DESCRIPTION

Connect to 141.85.224.106:31344 and get the flag. ASLR is on.

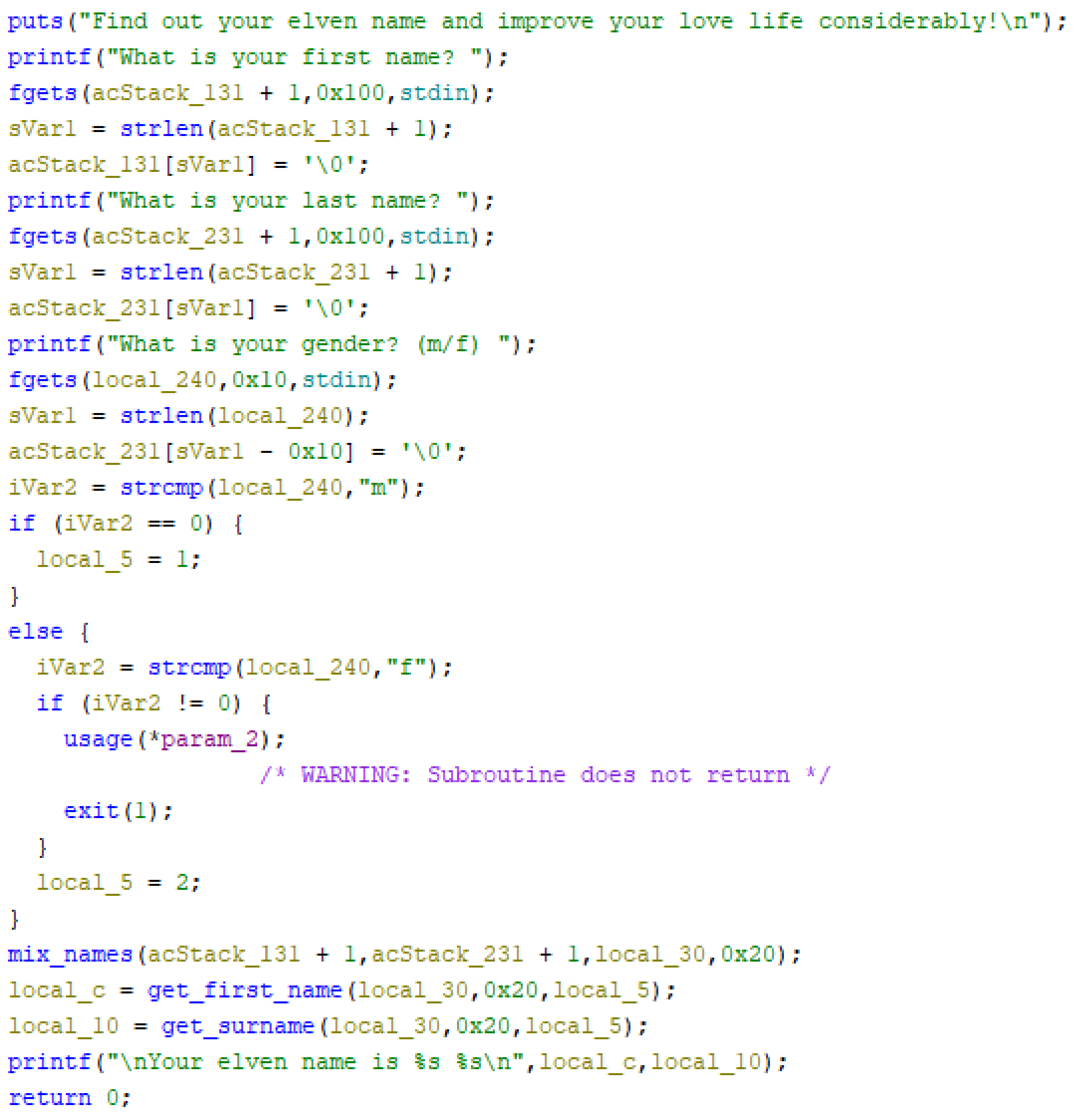
RESOURCES

As part of the challenge I received an executable file called **elven_godmother** as an attachment for analysis as well as a **libc** version used for compilation.

