Secure programming

M. Dechaux

Antoine Puissant

16 septembre 2015

Résumé

Table des matières

1 Projet

Lastname_Firstname_SEC_secprog.zip
Programmation sécurisée permettant d'éviter les erreurs classiques

- Static analysis : rapport du code ligne à ligne (connaît les fonctions)
- Code checking :
- Analyse dynamique : Analyse du comportement du programme
- Reverse engineering : pas sur les applications sur un soft propriétaire
- Flow control : analyse le contrôle et le suivi du code

Pour trouver des bugs, on peut faire du dynamic testing, de multiples tests sur differentes conditions ou utiliser un groupe de qualité utilisation.

2 Software vulnerability

Dans l'exemple de code, on a deux petites erreurs :

- *strcpy* : il faut vérifier que le buffer de destination corresponde avec celui de départ.
- *sizeof et srtlen* : le premier ne prend pas en compte le NUL en fin de chaîne alors que le second le prend en compte.

Dans le second code, on retrouve les erreurs suivantes :

- Dans le second *if*, il faudrait désaouller la mémoire de la première allocation.

Dans le troisième code, on retrouve les erreurs suivantes :

- Si length est égal à 0x7FFFFFFF, quand on fait le +1 dans le *if*, on passe alors à 0x80000000 qui est un nombre négatif.

La fonction sprintf est a utiliser de différentes façons. La première utilisation est à préserver.
« %.nombre s » permet d'afficher le nombre de caractères.

3 TP 1 : Buffer overflow

Ici, on va faire des stack base buffer overflow.
La pile est une *FILO*[1].
Prolog = initialisation de la mémoire : Avant même de faire le prolog, le programme va sauvegarder la prochaine instruction (eip)

- push de la valeur de ebp → sauvegarde de la valeur actuelle du pointeur.
- move ebp, esp. On fait pointer epb sur esp.
- sub esp, xxx.

Ainsi, entre ebp et esp, nous allons retrouver l'ensemble de nos variables locales définie dans la fonction.
Epilog = restauration de l'espace mémoire :

- move esp ebp
-
- ret (move eip) → retourne à la fonction main

1. First In, Last Out

4 ...

L'option de compilation */NXCOMPAT* sa spécifier qu'à chaque emplacement mémoire du programme ne sera pas exécutable.
Avec l'*ASLR*, pour que l'on soit sur d'avoir toujours à peu près les variables dans le même espace, on va remplir la mémoire jusqu'à laisser un seul espace mémoire pour le programme cible. Ainsi, le programme se lancera toujours dans le même espace mémoire. On pourra ainsi retrouver les adresses de manière plus simple.