passe hashé, probablement avec un protocole Microsoft spéciale pour les GPO ;



Je cherche un moyen déchiffrer le code autour de GPP (Group Policy Preferences) et je trouve un outil sur kali appelé GPP-decrypt donc j’utilise la commande sur ma machine virtuelle kali et je trouve :



Le flag est donc **TuM@sTrouv3**

****

--------------------------------------------------- Cryptanalyse ----------------------------------------------------

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Pour ce CTF, nous avons un fichier zip protégée par un mot de passe (que nous n’avons pas évidement) . L’objectif est donc de casser le code et de récupérer le contenu du dossier zip.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Dans un premier temp j’ai chercher un moyen de cracker un mot de passe de ZIP sous kali et j’ai trouvé ce site : <https://medium.com/@rajendraprasanth/password-cracking-using-kali-67e0b89578df>

Dans ce site, je trouve un outil à installer sur kali et simple d’utilisation :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Donc j’ouvre ma machine kali et j’installe l’outil :

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Je lance donc la commande suivante :   
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Par défaut dans kali, il existe des dictionnaire de mot de passe pour faire des attaque par dictionnaire qui vont tester tout les mot de passe couramment utilisé recensé dans ce fichier. Chez moi ce fichier est dans le répertoire **/usr/share/wordlists/rockyou.txt.** Donc l’option ‘**-D**’ indique que l’on va chercher le mot de passe dans un dictionnaire, ‘**-p**’ indique le chemin du dictionnaire de mot de passe et enfin l’option ‘**-u**’ indique que l’on décompresse le fichier avec le code trouvé. Et au bout de 2-3 réel longues minutes, on obtient :

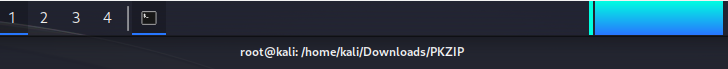
**PASSWORD FOUND!!!!: pw == 14535**

Avant de valider, j’ai essayé de le casser le mot de passe par force brut pour comparer le temp de résolution du mot de passe :

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Le temp est incomparable, je n’ai pas eu la patience d’attendre la fin du cassage de mot de passe par force brut car c’était beaucoup trop long (+ de 10 minutes)



Sans compter que le processeur de ma machine virtuelle était à 100%

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Pour ce CTF, L’objectif était de décoder un message. L’indice dans le titre est suffisant pour comprendre que le message suivant est chiffré avec la méthode César :

Une image contenant texte, Police, capture d’écran, algèbre

Description générée automatiquementLe chiffrement par la méthode césar est un concept simple de décalage de l’alphabet ou de la table ascii. Par exemple si je décale la lettre ‘**a**’ de 3 elle devient ‘**d**’ ou ‘**x**’ en fonction du sens de décalage (vers le début de l’alphabet ou vers la fin si on utilise seulement l’alphabet. Donc un message comme ‘**Bonjour**’ peut devenir ‘**Erqmrxu**’ ou ‘**Ylkglro**’ toujours en fonction du sens du décalage et seulement si on n’utilise que l’alphabet.

Pour décoder ce message je vais utiliser le site dCode (https://www.dcode.fr/chiffre-cesar) qui me permettra de décoder le message avec tout les décalage possible. Evidemment, après avoir essayer de décodé le message en une fois, j’ai pu voir que des mots apparaissaient clairement à chaque décalage mais pas tous en même temps, J’ai donc rassembler les mot et j’ai obtenus :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement  
Deuxième caractère décodé du message on obtient **deux**

Une image contenant texte, Police, capture d’écran, algèbre

Description générée automatiquement

Message décodé :

Un deux trois  
J’irai dans les bois  
Quatre cinq six  
cueillir des cerises  
Sept huit neuf  
Dans un panier neuf  
dix onze douze  
elles seront toutes rouges

En concaténant les premières lettres de chaque ligne suivi des dernières lettres de chaque ligne on obtient le flag : **ujqcsddessxsffes**

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

--------------------------------------------------- Web-serveur ----------------------------------------------------

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pour ce CTF, l’objectif est d’utilisé une injection SQL (L'injection SQL est une attaque informatique où des pirates insèrent du code malveillant dans les champs de saisie d'un site web pour manipuler la base de données associés au site)

Notons qu’un indice est que le mot de passe est dans un fichier appelé **index.php**

La page du challenge affiche ceci :

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

J’essaye dans un premier temp de voir si je peux accéder directement au fichier index.php depuis l’url :

