MOTA Yuri

CTF Night Hack: Documentation de walkthrough des challenges personnels

/!\

Ce document décrit seulement les étapes de résolutions et le bon fonctionnement.

Toutes les réponses directes (flag) sont situées dans le panel d'administration de la plateforme CTFd pour chaque challenge ou dans un fichier "réponses.txt".

De plus, tous les fichiers de résolutions de challenge (source, exécutables) sont situés dans un dossier "Réponses".

Tous les challenges présentés ici ont été conçus et rédigés par moi-même.

/!\

INTRODUCTION

Nom : ROT

Type : Cryptographie

Solution :

Pour ce challenge il suffit de déchiffrer le message avec un outil de décodage ROT47.

Nom : Fingerprint

Type : Forensic

Solution :

Ici, il n'y a pas besoin d'outil externe. Il suffit de dézipper le fichier JPG (« unzip » sous linux).

Nom : Pastebin

Type : OSINT

Solution :

Le flag se trouve à la page suivante https://pastebin.com/a1gigiyF.

Pour le trouver il suffit de rechercher les tags CTF, FLAG, INTRODUCTION ou NHM2I sur la plateforme.

Nom : Hello world !

Type : Reverse

Solution :

On peut utiliser l'outil ghidra pour décompiler le fichier binaire et trouver le flag dans le code source. Il est stocké dans un « char* ».

Il suffit de créer un nouveau fichier avec ghidra. Ajouter avec « i » un fichier soit l'exécutable « hello-world » et de lancer l'outil d'analyse. Puis de l'ouvrir pour l'analyser.

En cherchant dans le panneau de gauche « Symbol Tree > Functions > main », on trouve dans la décomposition hexadécimale le flag dans « main:00101148 ».

FACILE

Nom : XOR

Type : Cryptographie

Input : 674

Solution :

Le but est de trouver le bon input sans modification nécessaire du programme.

Pour cela il faut s'avoir que certains caractères sont des associations de plusieurs caractères (Unicode). C'est le cas ici avec le caractère « o » ou U+006F (first_composition) à laquelle on peut ajouter deux autres caractères qui donneront au final un caractère spécial soit la clé.

On doit donc déterminer la valeur des deux autres compositions en hexadécimal (second_composition et third_composition) à l'aide du commentaire, puis les ajouter à first_composition pour trouver la somme soit l'input du programme.

Le programme ajoute une valeur en plus pour déterminer la clé, et fait un XOR avec sur le message chiffré pour trouver le flag qui s'affichera.

Nom : Data stream CAPture

Type : Forensic

Solution :

Avec Wireshark il suffit de regarder sur le fichier, le flux 5 dans les trames TCP.

Il faut faire un clic droit sur une trame TCP puis « Follow > TCP Stream » et changer la valeur du « Stream » à 5.

Une variable « msg » est présente. Celle-ci contient une valeur avec un « = » à la fin. On comprend facilement qu'il s'agit d'une valeur en base64 à décoder pour trouver le flag.

Nom : Overpass

Type : OSINT

Solution :

Ici, il ne suffit pas de faire du « Reverse Image Search » car vous trouverez facilement que l'image a été prise à Rome.

Le nœud, chemin ou relation correspondant au corps du flag est attribué au bâtiment situé en arrière-plan, à droite. Ce bâtiment est le Tribunal de surveillance de Rome.

A l'aide du lien de documentation de l'API Overpass, on peut faire une recherche sur le mot « law » par exemple, et on tombe sur un tag (amenity=courthouse), que l'on peut rechercher dans l'assistant « Query Wizard » en plaçant bien la bounding box (carte) autour du bâtiment.

Résultats de recherche pour « law » - OpenStreetMap Wiki

Après avoir construit et exécuté la requête on trouve bien le nœud correspondant au bâtiment, en cliquant dessus. Et on peut l'inclure au corps du flag.

/!\

Le challenge qui suit a été annulé car initialement prévu pour un seul conteneur pour tous les challengers.

Le risque étant de donner l'accès au fichier /etc/passwd qui permettrait de changer le mot de passe root avec n'importe quel binaire avec le SUID actif, ayant comme propriétaire root et d'avoir un accès complet au conteneur commun (compromission, évasion).

