EJCP 23
Marie-Laure Potet, Laurent Mounier

Code sécurisé : reverse, attaque, durcissement

Outils de désassemblage

Pour regarder le code produit par la compilation on pourra utiliser plusieurs outils :

- Objdump (objdump -S)
- Ghidra : déassemblage + décompilation (- :
  commande : ghidraRun
  A faire :
  - Créer un nouveau projet : File puis New Project puis Non-Shared Project
  - Donner un nom à Project Name (ex : TP-secu)
  - Puis cliquer sur codebrowser (l'icone en forme de dragon. . .)
  - Importer le fichier exécutable souhaité au projet ex : File puis import file puis nom-du-binaire
  - Lancer l'analyse du fichier et dans Symbol Tree puis functions, chercher la fonction qui nous intéresse

Exercice 1: Optimise

Regarder le code source de `optimise.c`.

▷ **Question 1** *Compiler et exécuter pour chacun des cas suivants. Justifier les résultats obtenus pour chacun de ces cas :*

no option :
```
gcc -o optimise1 optimise.c
```
optimisation option :
```
gcc -O2 -o optimise2 optimise.c
```
overflow detection option :
```
gcc -fno-strict-overflow -o optimise3 optimise.c
```
optimisation and overflow detection option :
```
gcc -O2 -fno-strict-overflow -o optimise4 optimise.c
```

Un lien sur les options de compilation :

https://gcc.gnu.org/onlinedocs/gcc-4.9.1/gcc/Optimize-Options.html

L'option `-fno-strict-overflow` *a pour effet de désactiver les optimisations liées au comportement indéfini sur les overflows.*

▷ **Question 2** *Proposer une solution pour rendre cette fonction sécurisée en utilisant les préconisations données ici :* `https://www.securecoding.cert.org/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow`

Exercice 2: Winloose

Regarder le code du programme `winloose.c`. Ce programme prend 2 entiers sur la ligne de commande (`argv[1]` et `argv[2]`).

▷ **Question 1** *Compiler ce programme avec la commande suivante :*

```
gcc -fno-stack-protector -o winloose winloose.c
```

Ce programme peut mener à différents résultats :

- *print* `"You loose"`
- *infinite loop*
- *crash*
- *etc.*

Trouver les entrées qui permettent d'imprimer `"You win"` *ou de provoquer une boucle infinie !*

Expliquer les résultats obtenus en dessinant la pile à l'exécution.

▷ **Question 2** *Désassembler le programme. Regarder le code du main et retrouver le placement des variables locales dans la pile.*

Indication : *dans une architecture 64-bits le frame pointeur s'appelle* `rbp` *et le pointeur de sommet de pile* `rsp`*.*

▷ **Question 3** *Compiler maintenant le programme* `winloose.c` *avec l'option "stack protection" :*

```
gcc -fstack-protector -o winloose winloose.c
```

Exécuter le programme avec les entrées précédentes. Que se passe-t-il ?

Désassembler le programme et comprendre comment le mécanisme de protection de la pile est implémenté.

Exercice 3: Fonction bannie -IspasswordOK

▷ **Question 1** *Regarder en détail la spécification de fgets :*
http ://www.cplusplus.com/reference/cstdio/fgets/

Quels sont les différents comportements possibles ?

▷ **Question 2** *Tester le programme* `fgets.c` *et trouver une attaque qui permet de s'authentifier sans rentrer 2 fois le bon mot de passe. Rappel : au terminal on peut encoder la fin de fichier par contrôle D*

Que se passe t-il pour que l'attaque fonctionne ?

▷ **Question 3** *Corriger le programme en prenant en compte les cas d'erreurs. On pourra s'inspirer du site suivant :*

https://wiki.sei.cmu.edu/confluence/display/c/ERR33-C.+Detect+and+handle+standard+library+errors

▷ **Question 4** *Une préconisation en sécurité consiste à nettoyer les mémoires. Compléter votre solution. Est-elle robuste aux optimisations ?*

▷ **Question 5** *Soit le programme* `exploit2-fixed-fgets.c` *qui teste un appel à IsPasswordOK. Montrer que ce programme est potentiellement vulnérable en changeant la valeur de* `size`.

Un site qui explique les problèmes de lecture :

https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/16993-securisez-la-s

Exercice 4: Un mauvais shell

▷ **Question 1** *Regarder le fichier de commande* `shell1` *et l'exécuter avec différentes entrées.*

Quelle est sa faiblesse ? Comment le rendre plus robuste ?

Exercice 5: Un UseAfterFree

Cet exemple montre une exploitation d'un Use After Free.

▷ **Question 1** *Comprendre le programme* `uaf2.c`*, détecter le use after free et comment un attaquant peut faire exécuter son propre code.*

▷ **Question 2** *Executer les commandes du fichier exploiter-uaf2.txt.*