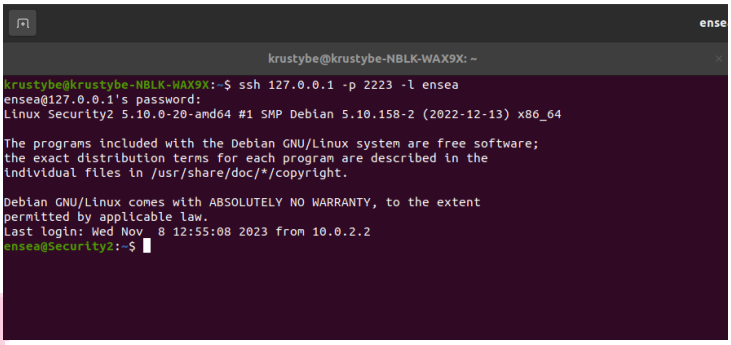
TP n°2 : Software Security

I- Let's go

Une fois la machine virtuelle configurée et ouverte. Nous pouvons nous connecter à cette dernière via connexion **ssh** comme nous pouvons le vopir si dessous :



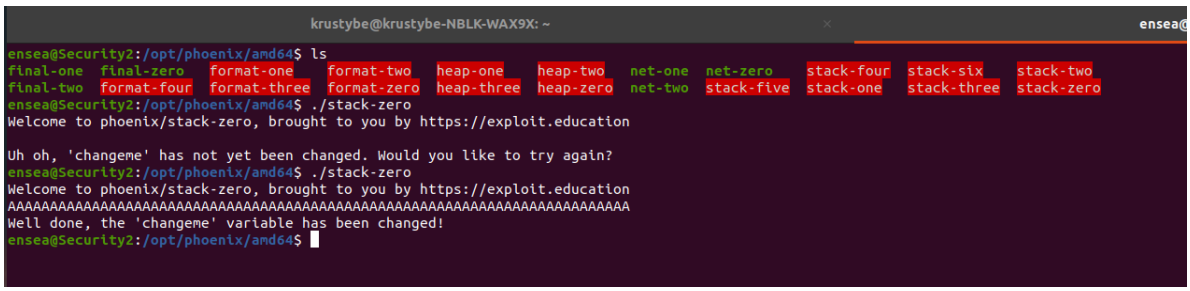
Chaque élément de cette commande à une fonction précise :
- **ssh** : correspond au programme client SSH que j'utilise pour me connecter à un serveur distant de manière sécurisée.
- **127.0.0.1** : correspond l'adresse IP du serveur auquel je me connecte. Dans mon cas, c'est l'adresse IP locale qui pointe vers la machine sur laquelle j'exécute la commande.
- **-p 2223** : correspond à l'option pour spécifier le port SSH. Dans mon cas, le port est défini à 2223 comme demandé dans l'énoncé.
- **-l ensea** : correspond à l'option pour spécifier le nom d'utilisateur avec lequel je me connecte.

II- Stack

1. Stack Zero

Objectif :
Trouver un moyen de changer le contenu de la variable "**changeme**".
Ce niveau introduit le concept selon lequel la mémoire peut être accédée en dehors de sa région allouée, comment les variables de la pile sont disposées, et que la modification en dehors de la mémoire allouée peut modifier l'exécution du programme.

Solution :
La solution que j'ai trouvée utilise une vulnérabilité de débordement de tampon. Cela fonctionne en exploitant le fait que la fonction **gets()** ne vérifie pas la taille du tampon dans lequel elle lit les données, ce qui permet à l'utilisateur de saisir plus de données que la taille allouée pour le tampon. Pour pallier à cette défaillance de sécurité, on peut utiliser la fonction **fgets()** qui permet de spécifier la taille maximale des données à lire.



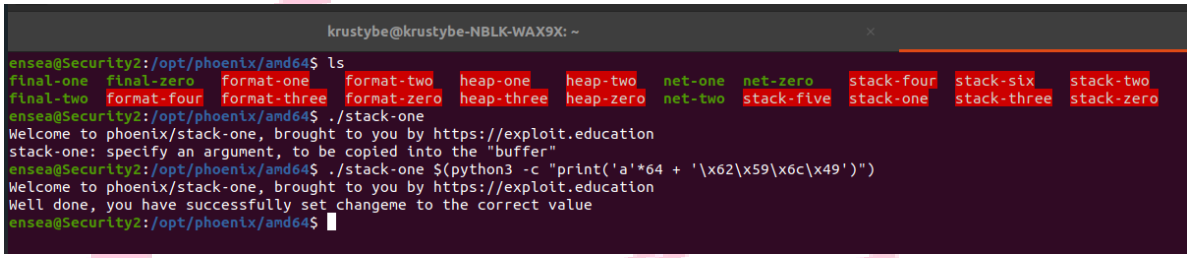
2. Stack One

Objectif :

Trouver un moyen de changer le contenu de la variable "**changeme**" en **0x496c5962**.
Ce niveau examine le concept de modification des variables pour leur attribuer des valeurs spécifiques dans le programme, ainsi que la manière dont les variables sont disposées en mémoire.