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Visiblement rien ne change. Donc je rentre une adresse dans la case, je prend celle proposé (**127.0.0.1**) et j’obtiens ceci :

**Une image contenant texte, Police, capture d’écran

Description générée automatiquement**

Je comprends que le serveur exécute le code php sur une machine dans un environnement linux car il exécute un ping et que le ttl (time to live) de la requête ICMP est 64 par défaut sur linux. Donc j’essaye d’y ajouter une autre commande en bash (langage des terminaux linux) :

Une image contenant texte, capture d’écran, Police, document

Description générée automatiquementIci j’exécute la commande ls pour voir le contenu du répertoire actuel et on voit **index.php** qui est écrit en dessous de la réponse du ping.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquementDonc j’affiche le contenu de l’index.php :

Ici on voit que le contenu du fichier **index.php** n’indique pas de mot de passe mais affiche la case de la page.

Donc je penses à afficher tout le contenu du dossier actuel (même les caché) :  
(**127.0.0.1 && ls -a**) 🡺 -a pour ‘all’

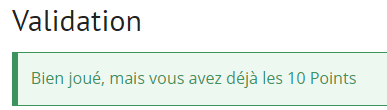
Une image contenant texte, reçu, Police, algèbre

Description générée automatiquement

Je vois le ficher caché **.passwd**. En visualisant son contenu je trouve ceci (**127.0.0.1 && cat .passwd**) :

* Servicepingsupersecure

Donc le flag est : **S3rv1ceP1n9Sup3rS3cure**

****

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pour ce CTF, l’objectif est d’injecter des données erronées dans le fichier de journalisation :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Fichier de journalisation à corrompre

Pour m’aider, j’ai cherché sur internet ‘**crlf** ‘ et je me suis aidé de la page d’aide de RootMe  
(<https://but.pro.root-me.org/CRLF>)

J’ai compris que cette technique consistait à modifier les URLs de recherche à l’aide de caractère de la table ascii codé en Hexadécimal. Donc j’ai entré des première information dans les cases pour voir l’url de requête :

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

Je remarque que l’url ajoute **?username=salut+&pasword=test** et que ‘**salut**’ est affiché dans le fichier de journalisation avec un message ‘failed to authenticate.’

Mon objectif est donc de faussé le fichier de journalisation insérant comme information que je suis connecté. Pour cela je comprends qu’il faut que je fasse une phrase du type ‘**Souhayl authenticated.**’ Mais sachant que je n’ai le mot de passe d’aucun utilisateur, il faut que la requête prennent aussi un utilisateur et que j’aille à la ligne afin qu’il affiche quand même ‘**failed to authenticate.**’

Il faut donc maintenant chercher quel caractère me permet d’aller à la ligne et et de faire un espace en hexadécimal dans la table ascii (<https://fr.wikibooks.org/wiki/Les_ASCII_de_0_%C3%A0_127/La_table_ASCII>) :

Une image contenant texte, capture d’écran, nombre, Parallèle

Description générée automatiquement

Je trouve l’espace et le saut de ligne donc je modifie l’url de façons à ce que ce message soit inséré dans le fichier de journalisation :

Souhayl authenticated.  
admin failed to authenticate.

Donc après ‘**?username=’** j’ajoute 🡺 **Souhayl%20authenticated.%0Aadmin**ce sera donc « Souhayl authenticated.(retour à la ligne) admin » qui échouera l’authentification mais le journal l’affichera autrement.

On a donc :

http://challenge01.root-me.org/web-serveur/ch14/?username= **Souhayl%20authenticated.%0Aadmin**&password=test

Et on obtient bien le résultat :

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Flag :  **rFSP&G0p&5uAg1%**

**Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement**

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pour ce CTF nous avons peu d’indice, mais l’indice devrait être dans le code source :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Lorsque j’affiche le code source, on voit un commentaire (**include(‘‘admin/pass.html’’)**) et ayant eu l’occasion de réalisé des petits sites web, je sais que les pages web sont en fait plusieurs fichier dans un répertoire sur une machine (généralement un serveur web). Donc depuis l’url j’essaie d’afficher la page ‘**admin/pass.html**’ :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Après une longue réflexion et plusieurs tentatives inutile (pass.php, index.html, password.html etc..) je me suis dit qu’il fallait tout simplement que j’essaie de remonter dans le répertoire (d’où le tire **Directory indexing**) pour voir tous les fichiers et simplement ajouter ‘**/admin/**’ à l’url :

