Rapport Projet : ret2win

Guillaume RAISON - MCS 26.3

Date du rapport : 08/01/2026

1. Identification de la vulnérabilité

En analysant le fichier `ret2win.c`, j'ai repéré la faille dans la fonction `vulnerable_function`. Le problème vient de l'utilisation de `strcpy` à la ligne 21. Cette fonction copie l'entrée utilisateur (`input`) dans le `buffer` sans vérifier si la taille dépasse la capacité prévue.

Comme le buffer est déclaré avec une taille de **64 octets** (`char buffer[64]`), si j'envoie plus de données que prévu, je vais écraser la mémoire qui suit.

2. Analyse et calcul de l'offset

Avant d'essayer d'exploiter la faille, j'ai vérifié le fichier `Makefile`. On voit que le programme est compilé avec `-fno-stack-protector` et `-no-pie`.

Cela signifie deux choses importantes pour moi : 1. Il n'y a pas de protection "Canary" qui empêcherait le débordement. 2. Les adresses des fonctions ne changent pas à chaque exécution.

Calcul de la distance (Offset) : Pour prendre le contrôle, je dois écraser l'adresse de retour (Saved RIP) stockée sur la pile. Voici comment j'ai calculé ce qu'il fallait remplir avant de l'atteindre : * Le tampon (`buffer`) prend **64 octets**. * Juste après, il y a le pointeur de base sauvegardé qui prend **8 octets**. * L'adresse de retour est juste derrière.

Donc, mon point de rupture est à : $64 + 8 = 72$ octets.

3. Stratégie d'exploitation

Mon but est de forcer le programme à exécuter la fonction `win_function` au lieu de finir normalement.

Mon plan d'attaque : 1. **Injection via argv :** J'utilise le premier argument du programme pour injecter mon payload. Comme `strcpy` s'arrête au premier octet nul (`\x00`), je dois m'assurer de retirer les zéros terminaux de mon adresse lors de la construction du payload. 2. **Padding :** J'envoie **72 caractères quelconques** pour remplir le buffer et le RBP sauvegardé. 3. **Détournement (Override RIP) :** Juste après, je place l'adresse de `win_function`.

Contrainte technique (Stack Alignment) : Lors de mes tests, sauter directement au début de `win_function` provoquait un crash au moment de l'exécution de `system("/bin/sh")`. Ceci est dû à une contrainte d'alignement de la pile sur 16 octets imposée par l'architecture x64 pour les instructions SIMD. **Solution :** J'ai ajouté **+1** à l'adresse de `win_function` pour sauter l'instruction

`push rbp`. Cela décale la pile de 8 octets, rétablit l'alignement correct, et permet au shell de s'ouvrir sans planter.

4. Résultats

J'ai automatisé cette attaque avec le script `exploit.py` (en utilisant `pwntools`). Le script récupère l'adresse de `win_function` directement depuis le binaire, construit le payload de 72 octets + l'adresse, et lance le processus.

Le message "SUCCÈS! Fonction win() exécutée!" apparaît bien, confirmant que le flux a été détourné. J'obtiens ensuite un shell interactif.