Solution :

La solution que j'ai trouvée utilise une vulnérabilité de débordement de tampon comme dans l'exercice précédent. Cela fonctionne en exploitant le fait que la fonction **strcpy()** ne vérifie pas la taille de la chaîne d'entrée, ce qui permet un débordement de tampon. Avec cette faille, je peux donc écrire dans la mémoire adjacente à la variable **changeme** et donc la modifier.
Pour pallier à cette défaillance de sécurité, on peut utiliser la fonction **strncpy()** qui permet de spécifier la longueur maximale de la copie et d'empêcher un dépassement de tampon.



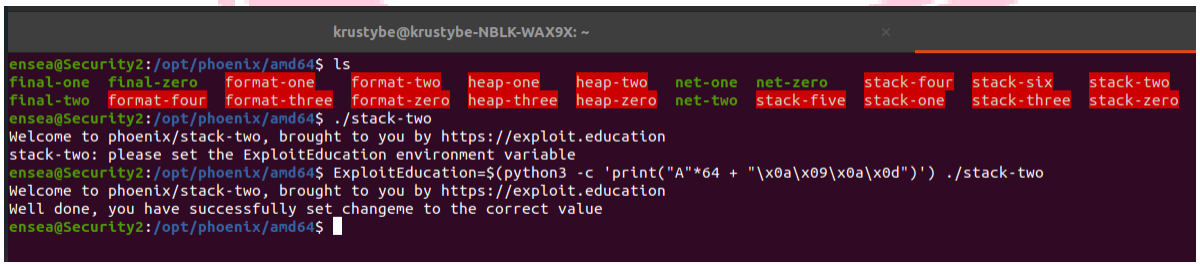
2. Stack Two

Objectif :

Trouver un moyen de changer le contenu de la variable "**changeme**" en **0x0d0a090a**.

Solution :

La solution que j'ai trouvée utilise une vulnérabilité de débordement de tampon comme dans les exercices précédents. Comme pour stack-one la faille de sécurité réside dans la fonction **strcpy()**. La grande différence avec l'exercice précédent est que dans celui-là on utilise une variable d'environnement. Cela démontre comment des données externes, peuvent être utilisées pour exploiter les vulnérabilités de débordement de tampon dans un programme.
Comme pour l'exercice précédent, pour pallier à cette défaillance de sécurité, on peut utiliser la fonction **strncpy()** qui permet de spécifier la longueur maximale de la copie et d'empêcher un dépassement de tampon.



III- Format

1. Format Zero

Objectif :

Modifier la variable "**changeme**".

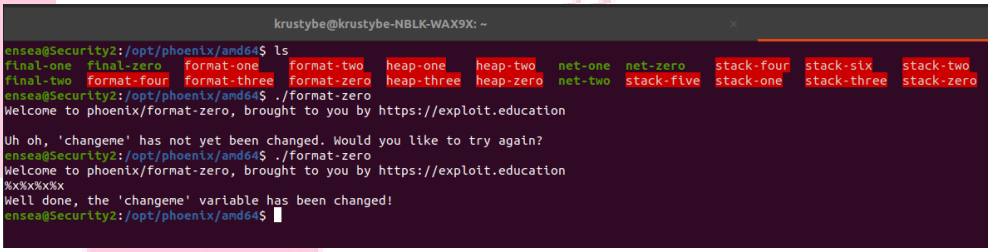
Ce niveau introduit les chaînes de format et comment les chaînes de format fournies par un attaquant peuvent modifier l'exécution du programme.

Solution :

Dans cet exercice, la faille de sécurité réside dans le fait que le programme utilise la fonction **sprintf()** avec une chaîne de format fournie par l'utilisateur sans validation, ce qui peut conduire à une attaque de format de chaîne, permettant à l'utilisateur de lire et de modifier des valeurs de la pile, y compris la valeur de la variable **changeme**.

Dans mon cas, j'utilise une chaîne de format qui contient plusieurs spécificateurs de conversion (**%x**). En utilisant ces spécificateurs, j'indique à la fonction de lire les valeurs suivantes dans la pile. Ce qui pour but à la fin modifier la valeur de **changeme**.

Pour pallier à cette défaillance de sécurité, on peut utiliser la fonction **snprintf()** qui est plus sécurisé que **sprintf()**, avec cette fonction on prend un argument supplémentaire qui va servir à spécifier la taille maximale du tampon de sortie.



2. Format One

Objectif :

Modifier la variable "**changeme**".

Ce niveau introduit les chaînes de format et comment les chaînes de format fournies par un attaquant peuvent modifier l'exécution du programme.

Solution :

Comme dans l'exercice précédent, la faille de sécurité réside dans le fait que le programme utilise la fonction **sprintf()** avec une chaîne de format fournie par l'utilisateur sans validation. Cependant contrairement à l'exercice format-zero, la on utiliser le spécificateur de conversion **%32x**. Faire cela indique à la fonction **sprintf()** d'afficher 32 valeur hexadécimales, ce qui va donc occuper plus d'espace dans la sortie. Puis, nous écrivons ce que nous valons dans **changeme.**

Pour pallier à cette défaillance de sécurité comme dans l'exercice précédent, on peut utiliser la fonction **snprintf()** qui est plus sécurisé que **sprintf()**, avec cette fonction on prend un argument supplémentaire qui va servir à spécifier la taille maximale du tampon de sortie.