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

On voit bien qu’il y a le fichier pass.html mais aussi un autre répertoire (on ne compte pas le ‘**Parent directory/**’ qui correspond au dossier ‘**ch4/**’)  donc on a (backup/) :

Une image contenant texte, capture d’écran, Police, logiciel

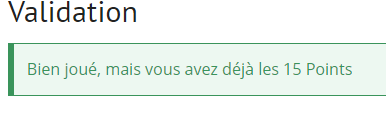
Description générée automatiquement

On voit le fichier admin.txt donc je suppose qu’il contient le flag

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Fag : **LINUX**



Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pour ce dernier CTF, on a des indices qui nous aides à savoir « **serveur de gestion du code source** » ainsi que les documentations fourni par la plateforme :Une image contenant texte, Police, capture d’écran, conception

Description générée automatiquement

Avec ces indices je pense tout de suite à Git qui est un outil permettant de suivre les changement dans le code (gestion de code source) généralement utilisé par les développeur dans n’importe quel domaine. Dans notre cas il s’agit d’un git utilisé dans le cadre d’un serveur web. En lisant la documentation de RootMe (https://but.pro.root-me.org/Outil-Git-Dumper)  
je vois qu’il y a un outils permettant de récupérer le git d’un serveur web (git-dumper) donc dans un premier temp je verifie qu’il y a bien un git disponible pour cet page web :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement  
(Page d’origine du challenge : http://challenge01.root-me.org/web-serveur/ch61/)

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement  
(git du challenge : <http://challenge01.root-me.org/web-serveur/ch61/.git/>)

On peut y voir différent fichier. Mais pour être plus à l’aise, je récupère le contenu avec Git-dumper sur ma machine kali :

Une image contenant texte, capture d’écran

Description générée automatiquement

Une fois tout chargé, on :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

On retrouve des dossiers et des fichiers (les fichiers ne présentent pas d’information pertinente si ce n’est le hash du mot de passe du compte admin et son pseudo (admin), sachant que le hash est en sha256, il n’est donc pas réversible.). Je reste sur la piste des indices à savoir git donc je vais voir ce qu’il y a dans le dossier ‘**.git**’ :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Un dossier de journalisation est présent (**logs**) et m’intéresse donc je vais voir ce qu’il contient :

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

Je trouve un fichier HEAD dont j’affiche le contenu et je vois que **John** à réalisé un changement de mot de passe. Donc ayant déjà utilisé Git, je savais que je pouvais voir les changements de commit (git commit permet d’enregister localement la version que l’on vient d’éditer )avec la commande **git diff <numéro de commit1> <numéro de commit2>,** on peut voir ce qui a été modifié entre 2 commit. Ici les commit qui nous intéresse sont celui ou John s’authentifie et celui ou il change de mot de passe :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Le rouge indique ce qui a était supprimé, et le vert ce qui a était ajouté ou changé, dans notre cas, il s’agit du mot de passe de admin qui passe de ‘‘**admin**’’ à ‘‘s3cureP@ssw0rd’’

Le flag est donc : **s3cureP@ssw0rd**

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

------------------------------------------ ANNEXE ------------------------------------------

**JavaScript :** Langage de programmation utilisé pour rendre les sites web interactifs

**Native code :** Code informatique directement compréhensible par le processeur d'un ordinateur.

**Obfusquation** : Processus de rendre intentionnellement du code source plus complexe ou difficile à comprendre, souvent utilisé en sécurité informatique pour décourager la compréhension ou la modification non autorisée. Wireshark : Outil de capture et d'analyse de paquets réseau.

**Logs** : Enregistrements détaillés des activités d'un système ou d'une application.

**Table ASCII** : Table de codage qui attribue des nombres aux caractères, utilisée dans l'informatique.

**Zip** : Format de compression de fichiers.

**Forensic** : Méthode d'investigation pour collecter et analyser des preuves numériques.

**Cryptanalyse** : L'art de décrypter des codes ou de casser des systèmes de chiffrement.

**Capture** **the** **Flag** : Compétition de cybersécurité où les participants résolvent des défis pour trouver des "drapeaux" virtuels.

**Trame** : Unité de données transmise sur un réseau.

**ICMP** : Protocole de contrôle de messages Internet, utilisé notamment pour le diagnostic réseau.

**Active** **Directory** : Service de gestion des identités et des accès dans les environnements Windows.

**Kali** : Distribution Linux spécialisée dans la sécurité informatique et les tests de pénétration.