/!\

Nom : UNIX command abusing SUID

Type : PWN

Solution :

Ici pour accéder au fichier il faut obligatoirement obtenir un accès à l'utilisateur root car le fichier n'est lisible que par lui.

Pour cela, d'après le titre du challenge, on comprend qu'il faut trouver un fichier binaire permettant d'exécuter une commande avec les droits du propriétaire soit root dans notre cas.

Pour trouver un fichier contenant le SUID actif on peut par exemple exécuter la commande « find / -perm -u=s -type f 2>dev/null ». On voit que la commande « cp » peut être utilisé.

De là, il existe beaucoup de possibilité de retrouver le flag situé dans « /home/user/flag » avec cette commande.

Un exemple est de changer le mot de passe root pour pouvoir récupérer l'accès root et lire le fichier.

Voici une procédure le permettant :
https://medium.com/go-cyber/linux-privilege-escalation-with-suid-files-6119d73bc620.

Le challenge suivant vient remplacer le précédent :

Nom : Wrap de valeur, sauce overflow

Type : PWN

Solution :

Pour résoudre ce challenge, il faut comprendre l'algorithme du programme.

Le but étant de créer un dépassement de tampon (buffer overflow).

On peut inspecter le script pour se l'approprier.

```
import numpy as np

int64_tab = np.array([0], dtype=np.int64)
val = int(input("A vous de tester !\n"))
if val == -1:
        exit()

int64_tab[0] = val
int64_tab[0] = (int64_tab[0] * 7) + 1
print(int64_tab[0])
if int64_tab[0] == -1:
        print("Vous avez trouvé le flag!")
```

int-overflow.py

On peut voir que la valeur attendue est -1.

La bibliothèque `numpy` (alias : `np`) est utilisée pour la création du tableau d'entier sur 64 bits.

Le programme lit et stocke l'entier dans `val`. Si ce dernier est égal à -1 alors le programme se termine.

La valeur de val est initialisée dans `int64_tab` à l'indice 0.

La valeur de cette dernière variable est multipliée par 7, le tout est additionné à 1.

On peut tester avec python2 (non python qui affiche les valeurs entières avec des exposants (moins visuel) de trouver la valeur qui est demandée par le programme (et qui rappelons-le, convertit en base 16, donne le flag).

Les valeurs limites supportées pour les entiers sur 64 bits sont :

-9223372036854775808

+9223372036854775807

Dans la logique algorithmique il suffit de prendre la valeur supérieure, la multiplier par deux et de diviser le tout par l'opérateur inverse (/) à celui présent dans le programme (*).

```
>>> 9223372036854775807 * 2
18446744073709551614L
>>> 18446744073709551614 / 7
2635249153387078802L
```

On peut alors tester la valeur `2635249153387078802` dans le programme (on aura pris le soin de tester en amont différentes valeurs pour bien comprendre son fonctionnement).

Retour:

```
$ python test.py
A vous de tester !
2635249153387078802
/home/kali/test.py:9: RuntimeWarning: overflow encountered in scalar
multiply
  int64_tab[0] = (int64_tab[0] * 7) + 1
-1
Vous avez trouvé le flag!
```

La valeur de int64_tab[0] après calcul n'étant pas dans la plage des entiers de 64 bits (au dessus) la valeur est "wrappé" et ramené à -1.

Finalement on convertit la valeur décimale `2635249153387078802` en base 16 (hexadécimale).

Cela donne `2492492492492492`. Le flag est donc `NHM2I{2492492492492492}`.

Nom : Some base

Type : Reverse

Solution :

Si on décompile le fichier binaire dans ghidra comme pour le challenge d'introduction au reverse, on voit dans la fenêtre de décompilation qu'un test conditionnel est réalisé.

On peut alors tester d'ajouter la valeur qui valide la condition (0x466c6167) en input dans l'exécution du programme.

On trouve alors le flag.

Nom : Flag Master Chief

Type : Web

Solution :

Pour trouver le flag, il faut cliquer le « Play » de la quatrième vidéo proposé en bas de la page d'index. En passant la souris sur le « Coming Soon » on voit aperçoit un hyperlien.

En cliquant dessus on arrive sur une page cachée. En inspectant le code on voit dans la balise <head> qu'un chemin relatif vers un nouveau fichier est présent « secret/flag.html ».

En l'ajoutant à la racine du site on trouve le flag affiché sur la page.