## *0. Level 0*

On se connecte via SSH ; comme d'habitude, un **checksec** est lancé à la connection. On a les protections suivantes :

- **NX enabled** : pas d'exécution de shellcode sur la stack.
- **Partial RELRO** : cette protection n'est pas très utile. Ce tutoriel explique très bien à quoi sert RELRO, et la différence entre partial / full : https://www.redhat.com/en/blog/hardening-elf-binaries-using-relocation-read-only-relro

On reste sur des binaires 32 bits ; toujours pas de code en **PIE** ; **pas d'ASLR**.

Le code source est vraiment simple à reproduire, avec une fonction main très basique :

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

int     main(void)
{
    int     n;

    puts("***********************************");
    puts("*     -Level00-         *");
    puts("***********************************");
    printf("Password:");

    scanf("%d", &n);
    if (n == 0x149c)                // 5276 in decimal
    {
        puts("\nAuthenticated!");
        system("/bin/sh");
        return (0);
    }
    puts("\nInvalid Password!");
    return (1);
}
```

On comprend vite qu'il suffit de donner à la fonction **scanf** la valeur 5276, afin de la placer dans la variable **n,** et de déclencher l'appel à **system**.

>> Exploitation manuelle

```
$ echo '5276' > /tmp/payload && cat /tmp/payload - | ./level0
```

>> Exploit automatisé

```python
from pwn import *

s = ssh(host='192.168.1.3', port=4242, user='level00', password='level00')
p = s.process("/home/users/level00/./level00")

p.recvline(4)
```

```
p.sendline('5276')
p.interactive()
```

**Got flag**.