TP - **PROJET AVL - 2025**

January 8, 2025

Environnement de travail

Le projet se fera sur la même VM que les TPs.
Connectez-vous à la VM en utilisant un client SSH: `$> ssh ...`
Lancez la commande `$> seclab tutxx` afin de mettre en place l'environnement approprié pour chaque question
Un serveur SAMBA est fonctionnel sur la VM locale pour le partage de fichiers avec l'OS hôte.
Activez la création de crash dump `$> ulimit -c unlimited`. Les crash dumps seront créés dans le répertoire courant.

Fuzzing

L'objectif de cette partie est de comprendre l'utilité du Fuzzing dans la détection de vulnérabilités dans le code source ou binaire. Nous nous focaliserons ici sur l'outil AFL (American Fuzzy Lop). Pour ce faire, répondez à ces questions :

1. Expliquez le fonctionnement du Fuzzing à l'aide de l'outil AFL.
2. Expliquez de manière précise l'étape d'instrumentation de l'outil ainsi que son intérêt.
3. Trouvez les vulnérabilités dans ce binaireen utilisant AFL.

indication: En posant les bonnes questions à ChatGPT ou à un outil équivalent suffit pour répondre aux questions 1 et 2.

Exécution symbolique

Le but de cette partie est de se familiariser avec le concept de d'analyse symbolique.

1. Étudier, puis présenter avec des exemples le concept de l'analyse symbolique
   - LIEN 1
   - LIEN 2
2. Écrivez un programme utilisant `Angr` afin de trouver automatiquement l'entrée donnant lieu à l'affichage par `$> ~/tuts/lab01/tut01-crackme` du message `"Password␣OK␣:)"`.
3. Comment pourrait-on utiliser cette méthode pour exploiter la vulnérabilité de type buffer overflow du fichier `$> ~/tuts/lab03/tut03-stackovfl` par exemple.

Erreur Logique et Race condition

Soit le code vulnérable projetpart3.c et sa version compilée projetpart3 qui est liée à un jeu vidéo `snake` basique.

La fonction snake_main est le point d'entrée du jeu.

1. Affichez le flag (`/proc/flag`) en passant toutes les étapes du programme `projetpart3`

indication:

1. Analyser le code assembleur de la fonction `absolute()`. La succès est dans les limites de la représentation des nombres négatifs en complément à deux.

2. La deuxième étape peut être réussie en étant plus rapide que le programme pour lire le fichier de mot de passe créé et qui est supprimé juste après (Race condition vulnerability).

3. La dernière étape, qui est d'afficher le flag, est liée à une vulnérabilité dans le code de `snake_main`.
4. Vous pouvez utiliser le code 1 pour intéragir avec le programme.

Listing 1: template.py

```python
#!/usr/bin/env python2

import time
import os
import pty

(pid,fd) = pty.fork()

if pid == 0 : # le fils
    os.execle("./target","./projetpart1",os.environ)
    exit(0)
else : # le pere

    lock = "/tmp/.lock-%d" % pid
    print("pid= %s, lock = %s" %(pid,lock))

    os.write(fd,"83648\n")

    while True:
        os.write(1,os.read(fd,1))

    os.wait()
```

Attaques liées au Tas (Heap)

1. En étudiant entre autres cette documentation :

   - Lien1
   - Lien2
   - Lien3

   expliquez en détail **deux** attaques liées au tas de votre choix.

2. soit le code vulnérable suite : heap_vul.c. En vous basant sur votre compréhension de la gestion du tas, tentez de forcer l'appel de la fonction **winner()**.

indication:

- Pour compiler le programme : `gcc -m32 -no-pie simple.c -o simple`
- Vous pouvez utiliser la commande gdb `info proc map` ou `vmmap` de pwndbg.
- Vous pouvez utiliser la commande pwndbg `heap` pour connaître la structure courante du tas (heap)