APPROACHES

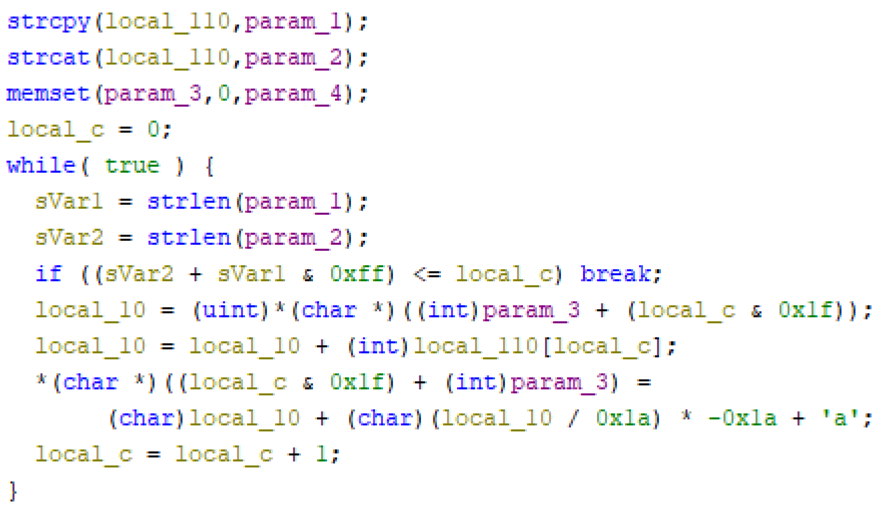
1. The first approach here was to run the program and after running it I could see that it is requiring an input consisting of a first name a last name an a gender (m/f). Thus, I gave some inputs and I received some other strings as output. My guess was that there might be an overflow when reading the inputs so I decompiled it using Ghidra.

2. Thus, the decompilation looks like this:



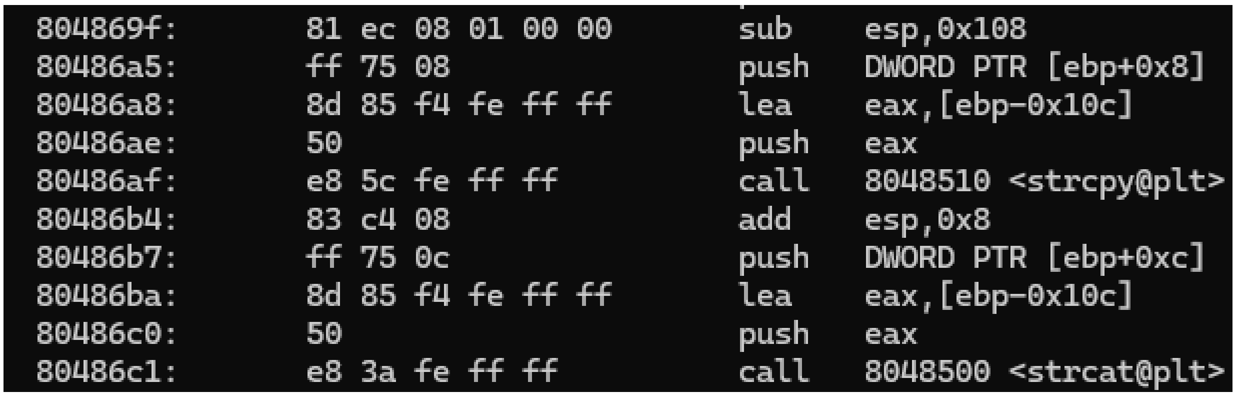
```
puts("Find out your elven name and improve your love life considerably!\n");
printf("What is your first name? ");
fgets(acStack_131 + 1,0x100,stdin);
sVar1 = strlen(acStack_131 + 1);
acStack_131[sVar1] = '\0';
printf("What is your last name? ");
fgets(acStack_231 + 1,0x100,stdin);
sVar1 = strlen(acStack_231 + 1);
acStack_231[sVar1] = '\0';
printf("What is your gender? (m/f) ");
fgets(local_240,0x10,stdin);
sVar1 = strlen(local_240);
acStack_231[sVar1 - 0x10] = '\0';
iVar2 = strcmp(local_240,"m");
if (iVar2 == 0) {
  local_5 = 1;
}
else {
  iVar2 = strcmp(local_240,"f");
  if (iVar2 != 0) {
    usage(*param_2);
                  /* WARNING: Subroutine does not return */
    exit(1);
  }
  local_5 = 2;
}
mix_names(acStack_131 + 1,acStack_231 + 1,local_30,0x20);
local_c = get_first_name(local_30,0x20,local_5);
local_10 = get_surname(local_30,0x20,local_5);
printf("\nYour elven name is %s %s\n",local_c,local_10);
return 0;
```

3. We can see here that there are two reads for the first and last name and one read for the gender, all of them using **fgets** and properly sized buffers so I could not find vulnerabilities so I went forward to check the next interesting function called **mix_names**. It's decompilation can be seen below:



```
strcpy(local_110,param_1);
strcat(local_110,param_2);
memset(param_3,0,param_4);
local_c = 0;
while( true ) {
  sVar1 = strlen(param_1);
  sVar2 = strlen(param_2);
  if ((sVar2 + sVar1 & 0xff) <= local_c) break;
  local_10 = (uint)*(char *)((int)param_3 + (local_c & 0x1f));
  local_10 = local_10 + (int)local_110[local_c];
  *(char *)((local_c & 0x1f) + (int)param_3) =
       (char)local_10 + (char)(local_10 / 0x1a) * -0x1a + 'a';
  local_c = local_c + 1;
}
```

4. Here we can see that the first two parameters which are the first and last name read before concatenated in the same local variable. If we remember, each of them could have been at maximum size 0x100 (256 characters), so that would be a **512 total**. However, if we look in the assembly code, we can see that the size of the local parameter in which they are copied has a maximum capacity **of 268 characters** so there is the vulnerability.



```
804869f:       81 ec 08 01 00 00       sub    esp,0x108
80486a5:       ff 75 08                push   DWORD PTR [ebp+0x8]
80486a8:       8d 85 f4 fe ff ff       lea    eax,[ebp-0x10c]
80486ae:       50                      push   eax
80486af:       e8 5c fe ff ff          call   8048510 <strcpy@plt>
80486b4:       83 c4 08                add    esp,0x8
80486b7:       ff 75 0c                push   DWORD PTR [ebp+0xc]
80486ba:       8d 85 f4 fe ff ff       lea    eax,[ebp-0x10c]
80486c0:       50                      push   eax
80486c1:       e8 3a fe ff ff          call   8048500 <strcat@plt>
```

5. Another interesting observation here would be that the binary is on 32bits so all the parameters for a function are going to be placed on the stack. The exploit would be here to again call **puts@plt(puts@got)**, find the address of **puts** in **libc** and having the **libc** version, we could determine the address of any function in **libc** and we would **call system(„/bin/sh")**.
6. Thus, the tactics was the following, full the first buffer with 0x100 non-relevant characters as well as the next characters from the second buffer (last name) until all the 268 characters are full from the buffer in which they are copied and the **old_ebp** as well. Then, we put the return address as the address of **puts@plt** and the return address of **puts@plt** to be **main** and the parameter of **puts@plt** to be **puts@got**.
7. Then, at the second run of main we do the same overflow just that the function called is going to be **system(„/bin/sh");**
8. So, if you now run the script (**python3 script.py**) then we will obtain a shell on remote and if we run **cat /home/ctf/flag**, we will get the flag for this current task:

CNS_CTF{7d68968119deb2413ab2041f698aa6d6}